THE UNIDEX® 600 SERIES
USER'S GUIDE

P/N: EDU157 (V1.4)

AEROTECH, Inc. • 101 Zeta Drive • Pittsburgh, PA. 15238-2897 • USA
Phone (412) 963-7470 • Fax (412) 963-7459
Product Service: (412) 967-6440; (412) 967-6870 (Fax)

www.aerotechinc.com

If you should have any questions about the UNIDEX 600 or comments regarding the documentation, please refer to Aerotech online at:

http://www.aerotechinc.com

For your convenience, a product registration form is available at our web site.

Our web site is continually updated with new product information, free downloadable software and special pricing on selected products.

The Aerdebug User's Manual Revision History:

Rev 1.0	October 14, 1996

The UNIDEX 600 User's Guide (previously the AerDebug User's Manual) Revision History:

Rev 1.1	May 5, 1997
Rev 1.2	September, 1997
Rev 1.3	July 10, 2000
Rev 1.4	December 5, 2000

TABLE OF CONTENTS

∇ ∇ ∇

LIST OF FIGURES

∇ ∇ ∇

LIST OF TABLES

∇ ∇ ∇

PREFACE

This section gives you an overview of topics covered in each of the sections of this manual as well as conventions used in this manual. This manual contains information on the following topics:

CHAPTER 1: INTRODUCTION

This chapter contains an introduction to the hardware and software architecture of the UNIDEX 600 Series motion controllers.

CHAPTER 2: GETTING STARTED

This chapter contains information about the minimum software requirements for proper operation. It also contains information on the system setup and installation of the UTIL600 software.

CHAPTER 3: PROGRAMMING

This chapter provides an overview of the two available programming interfaces and the trade-offs between using one or the other to program the U600 Series Controller. The correct interface or combination of interfaces the programmer should use depends on the target application.

CHAPTER 4: AERDEBUG

This chapter contains information about the AerDebug command line oriented program that can be used for examining or controlling the UNIDEX 600 Series axis processor card. Additional information includes description of the AerDebug screen.

CHAPTER 5: AERTUNE

This chapter supplies instructions on tuning an axis and using the AerTune program, a utility for visually observing and fine-tuning the performance of the motion generated by a UNIDEX 600 Series Controller.

CHAPTER 6: AERPLOT

This chapter provides information on the AerPlot program that allows the user to display a mix of up to 16 axes and/or analog user input information from the UNIDEX 600 series controller card in a visual format with a user definable time base reference.

CHAPTER 7: AERSTAT

This chapter contains information about the AerStat utility, a debugging tool that displays the status of all 16 axes of the controller.

CHAPTER 8: AERREG

This chapter contains information about AerReg, Aerotech's operating system registry editor program that allows registry information to be created or edited by the user.

CHAPTER 9: AERPLOT3D

This chapter contains information about AerPlot3D. This utility program allows for continuous plotting of three axes of motion during a CNC program.

CHAPTER 10: AERPLOTIO

This chapter contains information about AerPlotIO. This untility program plots I/O and Registers vs. Time.

CHAPTER 11: FILTER

This chapter contains information about the Filter utility. This program calculates values for the second order of digital filter.

CHAPTER 12: SETUP WIZ

This chapter contains information about the Setup Wiz utility. This program allows you to configure your controller with English text prompts for required parameters.

CHAPTER 13: PRMSETUP

This chapter contains information about the PrmSetup utility. This program will install the pre-configured .Ini files onto your PC.

APPENDIX A: GLOSSARY OF TERMS

Appendix A contains a list of definitions of terms used in this manual.

APPENDIX B: TROUBLESHOOTING

Appendix B contains information on troubleshooting the most common problems when using the UNIDEX 600 Series motion controllers.

APPENDIX C: PARAMETERS

Appendix C contains all the available parameters used by the UNIDEX 600 Series controllers.

APPENDIX D: WARRANTY AND FIELD SERVICE

Appendix D contains the warranty and field service policy for Aerotech products.

INDEX

The index contains a page number reference of topics discussed in this manual. Locator page references in the index contain the chapter number (or appendix letter) followed by the page number of the reference.

CUSTOMER SURVEY FORM

A customer survey form is included at the end of this manual for the reader's comments and suggestions about this manual. Reader's are encouraged to critique the manual and offer their feedback by completing the form and either mailing or faxing it to Aerotech.

Throughout this manual the following conventions are used:

- The terms UNIDEX 600 and U600 are used interchangeably throughout this manual
- The text <ENTER> is used to indicate that the Enter/Return key on the keyboard is to be pressed.
- Hexadecimal numbers are listed using a preceding "0x" (for example, 0x300, 0x12F, 0x01EA, etc.,) to distinguish them from decimal numbers
- Graphic icons or keywords may appear in the outer margins to provide visual references of key features, components, operations or notes.
- This manual uses the symbol "∇ ∇ ∇" to indicate the end of a chapter.

Although every effort has been made to ensure consistency, subtle differences may exist between the illustrations in this manual and the component and/or software screens that they represent.

∇ ∇ ∇

CHAPTER 1: INTRODUCTION AND OVERVIEW

In This Section:

1.1. Introduction

The UNIDEX 600 Series controller is a PC based ISA bus controller. The basic software package includes three items: a RS-274 G-code compiler, library routines and utility programs. The RS-274 compiler and library routines program the controller, and are described in Chapter 3. The utilities set up, troubleshoot, and operate the controller, and are described in Chapter 4 through Chapter 8. Chapter 2 is devoted to the information needed to install, setup, and test the controller.

Refer to Section 1.7. for details on additional software, including U600 MMI; a Windows NT GUI interface, that can be purchased to ease operation of the U600 Series controller. If the U600 MMI was not purchased, then the utilities described in Chapter 4 through Chapter 8 must be used to set up the controller.

1.2. Installation

Figure 1-1 flowcharts an overview of the entire installation process. Chapter 2 contains detailed information on the installation process.

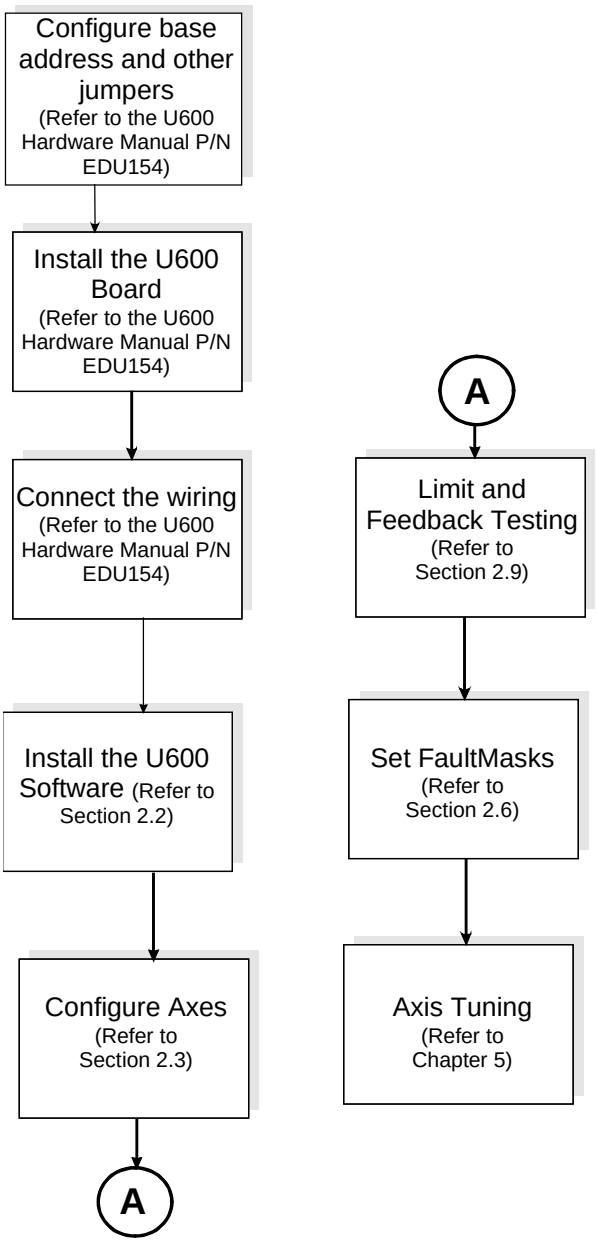


Figure 1-1. Installation Process

1.3. Programming

The U600 Series Controllers offer two independent programming interfaces: Library Called, and RS-274 G-code. The correct interface or combination of interfaces the programmer should use depends on the target application, therefore, the programmer must understand the fundamentals of both in order to make the correct decision. Refer to Chapter 3: Programming, for more details.

1.4. Architecture Overview

A UNIDEX 600 Series controller installed in a PC acts as a dual processor system. The PC's CPU is one processor and the UNIDEX 600 card (referred to as the axis processor) is another independent processor.

Although the axis processor is one processor, through the use of polling and interrupts, it acts as three separate execution units, called the library servicer, the CNC engine, and the motion controller (refer to Figure 1-2).

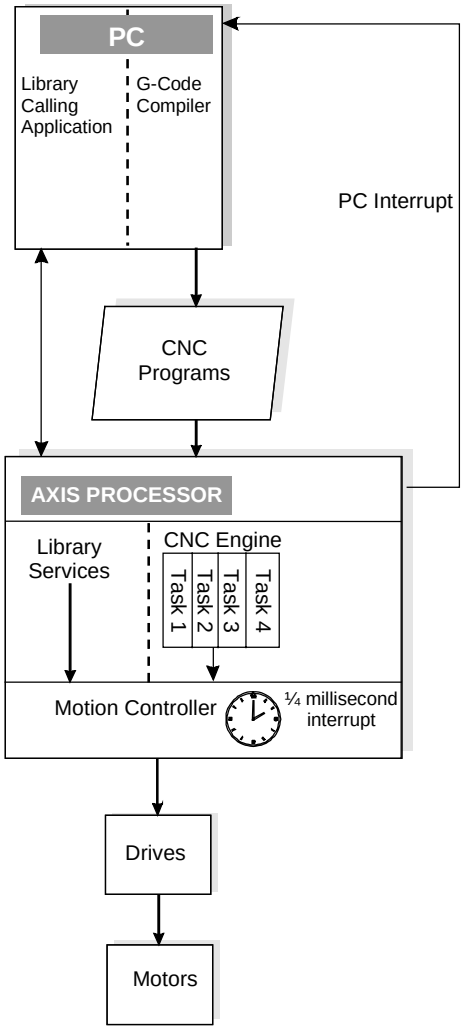


Figure 1-2. System Architecture

1.4.1. Axis Processor

The motion controller has highest priority and is run off of an internal interrupt tied to a clock. The CNC engine and library servicer run at equal priority to each other; running in whatever time remains after the motion controller completes. In pseudo-code, the execution is:

```
DO forever
    for i=1 to 16 axes
        Run ilibrary servicer for axis i
    end for
    for i=1 to 4 tasks
        Run a CNC command line for task i
     end for
  end do
```

The Global parameter *AVGPOLLTIMESEC* will read out the average time it is taking to run one iteration of the forever loop shown above (averaged over the last 100 loop iterations). As the motion controller becomes increasingly loaded down running the servo loop, the value of *AVGPOLLTIMESEC* increases.

1.4.1.1. The Motion Controller Execution Unit

The motion controller's execution unit's main function is to execute the servo loop that reads the feedback signals and generates the velocity and position commands for the drives. It is triggered by an interrupt, that normally fires once every 1/4 milliseconds (refer to Chapter 5: AerTune for more details on the servo loop). However, this can be readjusted to every 1 millisecond with the *Enable1KhtzServo* global parameter.

The motion controller execution unit also executes other high priority functions requiring tight loops in addition to the servo loop, refer to Table 1-1. These actions are always performed every millisecond (every fourth interrupt). None of these actions are performed unless they are specifically activated.

Table 1-1. Other Actions Performed Every Millisecond

Action	For more information see
Error Map Correction	EDU 156, under AerAxisCal functions
Strip Chart Data Gathering	EDU 156, under AerStrip functions
Monitoring for Axis Faults	EDU 157 under Fault Masks
Digital Probe monitoring	EDU 158, under *PROBE* command
Increment Clock	EDU 157, under *CLOCK* axis parameter
Read analog MFO/MSO	EDU 157 under *AnalogMFOInput* task parameter

1.4.1.2. Library Servicer

This execution unit exists to service library function calls (functions in the form of Aerxxx()) from C, C++, or Visual Basic. It only executes in the time remaining after the motion controller completes its activities. The library calling interface is described in more detail in Chapter 3.

1.4.1.3. CNC Engine

This execution unit runs CNC programs. It executes only in the time remaining after the motion controller completes its activities. A CNC motion command, like a G1, sets up the motion controller execution unit with the parameters it needs to run the motion. The motion command then waits (will not execute the next CNC command) until the motion controller indicates the motion ended. The CNC engine consists of four tasks that can run four CNC programs independently. Chapter 3 describes the CNC interface in more detail.

1.4.2. The PC

The PC communicates to the axis processor via Aerxxx() library function calls (where "xxx" is some name). Data can be passed both to and from the axis processor via these calls. Two special classes of library functions: AerCompilrxxx() and AerTaskxxx() can be used to control the CNC engine execution unit.

The axis processor returns data to the PC through return variables in the library calls or through PC interrupts.

1.4.2.1. The PC Interrupt

The Axis processor generates PC interrupts in two cases: Faults (axis or task) and Callbacks. The U600MMI or any other Aerotech utility software running on the PC does not use fault interrupts; they are for the convenience of the application programmer, if they desire to use them (see the AerEventxxx functions). Callback interrupts are used in a number of CNC statements, their purpose is to have the axis processor instruct the PC it needs to do something to support the execution of a CNC program. For example, **FILEWRITE** and **DISPLAY** CNC statements require the use of the PC, even though the axis processor executes these statements. Another CNC statement of interest that uses callback interrupts is the **CALLDLL** statement, that allows the programmer to run C or C++ functions that they write in DLLs from a CNC program.

Chapter 2 discusses the causes and effects of Task and Axis faults and the *UNIDEX 600 Series CNC Programming Win NT/95 Manual, P/N EDU158* under the Extended Commands covers in more detail callback interrupts.

The axis processor will not wait for any response to an interrupt, it simply continues on. If no one picks up the interrupt, it has no effect. The U600MMI will respond appropriately to interrupts (either displaying the fault, or executing the callback), but AerDebug does not respond to interrupts. In order to respond to interrupts from the user's own application, the user must call the AerEventxxx() library function.

1.5. Motion

This section summarizes the basic features of motion generated by the U600 controller.

1.5.1. The Servo Loop

Translated into velocity and position commands are the programmer's commands. The controller obtains a new velocity and position command every millisecond. These commands are input to the servo loop, which generates the motor torque signals sent out to the amplifiers. The servo loop is responsible for reading the commands and the feedback (the actual position and speeds) and adjusting the motor torques in order to reduce the position and velocity errors. The definition of error is commanded minus the actual.

The servo loop normally operates off of a 1/4 millisecond interrupt. However, the user can slow this interrupt down to 1 millisecond, with the *Enable1KhtzServo* global parameter. This action may be necessary if there is too much processing off the interrupt potentially starving the forever loop (see Section 1.4.1).

The settings of various axis parameters adjust or tune the servo loop activity. Section 5.3 of Chapter 5: AerTune, summarizes the relevant axis parameters and their use.

1.5.2. Generating Motion

In general there are three types of motion available for a single axis: synchronous, asynchronous, and master-slave (sometimes called camming). Please see the appropriate programming manual, (EDU156 for the library interface, and EDU158 for the CNC interface) for complete details on how to invoke these types of motion, the rest of this section summarizes this information.

Table 1-2. Available Motion Types

Motion Type	Library Interface Access	CNC Interface Access
Synchronous	NONE AVAILABLE	G0, G1, etc. CNC commands
Asyncronous	AerMovexxx() functions	STRM etc. CNC commands
Cam table	AerCamTablexxx() function	NONE AVAILABLE

When executing synchronous motion, the controller does not move to the next step in the program until the motion finishes and the axes are in the commanded position.

In asynchronous motion, the motion is initiated, but the controller immediately moves to the next step in the program. The controller does not wait for the motion to complete.

Asynchronous motion offers more versatility, allowing the user to perform other tasks during a time consuming move. However, asynchronous moves are potentially more dangerous, since the programmer is responsible for making sure that the first move

finishes before a following command occurs which requires the first move to be complete. For example, a slow carriage move might need to be completed before a part loader can approach the axis or a collision occurs.

Cam Table (or master-slave) motion allows the programmer to direct that a particular axes position be dependent on another axes position or velocity. The programmer downloads a set of values into the axis processor (960) that dictate the required position or velocity of a slave axis, given another axes' (the master's) position. When in-between the dictated master positions, the controller interpolates linearly or by cubic spline to obtain the correct slave value for that master axis position. This type of movement is the most flexible, allowing the programmer to completely dictate any position or speed profile to the motion. The programmer can also achieve multi-axis synchronized movement by slaving multiple axes to the same master. Finally, this type of motion can be used to dictate motion based on analog input signals. See the AerCamTablexxx library functions for more details on camming motion.

1.5.3. Monitoring Motion

The user can monitor all sorts of axis processor status, including what stage the movement is in. Please see the *Status1*, *Status2*, *Status3*, and *Mode1* Task parameters in Appendix C for details on availability. The CNC interface allows the programmer to specify code to be executed when certain conditions occur. Also, see Chapter 2 section on Faults for more monitoring capabilities.

1.5.4. Multi-Axis Motion

In any of the three motion types, the user can direct that multiple axes be moved simultaneously. However, there are two types of simultaneous motion: contoured, and non-contoured. In both types the involved axes begin movement at the same time. In contoured movement the speed of the axes is coordinated so that all the axes complete their movement at the same time. Contoured movement follows specific controlled paths through the space defined by the axis through which it moves. In non-contoured moves, each axis follows its own dictated speeds independently, and may finish their movement at different times.

All synchronous and asynchronous motion is non-contoured, except certain motion available from the CNC: **G1, G2, G3**. The term contoured does not apply to cam directed motion, since the tables determine the intra-axis synchronization.

1.5.5. Motion Resolution

The servo loop normally runs 4 cycles per millisecond, translating velocity and distance commands into motor torques. Normally, the axis processor generates new velocity and position commands every millisecond, which rely upon the user-specified profile. However, in synchronized CNC motion (**G1,G2**) the points lying on the user-specified profile are generated every 10 milliseconds and the controller uses cubic splines to generate 1 millisecond values lying between the user-specified points.

1.6. Faults

Faults are generated by the UNIDEX 600 when errant or unusual conditions occur. There are many conditions that can cause faults, and through the use of Axis and Task parameters the programmer can impose a wide variety of actions to occur from faults – so faults is not a simple subject. What follows is a short summary; Chapter 2, under Axis Faults and Task Faults provides more detail. There are two types of faults: task and axis, refer to Table 1-3.

Table 1-3. Differences between Task and Axis Faults

	Axis Fault	Task Fault
Example	Axis Error (e.g. No feedback from drive)	CNC line error (e.g. "$GLOB1=7/0")
Subsequent Action	Controlled by bits (except "task fault" bit) in FAULTMASK, HALTMASK etc.	Controlled by "taskfault" bit in FAULTMASK, HALTMASK etc.
Viewing in U600MMI	Flashes in red over position display on Run/Manual Screen	Bottom Right of Run/Manual Screen
Viewing in AerDebug	PARMGET A FAULT	PARMGET T TASKFAULT
Resetting in U600MMI	Hit Fault acknowledge button	Hit Fault acknowledge button
Resetting in AerDebug	PARMSET A FAULT –1	PARMSET T TASKFAULT 0
Generates an Interrupt to PC	Depends on INTMASK setting	Yes
Scanned	Every millisecond	Every time CNC program executes a line (see Section 1.4.1)

1.6.1. Axis Faults

The UNIDEX 600 Series controllers monitor a large range of axis conditions (such as feedback error or too large position error) on a once per millisecond basis. When the axis processor finds that the condition is true, a fault occurs. The programmer can determine if one or more actions take place when these conditions occur, such as halting or aborting axis motion, disabling drives, setting digital outputs and generating interrupts back to the PC. The user can customize both the conditions to observe and the actions to take independently for each axis, simply by setting axis parameters; no programming code is necessary. The relevant axis parameters are: *FAULT*, *FAULTMASK*, *HALTMASK*, *ABORTMASK*, *DISABLEMASK*, *AUXMASK* and *INTMASK*. Refer to Chapter 2: Getting Started, under Fault Masks for a detailed description and examples.

1.6.2. Task Faults

Task faults occur when a CNC program performs an illegal action, such as a division by zero. Task faults stop the program running on the CNC task. The CNC programmer can override this (keep the program running) with the **ONGOSUB** CNC command. If the program was running a **G1** or other synchronous motion command at the time, then that motion stops. However, asynchronous motion is not normally halted by a Task Fault. To do this the user must use the TaskFault bit of the *FAULTMASK* Axis parameter.

The user can customize other actions to occur on a task fault by using the Axis masks *FAULTMASK*, *HALTMASK* etc. The bits of these parameters (see Chapter 2 under Faults) normally represent different axis faults, however, one bit called taskfault is used when encountering any task fault.

For example, if the taskfault bit is set for *ABORTMASK* of axis 1, then a task fault causes axis 1 to abort its motion.

Unlike axis faults, the user cannot impose different actions based on different types of task faults.

1.7. Option Ordering Information

Software options available, in addition to the base package, are shown in Table 1-4.

Table 1-4. Available Software Options

Part Number	Description
MMI600-NT	CNC MMI for Windows NT/95
SDK600-NT	Software Development Kit for Windows NT/95
MMISRC600-NT	Source Code for Aerotech's MMI600-NT CNC Application
CIMLITE	Computer Integrated Manufacturing Software
CIMCAD	Computer Integrated Design & Manufacturing Software
Custom	Turnkey Applications per Customer Requirements

Refer to the U600 hardware manual (EDU154) for information on hardware options.

∇ ∇ ∇

CHAPTER 2: GETTING STARTED

In This Section:

2.1. Introduction

Figure 2-1 is a flowchart providing an overview of the installation process from installing the software through preliminary servo loop tuning. If a complete system was purchased from Aerotech (with rotary and/or linear positioning tables), the configuration of the system was done at the factory. If drives and motors were only purchased, the axes will be configured for the respective type of motor, but the axes will most likely require tuning depending upon the load placed on the motor. Also, an Engineering Specification (ES) is provided along with the documentation package indicating the resolutions and maximum speeds of the axes.

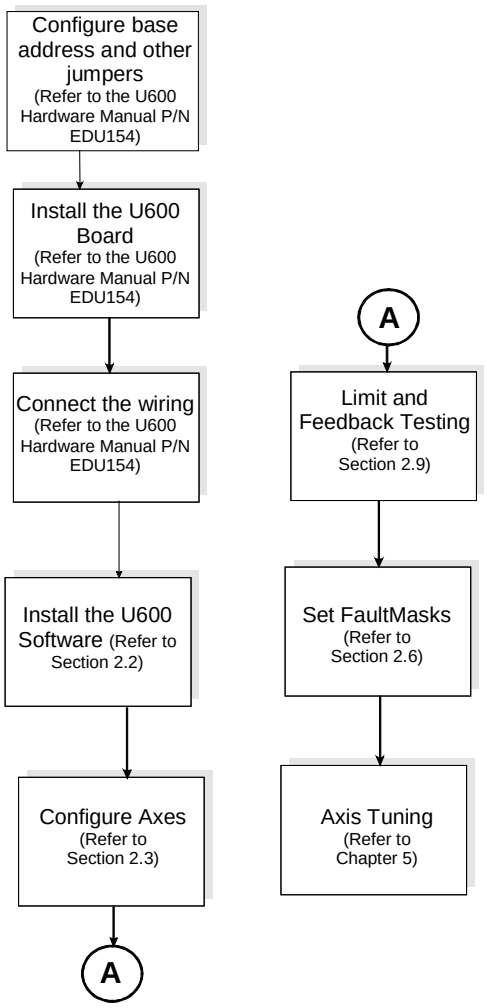


Figure 2-1. Flowchart Overviewing the Installation/Configuration Process

2.1.1. Minimum Requirements

Minimum requirements and recommended system configurations for the software installation are shown in Table 2-1. The free disk space requirements for specific software is shown in Table 2-2.

Table 2-1. Minimum Requirements and Recommendations

Minimum Requirements	Windows 95	Windows NT
PC Speed	90 MHz	90 MHz
RAM (per MS specs)	16 Megabytes	24 Megabytes
Graphics Display	800x600	800x600
OS version	4.0+	4.0

Table 2-2. Free Disk Space Requirements

Software	Part Number	Free Space
Utilities (required)	UTIL600-NT	10 Megabytes
CNC MMI Application	MMI600-NT	5 Megabytes
Software Development Kit	SDK600-NT	5 Megabytes
CNC MMI Source Code	MMISRC600-NT	1 Megabyte

The CNC MMI and the Software Development Kit share most of the files between the two applications, requiring approximately 5 megabytes of free disk space to install one or both applications.

2.1.2. Fundamentals

This section addresses some of the basic concepts and terms used in UNIDEX 600 setup and programming.

2.1.2.1. Parameters

Nearly all of the operational details of the UNIDEX 600 and the status information available to the user are controlled and presented by parameters. This manual uses *italics* to present the parameter names. Appendix C contains the descriptions of all parameters.

There are four types of parameters: Axis, Machine, Task, and Global. Some parameters are read-only (like raw position) and are for monitoring status. The user can set/view parameters through the U600MMI, or through AerDebug using the **PARMSET** and **PARMGET** commands. In AerDebug, the user can also use the **PARMMON** command to continuously read a parameter's value. See examples below (don't type in text after ";" these are only comments):

```
AX 2                    ; Tells AerDebug to look at axis 2 (it shows
                        ; the current axis in the prompt)
PARMSET A DRIVE 1       ; Enables drive of axis (A means axis
                        ; parameter
PARMGET M TYPE          ; Looks at type of current axis ("M" means
                        ; machine parameter
PARMMON A POS           ; Continuously monitors the position of
                        ; the current axis
```

Axis and Machine parameters apply to an individual axis. Axis parameters are always integers, and if distances or times apply, axis parameters are in counts and milliseconds. Machine parameters also apply to a particular axis, but they are floating point, or decimal values. Machine parameter distances and times are in user units and seconds.

Task parameters are only relevant to the CNC G-code interface and define information used by that interface, such as the speed of **G0** moves.

Global parameters are a collection of miscellaneous parameters relating to the controller as a whole; such as servo update rate.

2.1.2.2. Bit Masks

Many parameters are bitmasks, most notably the fault masks. A bitmask is a conglomerate of binary (zero or one) values. Each binary digit represents one value, which is accessed by or set by using the appropriate mask for that bit. The term "bit 0" (the first bit) always applies to the least significant bit, in other words, the bit whose mask is 1.

It is assumed that the reader is familiar with how to use hexadecimal masks and bitwise ORs, ANDs and NOTs to read and set bitmask values.

Table 2-3. Hexadecimal values of Bit Numbers

Bit Number	Mask (hexadecimal value)
0	0x1
1	0x2
2	0x4
…	…

Table 2-3 indicates the hexadecimal values are preceded by a "0x". This is the convention used throughout this and all other UNIDEX 600 Series manuals. However, the syntax for specifying hexadecimal numbers within the UNIDEX 600 Series programming language is, 0h (i.e.; 0hF [15 decimal]).

2.1.2.3. Faults

Faults are the mechanism that the UNIDEX 600 uses to detect errant conditions. It is crucial that the fault values and masks be setup properly before moving axes, in order to avoid damaging or destroying hardware. Fault masks are essential for safe and accurate operation of the UNIDEX 600 controller. Refer to Section 2.6. for details on faults.

2.2. Software Installation

2.2.1. UTIL600-NT Installation

Before the installation of the UTIL600-NT software package, the operator must ensure that the necessary hardware platform (as described earlier) and the Windows operating environment are in place. With these issues addressed, actual installation can begin.

The software installation process uses standard Windows installation techniques and is very easy to do. With even modest familiarity of the Windows environment, the entire process should take less than 5 minutes. The steps are outlined below.

1. Plug in the system PC and monitor into appropriate power outlets, then turn on the PC.
2. Insert UTIL600-NT software diskette number 1 into the floppy drive of the system PC (e.g., A:\).
3. Windows will start the UTIL600-NT startup program. From Windows95 (or Windows NT 4.0) select the Start button, then click the Run icon, type in A:\setup.exe in the Open box and Click OK.
4. A new pop-up window will be displayed. Enter the name of the target sub directory of the installation. By default, the SETUP.EXE program will install the UTIL600-NT software under the C:\U600 sub directory. If a different sub directory is desired, change the target directory from this pop-up window. If a new sub directory does not exist, the installation program will automatically create it.
5. Follow any subsequent instructions that are displayed on the screen.

When installation is complete, "UNIDEX 600" and "UNIDEX 600 MMI" entries are placed under the Programs menu, which appears after pushing the Start button. This indicates that the installation was successful and that the program is ready to be started.

When using the Windows NT operating system, the PC must be rebooted to load the device driver. Windows NT does not support dynamic loading.

2.2.2. MMI Software Installation

If the MMI600 interface was purchased, then install this now, in much the same way as the UTIL 600 was installed.

2.2.3. WinNT/Win95 Registration

The next step is to register the controller board setup parameters. This is only a new installation procedure of the UNIDEX 600 software, not when installing updates to existing UNIDEX 600 software. This will require three pieces of information:

- An available 15 byte block of I/O addresses within the PC's I/O address space,
- An available IRQ interrupt, and
- 16-kilobyte regions of unused high memory to map the memory window into the UNIDEX 600 controller's address space (an AT window). This might require changing the PC's CMOS settings to be sure that the selected memory window address range is not shadowed or cached by the PC.

The I/O address and the PC interrupt chosen must match the selection on the board via the jumpers. If the user did not change the jumpers, then they are set at the factory default positions. Refer to the *UNIDEX 600 Hardware Manual, P/N EDU154* under Getting Started for location of the jumpers and their default positions.

Use the AerReg program to set the registry values for the three items chosen.

2.2.4. Software Installation Testing

Run AerDebug to check the I/O address value. It will deliver an appropriate error message on startup if the I/O addresses do not match. If it comes up with no error message (shows the <TK1/AX1> prompt) then the I/O address and the AT window are correct. If the AT window is bad, choose another address (using AerReg) or disable RAM shadowing in the CMOS setup. If the I/O address is bad, choose another address using AerReg. Remember to reboot or restart the device driver when in Win NT after making a change in AerReg.

AerDebug also detects and delivers an error message for some incorrect PC interrupt values, but not all. To test the PC interrupt properly, run AerDebug, and if it does not indicate that the PC interrupt is bad, type "!TI". It will display "No Error", or give an appropriate message describing a problem with the PC interrupt.

The axis processor interrupt is distinct from the PC interrupt. The PC interrupt is used only in selected situations, by the axis processor, to alert the front end. It is only used in conjunction with Callback extended statements and the *INTMASK* axis parameter.

The axis processor interrupt is much more important. It operates the motion controller execution unit (see System Architecture in chapter 1 for details) which runs the servo loop. The axis processor interrupt is private to the axis processor card, and the user does not need to make any settings or adjustments for it to work. The user should test the axis processor interrupt from AerDebug by typing the command "PARMMON A CLOCK." The user should see a continuously increasing number (counting in milliseconds). If the value does not change, then the axis processor interrupt is deactivated or otherwise not operating. This could be due to a bad image file, or a bad jumper setting of JP2 or JP3 on the UNIDEX 600 PC based motion controller card.

2.3. Axis Configuration

If a complete system was purchased from Aerotech (with rotary and/or linear positioning tables), the configuration of the system is done at the factory. Also, an Engineering Specification (ES) is provided along with the documentation package indicating the resolutions and maximum speeds of the axes.

The easiest way to configure axes is with the U600MMI, in the axis configuration page, with the Wizard. The user can also configure axes with the AerDebug utility, with the CONFIG* commands. (refer to Chapter 4: AerDebug, under Axis and FaultMask Configurations).

Axes can be configured manually within AerDebug using the *CONFIG* series of commands. Axes may be configured with the **CONFIGREAD** command that reads an axis configuration from a file previously created with the **CONFIGWRITE** command. The user can store axis configuration commands in a file that can be replayed with the **PLAY** command. For more information on configuring axes with AerDebug, refer to Chapter 4: AerDebug.

The CNC MMI application (P/N MMI600-NT) can configure the axes as well by selecting the axis configuration page. For more information on configuring axes with the MMI application, refer to the *MMI600 online help file.* To configure an axis from the library programming interface, refer to the axis configuration functions: AxisConfigxxxx() in the *U600 Library Reference Manua,l P/N EDU156*. The axes cannot be configured from the CNC programming interface.

2.3.1. Servo Loop Modes

There are four types of servo loop configurations: virtual, torque mode, velocity mode closed loop, and velocity mode open loop. Setting the motor type to zero chooses virtual mode. The other modes are selected by entering the correct motor type and entering the correct gains (gains are axis parameters) as shown in Table 2-4. Keep in mind that the correct values for gains labeled "non-zero" or "any" below, must be determined by axis tuning (see section 2.11.)

By default the axes are setup as virtual axes (axes without motors attached). In virtual axes the actual position and velocity are set equal to the commanded position and velocity and the servo loop is not run. Virtual axes are often useful for debugging programs without motors, but in order to run motors the user must configure the axes for torque or velocity mode, depending upon the configuration of the servo amplifier and feedback device.

Table 2-4. Gains for the Different Servo Loop Modes

	KP	KI	PGAIN	VFF	AFF
Virtual	Any	Any	Any	Any	Any
Torque Mode (normal)	Non-0	Non-0	Non-0	Any	Any
Open Loop Velocity (tachometer)	0	0	Non-0	Any	0
Open Loop Velocity (spindle)	1	0	0	0	0

In torque and velocity modes the controller will output to each amplifier an analog voltage of 10 volts (typically bipolar, i.e. –10 to +10 volts) that varies over time. The difference is in how the output should be interpreted: in torque mode the output is proportional to motor torque; in velocity mode the output is proportional to motor

velocity. For details on the servo loop's actions in the different non-virtual configuration modes, refer to the *UNIDEX 600 Hardware Manual, P/N EDU154* under Servo Loop.

Torque mode is the normal configuration when not running virtual axis. In this mode the servo loop continuously monitors position and velocity error (error is commanded minus actual) and adjusts the output to the amplifier based on these errors. This is called closing the loop for position and velocity. In torque mode, the output to the amplifier is expected to be proportional to motor torque.

In closed loop velocity mode, the servo loop will close the position loop, but will not close the velocity loop. In other words the servo loop will not regulate velocity. In this mode, the velocity feedback is generated by an external analog tachometer that is input to the drive, and the velocity loop is closed in the drive itself. The voltage output to the amplifier will be proportional to velocity command. Refer to Chapter 5: AerTune under the Tuning with Tachometer Feedback section for more details on this mode.

In open loop velocity mode, the servo loop does not close the position nor the velocity loop. There is no velocity feedback loop, the velocity is commanded, but not regulated. This mode is normally used only for spindles or other axes not requiring precise velocity control.

2.3.2. Configuring Closed-Loop (Torque or Velocity)

Configuring an axis for torque mode or closed loop velocity mode involves three steps. First, the type of motor should be determined (brush or brushless). Second, the user must specify a feedback device type with a defined channel number for feedback to be received. Third, designate a Digital-to-Analog Converter (DAC) output channel number to provide a command to the amplifier driving the motor.

2.3.2.1. Motor Types

The supported types of motors include brush (DC) and brushless (AC) motors as well as any motor type whose driver accepts a torque or velocity command. The command will be a ± 10-volt command (maximum) representing torque or velocity. For brushless motors requiring commutation, two sinusoidal torque commands will be provided (120 degrees phase displaced), optionally, a third phase may be provided by the BB500 breakout module to produce a third commutation signal displaced from the first by 240 degrees.

2.3.2.2. Feedback Devices

There are several supported feedback types. One feedback type is an incremental optical encoder feedback. Up to four channels of encoder feedback can be connected to the base UNIDEX 600 card. Additional encoder channels are available with the purchase of the encoder expansion card(s) (P/N 4EN-PC). Another feedback type is a resolver/inductosyn feedback via an optional four-channel resolver board with two standard channels (P/N. RDP-PC-2). A third feedback type, is a laser feedback via the LZR laser feedback system. All encoder feedback signals received by the controller are electronically multiplied four times, producing four times the pulses per revolution specified by the encoder manufacturer. It is this number, four times the physical number of pulses per revolution of the encoder that should be entered into the axes parameters for the pulses per revolution of the encoder.

2.3.2.3. Digital-to-Analog Conversion (Output to the Amplifier)

Eight 16-bit DAC's are available per each UNIDEX 600 or encoder expansion card. Two of the channels are used per axis to provide commutation for brushless motors. The DAC channels have 16 bits of resolution on the UNIDEX 600\650 Series. The DACs provide a maximum output of ±10 volts.

2.3.3. Configuring a Spindle (Open-Loop Velocity Mode)

A special condition exists when configuring an open-loop axis that requires a velocity command (as opposed to a torque command), such as a spindle axis. Configuring the axis for a D/A channel, but utilizing NULL feedback for the feedback device does this. In addition, the lines per revolution of the feedback device should be set to 100,000 lines per revolution. The *Type* machine parameter (see section 2.4.1.) should be set to 1 or 2 defining the axis as a rotary axis. If the spindle requires a unipolar velocity command (0 to 10 not -10 to +10) the *ICMDPOLARITY* axis parameter should be set to -1. The *CntsPerDeg* machine parameter (see 2.4.2.) should be set 277.777778 (100,000 lines per rev. / 360 degrees). The servo loop gains (axis parameters) should be set as follows; KI = 0, PGAIN = 0, KP = 1. If required, the *DACOFFSET* axis parameter may be used to null any offset from the D/A. The *IMAX* axis parameter defaults to 32,767 producing a 10 volt peak velocity command output. This value may be changed as required for the drive (for example a value of 16,384 will produce a 5 volt peak). For velocity command systems, *IAVGLIMIT* should be set to zero.

Now that all the parameters have been configured for the axis, it must be properly scaled to the desired maximum velocity. First, enable the spindle axis and command it to rotate at its maximum velocity. After it accelerates to its maximum velocity, increase the value of the *KP* axis parameter until the velocity command output from the D/A is at 10 volts. The D/A command may be monitored with a voltmeter, oscilloscope or by using the AerDebug utility program to monitor the Torque (the *ICMD* axis parameter) command for the spindle axis. AerDebug will display 32,767 when the velocity command is at 10 volts. If the *KP* axis parameter cannot be adjusted so that the velocity command is at 10 volts for the maximum velocity, the user may increase/decrease the lines per revolution of the NULL feedback device to achieve the desired scaling. After completing this, save all the parameters in an INI file, since the spindle is properly configured.

If the user desires to control the spindle in a CNC program with the standard M-codes, the user needs to setup two additional task parameters. The *S1_Index* (Spindle #1) task parameter should be set to the desired task-axis index and the *S1_RPM* can be set to specify a default feedrate for the S-Word. The UNIDEX 600 supports up to four spindles per task, to use more than one spindle, configure the axes as above and specify the proper task-axis indexes. For more information, see the description for the M-codes in the *UNIDEX 600 Series CNC Programming WIN95/NT Manual, P/N EDU158*.

2.4. Motor Units (Resolution and Direction)

Configuration defines the number of counts or machine steps per motor revolution. However, after configuration, the user must set two machine parameters that instruct the software how to convert from motor counts into user units (inches, degrees, or millimeters) and vice versa. In the U600MMI these can be set through the Machine Parameters page. In AerDebug they can be set via the "PARMSET M <name>" command, where <name> is the name of the parameter as described below.

2.4.1. Linear vs. Rotary type

An axis may be a type that produces linear or rotary motion. A linear axis is defined by setting the *Type* task parameter to zero. Linear axes use inches or millimeters as units, and are appropriate for stages.

There are two types of rotary axes, both use degrees as their distance units. The first, Type 1, is a rotary axis with modulo position. It is the appropriate type for stand-alone motors. An axis with modulo position displays 0 through 359 degrees, and rolls over its position display back to 0 when it reaches 360 degrees. This will repeat for each revolution in either direction. Also, when programming in absolute units (specifying a particular target in degrees), a modulo position rotary axis always chooses the direction to move (clockwise or counter-clockwise) in order to move the shortest distance to its angular destination. When there is no short distance (e.g., a move of 180 degrees), it always causes clockwise (CW) rotation to the destination. This behavior does not apply when programming in relative mode (specifying a distance relative to the current position), in which case the arithmetic sign of the target determines the direction, similar to a linear type motors. A modulo rotary axis is defined by setting the *Type* parameter to 1.

A non-modulo rotary axis uses degrees as its distance units, but does not turnover like the modulo rotary axis, nor will it pick the shortest distance to its angular destination in absolute mode. It is appropriate for motors operating in helical coordinate systems. A non-modulo rotary axis is defined by setting the *Type* parameter to 2.

2.4.2. Motor Resolution

An axis must also have its scaling defined, so the motor counts (or machine steps) can be converted to user program units (millimeters, inches or degrees). This is done by the *CntsPerDeg* and *CntsPerInch* machine parameters. Use the respective parameter based on the type of axis defined by the *Type* machine parameter (see 2.4.1.). These parameters indicate the number of machine counts per inch or degree.

For linear type motors, users must enter the counts per inch, even if doing all the programming in millimeters (using **G71**).

For axes using brushless linear motors, the value entered into the *CntsPerInch* parameter is the number of counts equal to one electrical cycle of the motor.

2.4.3. Motor Direction

Regardless of the motor type, the user must specify which rotational direction of the motor corresponds to a positive units value. This is done with the arithmetic sign of the *CntsPerDeg* or *CntsPerInch* task parameters. A positive sign indicates that a positive units value (degrees/inches/millimeters) as measured from zero units, is reached by a clockwise rotation of the motor. A negative sign indicates that a negative units value (degrees/inches/millimeters) as measured from zero units, is reached by a counter clockwise rotation of the motor.

2.5. Drive Signals

The binary interface signals that come from/to the drive can all be set active high or active low. For example, the user may signify that a zero voltage, or ground, on the drive enable line instructs the drive to enable, or he may specify that a zero voltage instructs the drive to disable. The setting the user chooses must match the drive hardware connected to the controller.

To specify active low or active high use the *IOLEVEL* axis parameter, which is a bit mask covering the following conditions:

1.	Drive Enable	(output to drive)	0x1
2.	AUX (Mode) Output	(output to drive)	0x2
3.	CW limit Switch	(input from drive)	0x4
4.	CCW limit Switch	(input from drive)	0x8
5.	Home limit Switch	(input from drive)	0x10
6.	Drive Fault	(input from drive)	0x20

Setting a bit to one (1) means it is active high, setting it to zero (0) means it is active low.

The easiest way to set the *IOLEVEL* axis parameter is to observe the state of these signals as reported by the Aerotech controller (this is easily done using the AerStat utility program and then viewing the Axis Status tab) and insure that these reported signals agree with the hardware. If a signal does not agree, then invert its bit in the *IOLEVEL* axis parameter, this will invert the active state of that signal.

For example, if the drive unit indicates via the LED that it is not enabled, but AerStat indicates that the drive is enabled, then flip the first bit in the *IOLEVEL* axis parameter (i.e., add 1 if *IOLEVEL* is a multiple of 2, else subtract one).

2.6. Axis Faults

Certain conditions (such as excessive position error, or EOT limits being hit) are continuously checked periodically on all axes, in a tight or fast loop (ten-millisecond cycle time). If any of these conditions occur, then an axis fault exists on that axis. Multiple axis faults can exist concurrently on the same axis.

Using the six fault masks, the user defines what is to be done, if anything, when axis faults occur. Usually, these actions are conditions such as disabling or halting the drives or applying a brake.

It is crucial that the fault masks are setup properly before moving axes in order to avoid damaging or destroying hardware. Fault masks are essential to the safe and accurate operation of the UNIDEX 600. Pay special attention to the position error and RMS current faults in order to prevent runaway conditions.

2.6.1. FAULT

The *FAULT* axis parameter, when non-zero, indicates an axis fault condition. The value is a bit mask, indicating one or more faults that occurred on that axis (see Table 2-5) These conditions are all on a per axis basis, meaning they can occur independently for each axis. Most fault conditions have threshold values that are set via other axis parameters, refer to Table 2-5

The U600MMI will report an axis fault by blinking the appropriate message in the position display. The user can also view the value of the *FAULT* parameter, along its bit breakdown in AerStat (which bits indicate which faults). AerDebug also reports the fault value with a "PARMGET A FAULT" command, but it will not provide a bit breakdown.

The value of the *FAULT* parameter remains set until the fault is acknowledged by the user or the application program. Writing its bit value back to the axis fault parameter clears a fault. For example, if the *FAULT* parameter indicated the CW, CCW and Position Error Limit faults occurred, the *FAULT* parameter would indicate a value of 13. Setting the *FAULT* axis parameter to 13 would clear all those faults. Setting it to 12 would clear the CW and CCW faults, but not the position error limit fault. Alternately, all faults may be simultaneously cleared by setting *FAULT* to a value of –1 (-1 = 0xffff). Pushing the Fault Acknowledge button on the U600MMI is equivalent to setting the *FAULT* axis parameter to –1. The user cannot acknowledge a fault with the AerStat utility.

2.6.1.1. Axis Faults and Programming

The Programming and Task axis faults listed in Table 2-5 (bits 8 and 16) exist to trap faults generated by faulty programming from the two interfaces: library and CNC, respectively. A programming Axis fault occurs when a faulty library call involving the given axis executed. A *Taskfault* Axis fault occurs when a faulty CNC command executes and the *Taskfault* bit is on in the *FAULTMASK* for that axis. Refer to Chapter 3: Programming for more details.

Often these programming interfaces provide additional protection. For example, velocity and velocity command faults are generated when exceeding the axis parameters *VELTRAP* and *VELCMDTRAP*. However, when programming from the CNC interface, additional protection is available in the *RapidFeedRateIPM* and *RapidFeedRateRPM* task parameters. If these are violated, the system generates a Task fault (see section 2.7).

The C/C++ programmer also has the User Axis Fault available (see chart), where they can force an axis fault.

Table 2-5. Axis Faults

Bit	Hex Value	Fault Name	Description
0	0x1	Position Error Limit	Difference between instantaneous commanded position and actual position exceeds the amount specified in the *POSERRLIMIT* parameter.
1	0x2	RMS Current Limit	Average current exceeds the amount specified in the *IAVGLIMIT* parameter averaged over *IAVGTIME* parameter.
2	0x4	CW Hard Limit	The system encountered the CW (clockwise) limit switch. (see *IOLEVEL* parameter)
3	0x8	CCW Hard Limit	The system encountered a CCW (counter clockwise) limit switch. (see *IOLEVEL* parameter)
4	0x10	CW Soft Limit	The user commanded an axis to move beyond the position specified in the *CWEOT* (clockwise end-of-travel) axis parameter.
5	0x20	CCW Soft Limit	The user commanded the axis to move beyond the position specified in the *CCWEOT* (counter-clockwise end-of-travel) axis parameter.
6	0x40	Drive Fault	Drive fault input. (see *IOLEVEL* parameter) However, after clearing the drive fault input, this bit continues to reflect the fact that the fault occurred.
7	0x80	Feedback Fault	Feedback failure input from the feedback associated with the axis. This typically occurs when the feedback device is not functioning properly, or the feedback cable is disconnected.
8	0x100	Programming Fault	Axis processor received an invalid command from the PC host. These only occur when processing programming commands from programs running on the PC (U600MMI, AerDebug). Refer to the *UNIDEX 600 Series Library Reference, P /N EDU156* under "Programming errors".
9	0x200	Master Feedback Fault	Feedback failure input from the feedback channel associated with the axis configured as a master. This usually occurs when the feedback device on the master axis is defective, or the cabling is bad.
10	0x400	Home Fault	System encountered a homing fault. This typically occurs for either of two reasons: while executing a home cycle the home limit switch input was not detected; or when the system encounters an end-of-travel limit switch before the first resolver null or marker pulse.
11	0x800	User Fault	Application has requested a fault be generated with the *AerProgSetUserFault( )* function. It provides a way for a programmer to generate an axis fault from within a C/C++ or VB application program.
12	0x1000	Velocity Trap	Actual velocity exceeded the value specified in the *VELTRAP* axis parameter.
13	0x2000	Velocity Command Trap	Instantaneous commanded velocity exceeded the value specified in the *VELCMDTRAP* axis parameter.
14	0x4000	Home Tolerance Fault	Distance traveled from when the system detected the marker pulse (or the Resolver null), until the system encountered the home limit switch is less than the value specified in the *HOMESWITCHTOL* parameter. This occurs during a homing sequence.

Table 2-5. Axis Faults (cont'd)

Bit	Hex Value	Fault Name	Description
15	0x8000	Probe Fault	Occurs each time the probe trigger causes the position to latch. This is useful for notifying the application program that position information is available.
16	0x10000	Taskfault	Taskfault occurred while executing a CNC command running a task. (please see the *Taskfault* Task parameter)
17	0x20000	External Feedback Fault	Difference between the integration of the velocity command and velocity feedback is greater than the *FBWINDOW* axis parameter.
18	0x40000	Safe Zone	*SAFEZONE* axis parameters are active and the axis has violated the defined safe zone.
19	0x80000	Constant Velocity Phase Interrupt	Axis interrupt was generated when move reached constant non-zero velocity (see *INTMASK* Axis parameter).
20	0x100000	Decel Phase Interrupt	Axis interrupt was generated when move reached the decel phase.
21	0x200000	Move Done Interrupt	Axis interrupt was generated when move was done.
22	0x400000	*POSTOGO* interrupt	Axis interrupt was generated when *POSTOGO* passed under the *POSTOGOIRQ* value (see the *POSTOGOIRQ* Axis parameter).
23	0x800000	ESTOP	Emergency stop has occurred (see Section 2.8.)
24	0x1000000	WatchDog	Fail Safe timer
25	0x2000000	Position Tolerance	Axis did not move the distance specified by POSTOLERANCE, within the POSTOLTIME period at the start of the move
26-31			Unused

2.6.2. Fault Masks

The UNIDEX 600 Series controller has several bit mask axis parameters that define the controller's reaction to fault conditions. These parameters are *FAULTMASK*, *AUXMASK*, *ABORTMASK*, *BRAKEMASK*, *DISABLEMASK*, *HALTMASK*, and *INTMASK*. The actions associated with these parameters are detailed in the following sections.

Keep in mind that the word "fault mask" refers to all of the parameters above, while *FAULTMASK* applies only to the first in the list above.

Each of the fault masks relates to a particular action to take. The value of a fault mask is a bit mask, representing a set of conditions for the given action to take. The given action takes place if one or more of the conditions is true. For example, the action for *DISABLEMASK* is disabling the axis.

Refer to Section 2.6.2.8., for an example of how to use fault masks.

Table 2-6. Fault Mask Actions

Axis Parameter	Action
FAULTMASK	Determines which faults will be detected or ignored.
DISABLEMASK	Disable drive
HALTMASK	Halt motion
AUXMASK	Set/clear *AUX* axis parameter
ABORTMASK	Abort motion
INTMASK	Send interrupt to PC application
BRAKEMASK	Activate Motor brakes

Setting a mask bit on a fault mask parameter causes the action associated with that mask parameter to automatically transpire when the condition associated with that bit occurs. All of these parameters have the same bit mask definitions, since they all apply to the same set of fault conditions. Table 2-5 lists their definitions.

A bit in the *FAULTMASK* must be set to true in order for the corresponding bit in the other masks to be acted upon. There is one exception, the *BRAKEMASK* will be activated when the drive is disabled, regardless of the setting of the *FAULTMASK*.

2.6.2.1. FAULTMASK

Setting a bit in the *FAULTMASK* causes an axis fault to transpire when the condition associated with that bit occurs, see Table 2-5. When an axis fault occurs by default, all CNC programs stop (see the *STOPAXISMASK* Task parameter to change this). If synchronous motion was executing in the task when the task stops, the motion is ramped down using the ramp parameters used during a normal deceleration. However, if the program running in the task executed asynchronous motion, then that motion is not stopped when the program is stopped (to force asynchronous motion). If the programmer requires different actions (other than a ramp down) in the currently running motion, they must set one of the other masks (*HALTMASK*, *DISABLEMASK*) in addition to the fault mask. The other masks (with the exception of *BRAKEMASK* in some circumstances) will not operate for a particular fault condition unless the *FAULTMASK* is set for that fault condition. Table 2-5 explains each fault mask in detail.

2.6.2.2. DISABLEMASK

Setting the *DISABLEMASK* bits causes the drive to be disabled when any of the conditions occur whose bit is set true in the *DISABLEMASK*. The disable mask bits are ANDed with the fault mask bits. If any of the resultant bits test true (match the *FAULT* axis parameter value), the UNIDEX 600 Series controller disables the drive. The bit descriptions are the same as those listed in Table 2-5.

2.6.2.3. HALTMASK

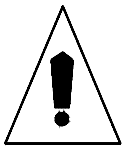
The *HALTMASK* causes the axis to decelerate to a stop in the rate/time indicated by the *DECEL*, *DECELRATE*, and *DECELMODE* axis parameters. The halt mask bits are ANDed with the fault mask bits. If any of the resultant bits test true (match the *FAULT* axis parameter value), the UNIDEX 600 Series controller decelerates the drive to a stop. The bit descriptions are listed in Table 2-5.

2.6.2.4. AUXMASK

Setting the bits in *AUXMASK* causes the auxiliary (mode) output to be set when the specified fault occurs. This output may be used to engage a brake on a vertical. The auxiliary mask bits are ANDed with the fault status bits. If any of the resultant bits test true (match the *FAULT* axis parameter value), the UNIDEX 600 Series controller sets the auxiliary output. The auxiliary mask bit descriptions are the same as those defined in Table 2-5. The auxiliary mode output clears when the fault is cleared. A delay timer is available, in the Axis parameter *AUXDELAY*.

2.6.2.5. ABORTMASK

Setting the bits in the *ABORTMASK* causes the axis to come to an abrupt (uncontrolled) stop without any programmed decel. The abort mask bits are ANDed with the fault status bits. If any of the resultant bits test true (match the *FAULT* axis parameter value) the UNIDEX 600 Series controller instantaneously zeros the command output to the drive.



When setting these bits, be especially careful, because an abrupt stop may cause physical harm to the user and/or damage the equipment.

2.6.2.6. INTMASK

Setting a bit in the *INTMASK* defines which fault conditions cause a hardware PC interrupt to be generated. Typically, this interrupt is handled by an application that runs on the PC under Windows NT. If there is no application programming being done on the PC, or are otherwise not interested in intercepting these interrupts, they can be ignored, as they have no effect on the motion generated or any other operation of the system. The user must use the AerEventxxx() library functions (refer to the *UNIDEX 600 Series Library Reference, Win NT/95 Manual, P/N EDU156*) to intercept these interrupts.

The interrupt mask bits are "ANDed" with the fault status bits. If any of the resultant bits test true (match), the UNIDEX 600 Series controller will generate an axis interrupt. The "phase" bits (bits 19 through 22) are an exception, the *FAULTMASK* does not need to be set for these to be active; that is they need only be set in the *INTMASK*. The interrupt will not occur properly if the jumper setting on the card and the Win95 registry value do not agree (see section 2.2.3 to set up and test the PC interrupts).

2.6.2.7. BRAKEMASK

The *BRAKEMASK* allows the user to define faults that cause the brake output to be activated, refer to the *UNIDEX 600 Hardware Manual, P/N EDU154*. The bits that are set true must also be set true in the *FAULTMASK* parameter. The UNIDEX 600/650 have a single brake output for all axes with a brake. Also, if any bit in the *BRAKEMASK* is set and the axis is disabled, the brake is activated. This occurs regardless of the setting of *FAULTMASK* for that axis.

Only faults corresponding to the bits set in the *FAULTMASK* will be recognized. Setting this same bit in the other axis parameter bit masks causes the fault to be acted upon by the defined function of the bit mask parameter.

2.6.2.8. Example

If it is desired to have an axis abort a move on a position error limit (bit 0) or a Clockwise (CW) or Counterclockwise (CCW) hardware end-of-travel limit (bits 2 and 3) and generate an interrupt to the application program for an RMS current limit (bit 1) or a drive fault (bit 6), the *FAULTMASK* axis parameter would have all these bits set as shown in Table 2-7.

Table 2-7. Bits Set for the *FAULTMASK* Parameter

Bit #	Decimal	Hex	Description
Bit 0	1	0x1	Position error exceeded *POSERRLIMIT* parameter.
Bit 1	2	0x2	RMS current limit exceeded *IAVGLIMIT* parameter.
Bit 2	4	0x4	CW hardware end of travel limit encountered.
Bit 3	8	0x8	CCW hardware end of travel limit encountered.
Bit 6	64	0x40	Drive fault has occurred.
	+	+	
FAULTMASK =	79	0x4F	

Likewise, the indicated bits would be set in the *ABORTMASK* and *INTMASK* parameters. Summing the value of the desired bits together and setting the parameter to that value as shown in Table 2-8 and Table 2-9 does this.

Table 2-8. Summing Bits for *INTMASK*

Bit #	Decimal	Hex	Description
Bit 1	2	0x2	RMS current limit exceeded *IAVGLIMIT* parameter.
Bit 6	64	0x40	Drive fault has occurred.
	+	+	
INTMASK =	66	0x42	

Table 2-9. Summing Bits for *ABORT* MASK

Bit #	Decimal	Hex	Description
Bit 0	1	0x1	Position error exceeded *POSERRLIMIT* parameter.
Bit 2	4	0x4	CW hardware end of travel limit encountered.
Bit 3	8	0x8	CCW hardware end-of-travel limit encountered.
	+	+	
ABORTMASK =	13	0x0D	

2.7. Task Faults

In addition to Axis faults there are Task Faults. A task runs a CNC program. Refer to Chapter 3: Programming, under CNC Tasks and Programs for details.

Task faults indicate an error in the execution of a CNC program. For example, dividing by zero or trying to set a non-existent parameter from a CNC program causes a task fault. There are many conditions that can cause task faults (for application programmers, any error prefixed by "AER960RET_" in AERCODE.H can potentially be returned as a task fault). The user can also trigger a task fault manually, by setting the *TaskFault* task parameter to non-zero, although only certain numbers (the AER960RET_" constants) will yield a recognizable description. Refer to the *TASKFAULT* Task parameter in Appendix C for more details.

2.8. Emergency Stop (ESTOP)

The UNIDEX 600 has a dedicated optically isolated emergency stop (ESTOP) sense input. Refer to the *UNIDEX 600 Hardware Manual, P/N EDU154*, under Technical Details for hardware details. If this input goes active, the axis processor generates an ESTOP Task fault on all four CNC tasks. However, the *EStopEnabled* global parameter must be set to 1 to enable the sensing of the opto-isolated emergency stop input.

Also, an emergency stop binary input may be defined for each task. Define the task ESTOP by setting the *EStopInput* task parameter to the value of the binary input driven by the external ESTOP circuit (set it to minus 1 if there is to be no task ESTOP). If this input goes active, the axis processor generates an ESTOP Task fault on the given task only.

Another parameter effecting ESTOP is the *GlobalEstopEnabled* Task parameter. If this parameter is non-zero, then the global ESTOP is ignored for that task only. This allows the user to use the global ESTOP on some tasks, but the task ESTOP on others. By default this parameter is zero.

By default the *EstopEnabled* Global parameter is zero, so it must be set in order to enable a global ESTOP. Note also that the *EstopInput* is minus one by default, so it must be set in order to have a task ESTOP.

When an ESTOP occurs (either through the optically isolated input or a binary input), the system generates an ESTOP Task fault.

What happens due to a Task fault is discussed in detail under the documentation for the *TaskFault* Task parameter. In summary: an ESTOP task fault stops the CNC program, and by default, generates axis faults on all the axes it is bound to, disabling those axes. However, the behavior of the system after an ESTOP can be customized, please see the *TaskFault* task parameter documentation for details.

The user cannot run programs on any task while the ESTOP task fault is on. However, the user can run most immediate mode commands, excepting those that initiate motion, or enable drives.

2.9. Axis Testing

The axes should be tested before their initial use, which includes verifying proper phasing of the feedback (encoder and/or resolver) as well as any required end-of-travel limits.

2.9.1. Axis Limits

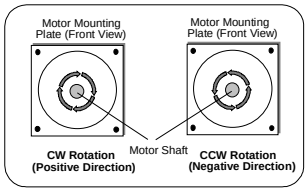
After configuring the axes, the limits are easily verified by running the AerStat utility and selecting the axis status tab to view the limits, which are displayed as "CW input", "CCW input" and "Home input". Sequentially activate the limits on each axis noting the appropriate limit indicates it is "ON" when the limit input is active. Be sure that the limits are connected, so when the motor rotates CW it encounters the CW limit not the CCW limit. If the limit reads opposite polarity (AerStat indicates off, when its on, and vice versa) the *IOLEVEL* axis parameter value is wrong, see Section 2.5.

The U600 software will only see a limit if the axis moves into a limit. This allows the motor to move off the limit (after the limit fault is cleared) without triggering another limit fault. The U600 will not report the limit if the switch is triggered when the motor is not moving.

2.9.2. Axis Feedback

Position feedback is easily verified by monitoring the axes' positions with the AerDebug utility. Sequentially, select each axis with the **AX** command, then use the **PARMMON** command to monitor the *POS* parameter for encoders or the *RESOLVER* parameter for resolvers.

```
AX 2                  ; Select the axis
PARMMON A POS         ; Monitor the encoder channel this axis has been
                      ; configured for
                      ; OR
PARMMON A Resolver    ; Monitor the resolver channel this axis has been
                      ; configured for.
```



Rotate the motor shaft CW (as viewed looking into the motor shaft). The position displayed should show a positive increase and stop counting when motor rotation stops. The position may dither back and forth slightly, particularly on high-resolution systems, even when the motor is at rest. If an axis is configured for dual feedback, be sure to verify each feedback device.

2.9.3. Axis Loop Closure

Once establishing the proper feedback, the user should enable the drive to see if it can hold position. The user can enable the drive by clicking on the axis name in the Manual screen of the U600MMI. In AerDebug, deliver the command "SET DRIVE=1". If possible, turn the motor manually and let go, and the servo loop should return it to its original position. The user should feel some shaft stiffness.

It is possible that moving the motor, or even enabling the drive causes instability. If so, the axis needs tuning. See section 2.11.

2.10. Accelerations

All moves except synchronous CNC moves (**G1, G2, G3**) and Cam Table motion use a set of axis parameters to determine acceleration and deceleration behavior. This includes homing and jogging motion.

The user must make sure these axis parameters are set properly before moving an axis: *ACCELMODE*, *ACCEL*, and *ACCELRATE*. Acceleration takes place at the given rate, (*ACCELRATE*) or within the given time (*ACCEL*) based on the value of *ACCELMODE*. *ACCELMODE* also determines the profile (as seen in velocity/time space) of the acceleration.

A similar set of axis parameters: *DECELMODE*, *DECEL*, and *DECELRATE* exist for setting the deceleration behavior of an axis. These also must be set properly.

2.11. Axis Tuning

The servo loop gains must be adjusted based upon the load and required performance of the axis. For detailed information on tuning a servo loop, refer to Chapter 5: AerTune.

The following axis parameters are used for adjusting the response of a torque (current) mode servo loop: *KI*, *KP*, *PGAIN*, *VFF*, *AFFGAIN*, *ALPHA*, and, the *VGAIN* parameter for large frictional loads.

The following axis parameters are used for adjusting the response of a velocity mode servo loop (see section 2.3.3.): *PGAIN*, *VGAIN*, *AFFGAIN*, and *ALPHA*. Both the *KI* and *KP* should be set to zero for velocity mode servo loops. In addition, the *DACOFFSET* parameter may be used for nulling offsets from the DAC that may be present in velocity loop systems.

Axis tuning is thoroughly described in Chapter 5: AerTune, which is designed for tuning and monitoring an axis' performance.

2.12. Jogging

The user can use the U600MMI in the manual page to jog the axes. Set the axes to move, the distances and velocities, enable the drive, and push an arrow button.

In AerDebug, the user can easily test axis motion with the **INDEX** or **MOVETO** CNC commands. See the example below, where we "jog" axis 2, 4000 mm from the current position, at a speed of 500 mm/second. Do not enter the text after the ";" below, these are just comments.

```
AX 2
EXEL "G71"                ; Sets unit used in "INDEX" to mm. Do a
                          ;  G70 instead to set them to inches
EXEL "MAP Y2"             ; "Y" now means physical axis (channel) 2
EXEL "BIND Y"
PARMSET A DRIVE 1
EXEL "INDEX Y 4000 500"   ; Starts the motion

These commands can be used to monitor the progress of the move.
TSKI                      ; This should show "No Errors" in the fault line
PARMMON A POS             ; Shows the position moving (counts)
PARMMON M POSUNITS        ; Shows the position moving (mm/inch)
```

See Chapter 4: AerDebug for details on the AX, EXEL and PARMSET parameters. See the *UNIDEX 600 Series CNC Programming manual, EDU158* for details on the **MAP**, **BIND** and **INDEX** commands.

2.13. Homing

There are a number of types of homing and a number of adjustable parameters affecting the performance of homing.

There are two axis faults that can occur during homing: a home fault or a home switch tolerance fault.

A homing fault typically occurs for either of two reasons: while executing a home cycle the home limit switch input was detected or when the system encounters an end-of-travel limit switch before the first resolver null or marker pulse.

HomeType Machine Parameter	Type of home cycle
HomeDirection Machine Parameter	Direction to begin homing
HomeOffsetDeg Machine Parameter	Home offset (for rotary axes)
HomeOffsetInch Machine Parameter	Home offset (for linear axes)
HomeFeedRateRPM Machine Parameter	Speed to home at (rotary axes)
HomeFeedRateIPM Machine Parameter	Speed to home at (linear axes)
HOMEVELMULT Axis Parameter	Ratio of home switch to reference pulse search speeds
FAULTMASK Axis parameter	Used to define behavior when EOT hit, or home fault occurs.
HOMESWITCHTOL Axis Parameter	Minimum separation of home position from home switch (resolvers only)

The *HomeType* machine parameter defines the behavior of the home cycle. See the *HomeType* machine parameter description for a definition of the home types.

Once the axis has completed the home cycle, if a home offset has been defined by the *HomeOffsetDeg* or *HomeOffsetInch* parameters, the value of the home offset will be loaded into the axis position registers and the home position will now be equal to the home offset value. The offset is loaded into the position, position command, raw position, and preset register values.

Once an axis is successfully homed, this fact is reflected in the home bit of the *STATUS* axis parameter and also in the at home bit of the *SERVOSTATUS* axis parameter (the user can use the AerStat utility to view these). Once homed, an axis stays homed, until it is reconfigured. Disabling an axis does not remove the homed status.

The *HomeFeedRateIPM* and *HomeFeedRateRPM* allow the feedrate to be specified during the home cycle. Typically, the home feedrate is a low velocity that produces an accurate home reference point. A low speed is not detrimental to machine throughput, since it is done occasionally or when the machine is first powered up. During homing, the axis obeys the accel/decel parameters as specified by the appropriate axis parameters (see *ACCELMODE* axis parameter)

The home cycle feedrate may be increased if the *HOMEVELMULT* axis parameter is used. This parameter allows the home feedrate to be scaled down by the percent specified in the *HOMEVELMULT* axis parameter during the reference pulse search. This allows a faster move into the home limit, where the axis may be a greater distance from the home

limit and accuracy is not an issue, until searching for the reference pulse. The value entered is an integer representing the desired percent of the home feedrate during the marker search, (i.e.; a value of 25 would scale the home velocity to 25% during the reference pulse search).

When an axis is not homed, software limits are inactive, that is they are ignored even if the fault masks are set to detect them. If the axis is configured with Axis calibration, Axis calibration is active regardless of whether an axis is homed or not.

2.14. Programmed Moves

The UNIDEX 600 Series controller provides two methods of programming: CNC and library interface, refer to Chapter 3 on Programming for details. The behavior of programmed moves is described under the documentation for that interface. For CNC moves refer to Appendix A in the *UNIDEX 600 Series CNC Programming Manual, P/N EDU158* for motion details. For the library interface refer to *the UNIDEX 600 Series Library Reference, P/N EDU156* under the AerMove functions chapter.

Programmed movement can be easily tested/debugged by setting the axis parameter *SIMULATION* to 1 for the axes to be moved. This allows programs to run without generating movement of the axis. Positions and velocities are reported just as if the axes were moving. Enabling the axis still produces motor torque, but there will be no motion generated. In this mode, drives and motors do not need connected.

2.15. Digital I/O

The UNIDEX 600 Series of controllers utilizes a Virtual I/O mapping technique that provides up to 512 digital inputs, 512 digital outputs, 128 16-bit register inputs and 128 16-bit register outputs. These I/O resources are referred to as Virtual, since they exist as blocks of memory in the UNIDEX controller. From the C/VB library programming level, these memory mapped I/O points can be accessed via the AerVirt series of commands (see the *U600 Series Library Reference, Win NT/95 Manual, P/N EDU156*). CNC programs have access to the Virtual I/O through **$BI**, **$BO** CNC commands or through user defined M and G-codes (see the *UNIDEX 600 Series CNC Programming, Win NT/95 Manual, P/N EDU158*).

It is possible to *write* Virtual inputs and *read* Virtual outputs. This capability provides for communication and synchronization of multiple tasks executing on the controller.

2.15.1. Associating Virtual I/O with Physical I/O

By default on the UNIDEX 600 controller the 16 digital inputs and 16 digital outputs located on the card are mapped into Virtual inputs 0 through 15 and Virtual outputs 0 through 15. Therefore, reading Virtual inputs 0 to 15 will return the state of the digital inputs on the card. Likewise, setting Virtual outputs 0 to 15 will modify the digital outputs on the UNIDEX 600 card.

Each encoder expansion card present in the system adds an additional 40 digital inputs and 40 digital outputs. These I/O points map into the next 40 Virtual I/O points (Virtual

I/O 16-55). Table 2-9 illustrates the relationship between UNIDEX 600/Encoder card I/O and Virtual I/O mapping.

The mapping of the encoder card I/O to Virtual I/O only occurs if an axis encoder feedback channel has been configured for that card. By configuring an encoder feedback channel on the encoder expansion card, the UNIDEX 600 controller is made aware of the presence of that card.

Any Virtual I/O not occupied by the UNIDEX 600 card or Encoder expansion cards can be mapped to third party I/O sources. This can be accomplished by running an application or thread at the host CPU level that accesses I/O through either the AT-Bus, a network/PLC interface or a user defined database and copies these I/O states into the Virtual I/O using the AerVirt functions. In this manner sophisticated communication structures can be created between the UNIDEX 600 and PLCs without using physical I/O points and all the associated wiring.

Table 2-10. Relationship between U600/Encoder I/O and Virtual I/O Mapping

INPUTS			
Board	**Label**	**Virtual Input #**	**Connector and Pin Numbers on the Respective Board**
UNIDEX 600	IN 0-15	0 through 15	P9 pins 31 - 1 (odd pins)
Expansion Board 1	IN 0-15	16 through 31	P9 pins 31 - 1 (odd pins)
Expansion Board 1	IN16-39	32 through 55	P8 pins 47 - 1 (odd pins)
Expansion Board 2	IN 0-15	56 through 71	P9 pins 31 -1 (odd pins)
Expansion Board 2	IN16-39	72 through 95	P8 pins 47 - 1 (odd pins)
Expansion Board 3	IN 0-15	96 through 111	P9 pins 31 -1 (odd pins)
Expansion Board 3	IN16-39	112 through 135	P8 pins 47 - 1 (odd pins)
OUTPUTS			
UNIDEX 600	OUT 0-7	0 through 7	P9 pins 47 - 33 (odd pins)
UNIDEX 600	OUT 8-15	8 through 15	P10 pins 47 - 33 (odd pins)
Expansion Board 1	OUT 0-7	16 through 23	P9 pins 47 - 33 (odd pins)
Expansion Board 1	OUT 8-15	24 through 31	P10 pins 47 - 33 (odd pins)
Expansion Board 1	OUT16-39	32 through 55	P7 pins 47 -1 (odd pins)
Expansion Board 2	OUT 0-7	56 through 63	P9 pins 47 - 33 (odd pins)
Expansion Board 2	OUT 8-15	64 through 71	P10 pins 47 - 33 (odd pins)
Expansion Board 2	OUT 16-39	72 through 95	P7 pins 47 -1 (odd pins)
Expansion Board 3	OUT 0-7	96 through 103	P9 pins 47 - 33 (odd pins)
Expansion Board 3	OUT 8-15	104 through 111	P10 pins 47 - 33 (odd pins)
Expansion Board 3	OUT 16-39	112 through 135	P7 pins 47 -1 (odd pins)

2.16. Other Manuals

The UNIDEX 600 Series of controllers has several other manuals documenting various aspects of the controllers use, hardware or programming, some of which are included as part of optional hardware or software.

2.16.1. Hardware Manuals (UNIDEX 600)

For a description of the controller's hardware, reference the hardware manual for the specific controller, either the UNIDEX 600 Hardware Manual (P/N EDU154). Other related Aerotech hardware manuals are the DR500 Hardware Manual (P/N EDA120), the BA Series Amplifier Manual (P/N EDA121) and the PSO-PC (Laser Firing) Manual (P/N EDO105).

2.16.2. Programming Manuals

For information on writing motion programs in the UNIDEX 600 Series CNC G-code programming language, reference the UNIDEX 600 Series CNC Programming Manual (P/N EDU158), or, preferably, the online help file for the most up-to-date information.

For Visual Basic and C programmers, reference the UNIDEX 600 Series Library Reference Manual (P/N EDU156) or for OLE Custom Control programming, reference the Software Development Kit online help file provided.

2.16.3. MMI Interface

Reference the online help file provided.

∇ ∇ ∇

CHAPTER 3: PROGRAMMING

In This Section:

3.1. Introduction

This chapter provides an overview of the two available programming interfaces and the trade-offs between using one or the other to program the U600 Series controller. The correct interface or combination of interfaces the programmer should use depends on the target application. Therefore, the programmer must understand the fundamentals of both in order to make the correct decision.

The UNIDEX 600 Series motion controllers can command or monitor motion through two fundamentally different methods; see summary in Table 3-1.

Table 3-1. The Two Programming Interfaces Available

Programming Interface	Language Syntax	Processor Run On	Manual
Library	C++, C, or Visual Basic	PC CPU x86	*U600 Series Library Reference Manual P/N EDU156*
CNC	RS-274 CNC	Axis Processor	*U600 CNC Programming Manual P/N EDU158*

The majority of the functionality of the U600 controller is accessible under either interface. In both interfaces the user generates motion by writing programs that contain steps or lines that execute sequentially. In both cases the lines execute as a background process to the actual motion controller (refer to Chapter 1: Introduction and Overview, under system architecture). There are two major differences between the two interfaces: the format of the programming language, and the processor that executes the program steps.

In PC host controlled motion, the programs execute on the PC processor. The programmer writes the program in C, C++, or Visual Basic, then compiles and executes it from the PC. The programmer controls the motion by making library function calls that invoke motion controller functions through a device driver running on the PC.

In G-code motion, the programs execute in the axis processor. The user writes the programs in an extended RS-274 standard syntax, compiles them on the PC, and downloads them to the axis processor via the device driver. In contrast to the library controlled programs, they execute on the axis processor independently of the PC.

3.1.1. Combination Programming

The user should understand that although the two interfaces are distinct and exclusive from each other (commands in one syntax are not understandable to the other), the majority of the functionally of the U600 axis processor is available from either interface (see Table 3-2 for a summary of the motions available from either interface). In addition, a programmer can construct an application that utilizes both interfaces simultaneously, since each interface is serviced by a separate and independent execution unit on the axis processor card (refer to Chapter 1: Introduction and Overview, under System Architecture).

For example, a common combination is to run the motion in the CNC interface, but write a library calling interface that runs a GUI interface to control and monitor program execution (this is what the U600 MMI actually does).

Table 3-2. Advantages of the Two Programming Interfaces

PC Controlled Advantages (Using Library Calls)	Axis Processor Controlled Advantages (Using G-codes)
Can use C, C++, or Visual Basic languages.	Program speed independent of PC processor speed.
Can use sophisticated WINNT multitasking capability.	Industry standard RS-274 G-code capability.
Full user interface control.	Easy modification of program source by end users.
Allows access to CNC G-code compiler calls.	Compatibility with output of CAD packages.

3.1.2. Multi-Tasking

Both interfaces allow multi-tasking; execution of multiple programs on a asynchronous basis. In the Library interface, the user has all the rich multitasking inherent in the PC at their disposal. The CNC interface has four CNC tasks that can execute up to four programs simultaneously. However, in both interfaces the user must understand that a single processor is utilized underneath the multi-tasking and that multi-tasking is only achieved at the cost of slowing down the execution of the individual tasks.

3.2. The Library Programming Interface

Library, or Host Controlled Motion programs, are Windows 95/NT application programs written in C/C++ or Visual Basic executing on the x86 PC. These application programs induce axis motion by executing U600 library functions that in turn run functions executing on the Library Servicer execution unit (refer to Chapter 1: Introduction and Overview, under System Architecture) on the axis processor.

The most important advantage of the library interface is the power inherent in the C/C++ programming (or Visual Basic) languages. These languages are more sophisticated than the RS-274 CNC language, so the user can more easily write complex or large programs. Furthermore, Visual Basic, Visual C++ and other compilers allow the user to call GUI functionality, providing the programmer the ability to define an interface in the same language as they write the motion commands.

Another important advantage of library calling over the CNC interface is its ability to configure motors. The user cannot configure motors from a CNC program, CNC programmers must do this through a library calling application (either the U600 MMI, or AerDebug, or a custom constructed application).

A similar advantage is downloading and running CNC programs; this can be done through library calls, but the CNC programmer must use either the U600 MMI, or AerDebug or a custom library application to do this.

An important feature of library controlled motion is the product executing on the PC is a binary executable file, which the end user cannot modify. The only way to modify the program is by altering the C/C++ or visual basic source code and generating a new executable via the C/C++ or Visual Basic compiler/linkers. Normally, these facilities are not present on the floor machine. Therefore, the machinist cannot alter the program. This can be an advantage or disadvantage depending on the specific target application.

The major disadvantage of library controlled programs is that the host processor execution speed could affect application execution. The user must take care in multitasking PC operating systems to ensure that their high priority motion control tasks are not starved for execution time by lower priority tasks, or other applications running on the PC.

The user should note that the library call invokes the CNC compiler and can pass lines to compile (as opposed to getting them from a file). Therefore, the library calling user can easily access all CNC functionality that is not directly accessible through library functions (i.e., contoured **G2** motion) by invoking the CNC compiler library functions. Furthermore, the library programmer can even construct their own CNC compiler that assembles CNC packets and sends them to the axis processor to execute, thereby completely duplicating the CNC interface. However, the programmer should be warned, this is not a simple thing to do due to the inherent complexity of compiling.

3.2.1. Basic Elements of a Library Interface Program

All library controlled application programs written for the UNIDEX 600 Series controllers must have a minimum subset of functionality in common. These functions include:

1) Opening a channel of communication to the UNIDEX 600 Controller and downloading the firmware.
2) Configuring the axes for the types of feedback transducers and motors present.
3) Setting all applicable axis parameters.
4) Running motion commands.
5) Fault Handling.

Opening a path of communication to the card and downloading the axis firmware is achieved by executing two commands:

- AerSysOpen(…)
- AerSysDownLoad(…)

After executing these commands, the UNIDEX controller is ready to accept all other function calls. The *AerSysDownLoad* statement only has to execute once to bring the controller out of the reset state. *AerSysReset()* can be used to return the processor to the reset state.

Specific function calls have been provided for the configuration of the most commonly used feedback devices such as encoders and resolvers for commutated and non-commutated motors. Also, functions have been defined to permit dual feedback transducers (one for position feedback and one for velocity feedback) on a per axis basis. All these functions are prefixed by "*AerConfig*".

Function calls exist for setting and retrieving Axis, Task, Gobal and Machine parameters. The programmer can choose to implement their own parameter management mechanism (i.e. writing and reading parameter values from files), hard code parameters in the application program or use the Aerotech provided parameter file management mechanisms. All these functions are prefixed by "*AerParam*".

Motion commands consist mainly of synchronous/asynchronous linear motion in either absolute or incremental coordinates, G-code commands and cam table/electronic gearing. Combinations of these types of motion may be active across multiple axes at the same time. Please see Chapter 1 for details on available motion. All these functions are prefixed by "*AerMove*" or "*AerCamTable*".

Fault handling from the library interface is accomplished via setting the appropriate axis parameters, which is done through the "AerParam" functions. Please see Chapter 1, under Faults, for a generic description of U600 fault handling.

3.3. CNC G-code Programming

G-code motion consists of RS-274 compatible programs executed on the UNIDEX 600 Series controller. The user compiles, downloads, and starts these programs using library interface calls, but the actual program execution is performed independent of the PC processor. Therefore, program execution speed is totally independent of what programs are running on the PC, the speed of the ISA bus, and the speed of the PC.

Another major advantage to CNC programming is, it uses the RS-274 standard, a industry standard programming language accepted and understood in many manufacturing environments. Furthermore, many third party applications exist that can translate CAD data into RS-274 G-code programs.

A serious drawback of the CNC interface is it lacks the language sophistication of PC languages like Visual C++ and Visual Basic. Most notably, the only GUI interface capability available in the CNC language is the **DISPLAY** command, as opposed to the full range of GUI capabilities in Visual C++ or Visual Basic. For this reason, CNC language users usually use the U600 MMI to control their programs from a visual environment, or they write their own library interface GUI application that controls the CNC.

Although the CNC language is not as rich as C or C++, it should be mentioned here that the AEROTECH U600 CNC language goes far beyond the RS-274 language in providing language structures normally only found in languages such as C. Refer to the *UNIDEX 600 CNC Programming Manual Win NT/95, P/N EDU 158* for more details.

- Block structures (if-endif, while-endwhile, etc.)
- Subroutines with parameters, returns and full stack capabilities
- Full access to all parameters and I/O (digital and analog)
- Full "define" capabilities, allowing for user-defined G-codes or M-codes.

In addition the AEROTECH CNC language offers a wide range of motion capabilities beyond the RS-274 standard:

- Coordinate system rotation and mirroring (G83, G84)
- Asynchronous Motion (STRM)
- Up to sixteen spindles (S word)
- Simultaneous rotational and linear movement (G98, G99)
- Cutter Compensation (G40)
- Normalcy Motion (G20)
- PSO support (PSOC)
- Data Collection (DATACOLLECT)
- HandWheel Support (HAND)
- Analog and Digital Probe Support (PROBE)
- Disk file I/O (FILEOPEN)
- Continuous I/O monitoring (ON/ONGOSUB)

Unlike the library interface, modifications to the sequence and types of motion executed can be made at any time by the user by modifying the G-code program and re-downloading the file to the UNIDEX 600 controller (using the U600 MMI program). Also, the end-user can stop and restart applications at will with the U600 MMI application. This may or may not be an advantage depending on the particular application.

One disadvantage of the CNC interface is that, the axis processor, unlike the PC processor, does not have virtual memory, so there is a strict limit to the size of a CNC program that can be run. However, there is a circular program buffer available, for programs exceeding the size of the available memory in the controller. This buffer allows execution of programs of infinite size by having the PC download new lines as the old ones execute. However, there are important restrictions on a program executed in such a fashion: it must not contain any **GOTO**s or jumps (it must run strictly sequentially through the lines in the program).

Another disadvantage is neither axis configuration nor CNC program control functions are available from the CNC language. However, Aerotech has created an application program that serves as the Man-Machine Interface (MMI600-NT) used to configure axes and control/monitor CNC program execution. This functionality is available from the line interface AerDebug utility.

3.3.1. CNC Tasks and Programs

The CNC engine running on the axis processor runs four execution threads concurrently, each of which can be running a CNC program independently. In addition, each task can execute a single immediate command, while running its CNC program (however, the command set available for immediate commands is limited to those that do not reference other program locations, or use program variables). The concurrent execution is accomplished by polling through the tasks sequentially, once for each poll cycle.

Many CNC commands take significant fixed time periods to execute (like a **G4** or **DWELL**), in these cases, the task gives up its execution time to the next task, until the allotted time passed. Therefore, time consuming statements in one task do not slow down processing of the other tasks.

Tasks can run any program regardless of what programs are being run by other tasks. Two tasks can even run the same program at the same time. Programs can freely call other programs by using the **FARCALL** CNC command and there is no limit to the nesting of such calls. Also allowed is recursion, where programs can call themselves or any program that called it. The axis processor can contain up to 100 programs at one time.

The programmer can use global variables or I/O to coordinate execution between the tasks. Global variables and I/O, as well as all parameters, have global scope. Meaning, they are equally accessible to all programs. For example, one task may run a program that begins by looping forever, waiting for a particular output to be set. Another program might run motion, and only set the output bit after that motion is done. In this way the two programs are coordinated, so that the first program cannot proceed until the second program completes its motion. The user need not worry about semaphores to arbitrate simultaneous access to globally scoped objects, the CNC engine ensures that no task can read or write a global scoped value while another task is writing or reading it.

Each task has a set of task variables that are not available to the other tasks. Task variable 1, for example, is a different variable for each of the four tasks. Task variables are shared among all programs running on that task. That is, task variable 1 is the same variable for any program running on task 1, but refers to a different variable for any program running on task 2.

Program variables have the lowest scope, existing only in the particular program running on a particular task. Even programs called by that program cannot access these variables.

Please see the *UNIDEX 600 Series CNC Programming Manual, Win NT/95, P/N EDU158* under Chapter 3, for more details on variable usage and scope.

3.3.2. CNC Program Execution

Although CNC programs run on the axis processor, the programmer must load, invoke and run CNC programs from the PC. The CNC programmer can invoke and control CNC program execution through the U600 MMI, AerDebug, or a custom library calling application.

In either case, there are a number of steps that must be performed, see Table 3-3. In general, they must be performed in the order listed. Note that the U600 MMI hides most of the details, see the U600 MMI online help file for simplified instructions on how to control programs from the U600 MMI.

Table 3-3. How to Run and Control CNC Programs

Action	AerDebug	U600 MMI	Library Interface Functions
Compile	PRGC	Run Screen, compile button	AerCompilerxxx()
Download	PRGL	Run Screen, compile button	AerCompilerDownLoad()
Associate to a task	TSKA	<NA>	AerTaskProgramAssociate()
Set Starting line	TSKPRG	Run Screen	AerTaskSetLineUser()
Execute program	TSKPRG/EXEP	Run Screen, Cycle Start button	AerTaskProgramExecute()
Stop Program	TSKPRG	Run Screen, Cycle Stop button	AerTaskProgramStop()
Reset Program	TSKPRG	Run Screen, Reset button	AerTaskProgramReset()

3.3.3. Motion from a CNC Program

The user must first configure the axes in order to execute motion. Refer to Chapter 2 for this procedure.

To invoke motion from the CNC each task must reserve axes for it's own use from the 16 axes available. The default names X/Y/Z/U/A/B/C/D/x/y/z/u/a/b/c/d are defined for each task, and are called "task axes."

Before executing motion, the user must bind the task axis to a physical channel number, and then claim control of that task axis. The typical CNC application always consists of the following:

```
MAP X1 Y2 Z3 U4          ; Assign a physical axis channel to a task axis
BIND X Y Z U             ; Declare ownership of a task axis to a task. Implies
                         ; the task owns the physical channel in which the axis
                         ; is mapped
                         ; Do motion here, such as a G1 X100 Y1000
FREE  X Y Z U            ; Release ownership of a task axis
```

Once a physical channel is bound by the **BIND** command on one task, it cannot be bound with a **BIND** command in another task. The axis must first be freed with the **FREE** command before it can be bound by another task. However, two different tasks can freely use the same task letters, as long as each task binds that letter to a different channel number.

∇ ∇ ∇

CHAPTER 4: AERDEBUG

In This Section:

4.1. Introduction

AerDebug is a command line oriented program that can be used for examining or controlling the UNIDEX 600 Series axis processor card. AerDebug is a WIN32 application that runs on the host PC. However, it is not a "Windows" program (it was built as a "console application"). The AerDebug user simply types in command strings, and the output, if any, is delivered to the terminal. Also, AerDebug offers a simple help facility (no context-oriented help) for its commands.

AerDebug covers a broad range of the axis processor capabilities including direct memory access and monitoring, CNC program handling, and parameter viewing access. AerDebug is the major debugging tool used by Aerotech to develop new axis processor functions and has become a robust and user-friendly program. AerDebug can safely run concurrently with any other application that communicates with the axis processor card, and is useful for monitoring the effects of that application on the axis processor. For example, the user can continuously monitor axis positions while motion is directed by the test application.

Also, AerDebug allows the user to read its input from text files. This permits the user to construct files containing multiple commands and execute them as if they were one command. Refer to the *PLAY* and *PLYREWIND* commands in Section 4.3. for more information. AerDebug allows the user to echo output to a text file, including prompts, user entered text, and delivered output. The user can also specify a "silent" or "block terminal" mode where the outputs are only delivered to the file, not to the terminal.

The following are the allowable command line execution options that must be given after the "aerdebug" command. Any number of options can be on the command line.

Command Line Options: (# is a number, filespec is a valid path and filename)

-D###	UNIDEX Device Identification (where ### is 600)
-C#	Card # (where # is 1 - 4)
-Ifilespec	Playback file on startup in continuous mode
-Jfilespec	Playback file on startup in step mode
-Ofilespec	Start up with OUTPUTON command active (echo mode)
-Pfilespec	Start up with OUTPUTON command active (block terminal mode)

EXAMPLE

```
AerDebug -JSetup.ply        ; run PLAY file commands in setup.ply
                            ; file in step mode
```

4.2. The Screen

The AerDebug screen is divided into three parts: DATA, HELP, and STATUS. Refer to Figure 4-1.

The DATA screen has a black background and contains the prompt line as well as the last twenty or so data lines printed. The user enters commands on the prompt line and views data returned by the command. The command on the lowest prompt line is the current command. Text on the DATA screen typed by the user appears in green, while text typed by AerDebug in response to commands is in red. When the data on the DATA screen exceeds the height of the screen, the screen scrolls up and the topmost lines are discarded.

The HELP screen has a BLUE background and is normally blank. However, if the user enters "?", then the requested help data is printed to the screen. This help data will remain on the screen until the next set of help data is requested. The type of help data returned depends on the text preceding the "?" on the command line.

The STATUS screen consists of a single line at the bottom of the screen. Its purpose is to indicate important conditions to the user and to echo partial command matches.

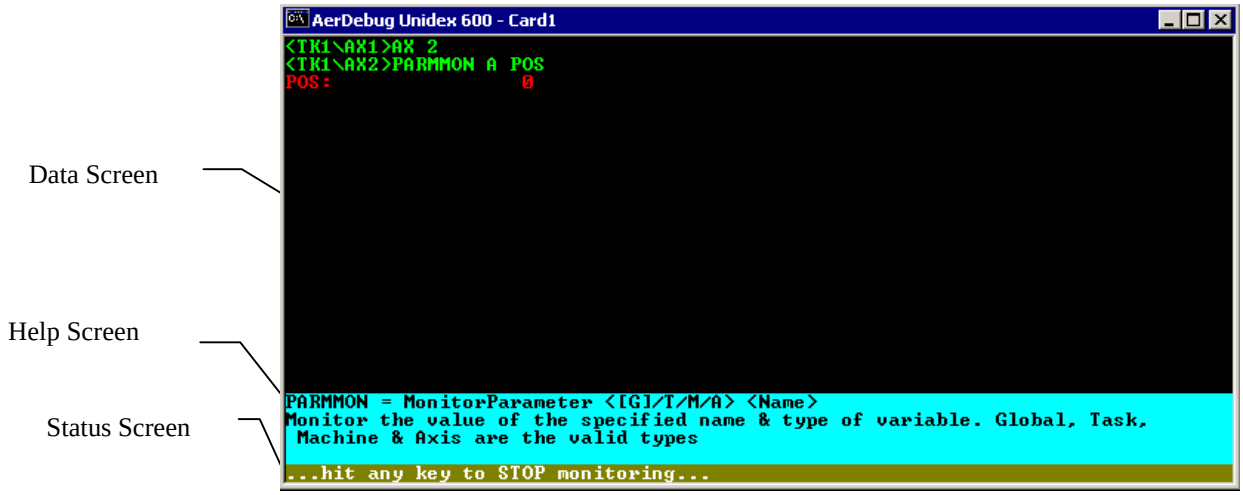


Figure 4-1. AerDebug Screen

4.3. The Prompt

Some commands apply only to an axis or task (for example, the command "*PARMGET A"* retrieves axis parameters). The user does not specify the task or axis for these, instead the default axis or task is used. Either the default task or the default axis is shown in the prompt preceding the command line. Use the *TK* and *AX* commands to change the default task or axis. If the user does not specify a task or axis as an argument to these commands, they simply switch the prompt to the default axis or task. View the following series of commands for an example.

Prompt and command are...	Defaults after command executed are....
	axis 1, task 1
AX1>TK 4	axis 1, task 4
TK4>AX 16	axis 16, task 4
AX16>TK	axis 16, task 4
TK4>AX	axis 16, task 4
AX16>	

4.3.1. Entering Commands

The command line is the line the cursor is on or the line the user is currently entering text on. Alphabetic keystrokes (a to z, 0 to 9, "-", "+", ".") result in the corresponding character being echoed on the command line and the cursor moving one character to the right. Non-alphabetic keystrokes (i.e. "$","#", Esc) are considered special characters and may or may not be echoed. Also, the characters "_", "\", and ":" are considered alphabetic and may be used in filenames. The case of alphabetic characters are ignored, lowercase characters are echoed back as upper case. After the command is fully entered, the user can hit the enter (return) key to execute the command.

As the user types, the status line will display all AerDebug commands that match the current text. For example, if *TSK* has been typed on the command line, the status line will display: "TSKASSOC, TSKDEASSOC, TSKINFO..." which are all valid AerDebug commands starting with *TSK*.

If the text on the line matches only one valid command, then a single line description of the command appears to the right of the command. For example, if *TSKA* is on the command line, then "TSKASSOC - Associates program with current task..." appears on the task line.

Once the text on the command line matches a single valid command, the user does not need to type the rest of the command. For example, "TSKA" or "TSKASS" or "TSKASSOC" are all equivalent.

In most cases, if the user provides an invalid parameter and hits return, AerDebug will reject the return (not execute anything) and print an error message in the status bar. For example, if the user types "AX 44" and hits return, Aerdebug will ignore the return key because 44 is not a valid axis.

4.3.2. Special Keys

The following keys are alphabetic and are merely echoed on the command line: a to z, A to Z, 0 to 9, +, -, _, :, \, and (.). All other keys are considered special keys.

All special keys are explained in Table 4-1, any other keys are ignored and not echoed to the command line. For example, the "}" key has no special meaning, therefore it will be ignored. However, the " character can be used to turn off recognition of some special characters. For example, the user could type "?{" without the ? invoking help (this is necessary in specifying string variable values, or strings to be compiled).

Special characters listed in Table 4-1 that have corresponding character representations will also be echoed on the command line.

Also, the keypad equivalents of keys will not work, (arrows, home etc. on keypad do nothing).

Table 4-1. AerDebug Special Character Keys

KEY	MEANING
Enter	Execute the command line. After execution move down a line, and open new prompt line.
Esc	Discontinue the multiple screen output, or discontinue monitoring (returns to prompt)
←	Move cursor one character to left in command line
→	Move cursor one character to right in command line
↑	Make the previous command the command line (erases current command line)
↓	Make the next command the command line (erases current command line)
Home	Move cursor to beginning of command line
End	Move cursor to end of command line
Insert	Toggles insert/overstrike mode
Delete	Deletes character at cursor, moves cursor one character to left
PageUp	Make the previous "PLAY" command the current command (erases current command line)
?	Retrieve help for the current command line
"	All ASCII characters between "" are NOT interpreted as a special character.
!	As first character in line, prefixes a direct memory access command (See Command Summary).

4.3.3. Help

Typing "?" at the command prompt will produce a list of the available *Aerdebug* commands. Most directly correspond to Aerotech library functions described within the UNIDEX 600 Series Library Reference Manual. Section 4.7., in this chapter, provides a cross reference between the commands and the library functions. To receive more help on a command the user may type that command followed by a question mark (?); (ie, "CMDERR ?", will display a help screen for the CMDERR command). Refer to Figure 4-2 for an example.

EXAMPLES

?	; Display help descriptions for all commands
PRGCMPL ?	; Display help for PRGCMPL command

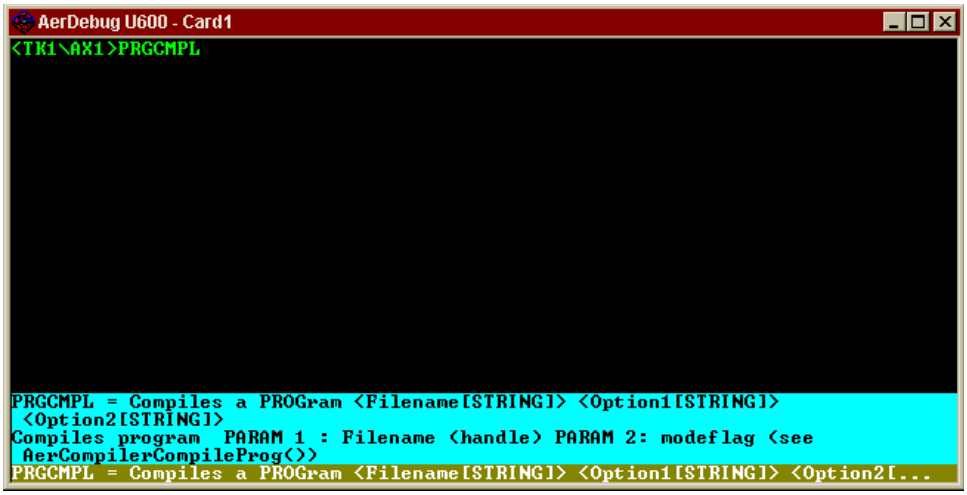


Figure 4-2. Help Screen (AerDebug)

4.4. Axis and Faultmask Configurations

Before an axis can be enabled and commanded to move, the axis must be configured for a D/A channel to the type of feedback device present and its parameters set defining the axis servo loop gains, faults, etc. Configuring a D/A channel for an axis is done with the *CONFIGD2A* command. Configuring feedback on an axis is accomplished with one of the following commands:

- *CONFIGRESOLVER* for resolver type feedback
- *CONFIGENCODER* for encoder feedback
- *CONFIGHRESOLVER* for resolver feedback with Hall effect sensors
- *CONFIGHENCODER* for encoder feedback with Hall effect sensors.

An axis can not be configured if the drive is enabled. (For information on the *DRIVE* parameter, refer to the *UNIDEX 600 Series Library Reference Manual, P/N EDU156* or the U600 MMI online help file). The parameter monitor command (*PARMMON*) in AerDebug is very useful for configuring and debuging the axes hardware in a system. For axes that are already configured, the user can use the parameter monitor command to observe axis position (*PARMMON A POS*). This will verify that the feedback for that axis is present and phased properly by manually moving the axis and observing the position changes.

The UNIDEX 600\650 expect positive velocity (clockwise motor rotation) feedback for a negative polarity torque command.

Also, the resolution of an axis can be confirmed using this method. To do this, move the axis a known amount (one revolution, one inch, etc.) and take note of the change in the monitored position.

To enable an axis, it is necessary to use the *PARMSET A DRIVE 1* command. If the *DRIVE* parameter is set to zero (0), then the axis is disabled. In order to enable an axis, the *DRIVE* parameter must be set to a one (1). If the state is reversed, use the *IOLEVEL* axis parameter to invert the state. Before enabling the drive, properly set the *FAULTMASK*, *DISABLEMASK*, *HALTMASK*, *ABORTMASK*, *BRAKEMASK*, *INTMASK*, and *AUXMASK* parameters.

Limiting the current command to the motor's continuous current rating is recommended when operating an axis for the first time. This can be done by setting the *IMAX* parameter. The default for the *IMAX* parameter is set to the maximum allowable current command. Setting *IMAX* to a value less than the default limits the current command to the drive.

After an axis has been configured and the operation of the feedback is confirmed, the axis servo loop gains can then be tuned for proper servo operation using the *AerTunet* Utility provided with the system. Refer to Chapter 5: AerTune.

4.4.1. Configuring an Axis

On initial power up or reset of the axis processor, all axis parameters and configuration data is cleared and reset to their internal defaults. To configure the axes, the user must specify axis configuration parameters such as axis type, feedback channel, and D/A channel. Also, fault masks and other axis parameters are important for configuring the axes.

Configuring the axis feedback type is accomplished by issuing one of the commands explained in the sections that follow.

All the arguments presented for the following commands are optional. The default values for the arguments are loaded by typing the command name, then pressing the **<ENTER>** key.

4.4.1.1. CONFIGRESOLVER - Resolver or Inductosyn Feedback

To configure a DC brush or an AC brushless motor using Resolver/Inductosyn feedback the following syntax applies:

CONFIGRESOLVER reschannel resolution poles offset bounded

where

reschannel	Resolver channel for position/velocity feedback. The default is the current axis.
resolution	Resolver to digital conversion resolution (10, 12, 14, or 16 bits). The default is 14 bits.
poles	Number of electrical poles for a brushless motor (32 max), set to zero for DC axis. The default is zero (0).
offset	Commutation offset for brushless motors (1,024 counts per 360° offset). The default is zero (0).
bounded	Enable (bounded=1), disable (bounded=0) software limits. The default is zero (0).
R2Dchannel	Resolver-to-digital (R/D) conversion channel (1 to 16 depending on number of installed R/D cards) for position feedback. The default is the current axis.

If specifying the number of bits for the resolution, it is also necessary to specify a value for the number of poles. Omission of any or all of the above arguments will cause default values to be used.

4.4.1.2. CONFIGENCODER - Encoder Feedback

To configure an axis with encoder feedback, the following syntax applies:

CONFIGENCODER encchannel lines bounded

where

encchannel	Encoder channel for servo position feedback. The default is the current axis.
lines	Encoder counts per motor rev. The default is 4000.
bounded	Enable (bounded=1), disable (bounded=0) software limits. The default is zero (0).

4.4.1.3. CONFIGHENCODER - Encoder and Hall Effect Sensor Feedback

To configure an AC brushless motor using Hall effect and encoder feedback for commutation, the following syntax applies:

CONFIGHENCODER encchanne lines_per_rev hall_lines com_offset comm_ch bounded

where

encchannel	Encoder channel for servo position feedback. The default is the current axis.
lines_per_rev	Encoder counts per motor revolution. The default is 4,000.
hall_lines	Encoder counts per electrical cycle. If set to zero (0), commutation defaults to six step. The default is 1000.
com_offset	Commutation offset for CONFIGHALL is specified in degrees (-360° ≤ offset ≤ 360°). The units are equal to 0 through 16,384 for 0 to 360°.
comm_ch	Encoder channel used to provide Hall effect and encoder commutation data. The default is the current axis.
bounded	Enable (bounded=1), disable (bounded=0) software limits. The default is zero (0).

4.4.1.4. CONFIGHRESOLVER - Resolver and Hall effect sensor feedback

To configure an AC brushless motor using Hall effect and resolver feedback for commutation, the following syntax applies:

CONFIGHRESOLVER reschannel resolution hall_lines com_offset comm_ch bounded

where

reschannel	Resolver channel for position/velocity feedback. The default is the current axis.
resolution	10, 12, 14 or 16 bits, which equals 1024, 4096, 16384 or 65535 counts per motor revolution.
hall_lines	Encoder counts per electrical cycle. If set to zero (0), commutation defaults to six step. The default is 1,500.
com_offset	Commutation offset for CONFIGHALL is specified in degrees (-360° ≤ offset ≤ 360°). The units are equal to 0 through 16,384 for 0 to 360°.
comm_ch	Hall channel used to provide Hall effect and commutation data. The default is the current axis.
bounded	Enable (bounded=1), disable (bounded=0) software limits. The default is zero (0).

4.4.1.5. ConfigD2A - Configure a DAC (D/A) Channel for Use by This Axis

Each axis must have a D/A channel assigned to it for the command (torque/velocity) to be delivered to the servo amplifier. This is done by specifying a D/A channel or omitting a channel number and the default channel will be assigned for that axis, based upon the axis number. The following syntax applies:

CONFIGD2A [channel]

where

channel	Channel specifies the D/A channel to be assigned to this axis.

4.4.1.6. ConfigRead - Read an Axis Configuration From a File

Axes may be configured quickly, once a configuration file has been created, preferably by the *ConfigWrite* command (the file format is documented in the UNIDEX 600 Series Library Manual). The *ConfigRead* command will read the configuration of the current axis from the file, as indicated by the AerDebug prompt and configure the axis.

The file will be read from the current directory unless a path is provided in the filespec. A file extension must be specified; an .INI file extension is not assumed.

CONFIGREAD (filespec)

where,

filespec	Filespec specifies the name of the configuration file to read the axis configuration from.

4.4.1.7. ConfigWrite - Write an Axis Configuration to a File

After initially configuring the axes manually, the configuration may be saved to a configuration file for recalling that configuration at a later time. The *ConfigWrite* command will save the configuration of the current axis, as indicated by the AerDebug prompt, for later use by the *ConfigRead* command to restore that configuration, after power-up or resetting the UNIDEX 600 Series controller.

The file will be written to the current directory unless a path is provided in the filespec. The file will have the file extension specified by the filespec. A new or existing file may be specified. The same file may contain configurations for multiple axes.

CONFIGWRITE (filespec)

where

filespec	Filespec specifies the name of the configuration file to write to.

4.4.2. Faults, Errors, and Faultmasks

Each axis uses a series of masks to enable or disable faults and to invoke protective measures (e.g., drive disabling or halting axis motion). These masks are *FAULTMASK*, *HALTMASK*, *DISABLEMASK*, *INTMASK*, *AUXMASK*, *ABORTMASK*, and *BRAKEMASK*.

The occurrence of axis faults and programming errors are not immediately reported to the user through AerDebug.

Notification does not occur because AerDebug would have to set the *INTMASK* to enable interrupts and claim that hardware interrupt under Windows NT/95. Enabling and claiming the interrupt may conflict with a user's application causing undesirable and unpredictable results. For a full description of faultmasks, refer to Chapter 2: Getting Started, Section 2.6.2.

4.4.2.1. Acknowledging (and Clearing) Faults

AerDebug does not inform the user when the fault occurs. The user must query the system with the *TskInfo* command or the *PARMGET A FAULT* command, which returns any active faults. Faults are acknowledged or cleared by using the *PARMSET A FAULT* command.

For example, if a position error and CW limit fault are active, the *PARMGET A FAULT* command would display:

```
FAULT: 5.000000
```

indicating bit 0 and bit 2 were set. To acknowledge and clear both of these faults, at the axis prompt, the user would enter:

```
PARMSET A FAULT 5
```

to clear bits 0 and 2. After a fault is cleared, there is no longer a record that it occurred.

The only way to remove fault conditions is to acknowledge them.

The user may also monitor a fault using the P*ARMMON A FAULT* command.

4.5. Programming Errors

Programming errors result when parameters are set outside their minimum or maximum allowable limits. They also occur when incorrect arguments are specified in configuration commands. To display programming errors, type *GETPROG* at the axis command prompt, then press the **<Enter>** key. If an error is present, AerDebug responds with diagnostic information that pertains to the type of error and the data received by the axis card. See the *U600 Series Library Reference Manual, P/N EDU156* under the introduction, for more details on programming errors.

Assume a programming error has been generated from setting a proportional gain (KP) parameter beyond its maximum limit. Typing *GETPROG* at the axis prompt and pressing **<Enter>** will reveal the following message.

(5): Parameter value too high

11 0 FF FF FF FF

Only the error message string provides useful information. The other information is used for internal diagnostic purposes only. Typing *GETPROG* will also clear and acknowledge any programming errors present.

If multiple programming errors have occurred, only the last error will be displayed by the *GETPROG* command. To allow all errors to be reported, it is up to the user to include a statement in the program that (e.g., setting the bit masks) allows the user to stop motion, do a status inquiry and view each error as it occurs.

4.5.1. Running CNC Programs

AerDebug allows the user to compile, download, and run CNC programs or single CNC lines. After AerDebug begins executing a CNC program, it returns the user to the prompt - AerDebug does not wait for the program to complete before returning to the prompt and allowing the user to enter new commands (see the *WAIT* command in AerDebug to force AerDebug to wait for program completion).

Therefore, the user can initiate programs on other tasks from AerDebug while the current task is busy running a program. However, each task can only execute one program at a time. The user can specify an immediate command to be executed while any or all other tasks are running a program. However, the commands that may be executed in the immediate mode are limited to those commands that are asynchronous. Asynchronous commands are commands that would finish execution immediately, meaning, they would have a defined cycle time no longer than the time required by the UNIDEX 600 Series controller to begin the command. This disallows commands such as G1 moves.

The *EXELINE* command executes single CNC lines in immediate mode.

EXAMPLE:

```
EXELINE "G70"          ; Set English programming mode active
```

Executing a CNC program on the UNIDEX 600 Series controller requires four steps. The program must first be compiled (converted) to a binary file that can be downloaded to the axis processor card for execution. Following the compile process, the binary equivalent of the users CNC program must be downloaded (transferred) to the axis processor card for

execution. This process loads the program into memory on the axis processor. The axis processor has the capability to run up to four simultaneous CNC programs, so the program must be associated with one of four tasks after it has been loaded into memory.

For information on CNC commands and their syntax reference the *UNIDEX 600 Series Programming WIN 95/NT Manual, P/N EDU158*.

EXAMPLES:

```
PRGCMPL \U600\TEST.PGM 10  ; Compile test.pgm with option "10",
                           ;  (see PRGCMPL for list of options).
PRGLOAD \U600\TEST.PGM     ; Load the program into axis processor.
TSKASSOC \U600\TEST.PGM    ; Associate the program with this task.
PRGRUN  \U600\TEST.PGM     ; Run the program.
```

A simpler command exists for performing all four steps at once. However, it does not allow for specifying different options on the program compile and load. The program runs on the current task.

EXAMPLE:

```
EXEPRG \U600\TEST.PGM
```

The user can execute a single CNC line as a CNC program. This is useful since many CNC commands are illegal in immediate mode. The process is similar to that of running a CNC program read from a file.

EXAMPLES:

```
PRG1 "G70"          ; Compile/load as program "?"
PRGLOAD "?"         ; Load compiled program
TSKASSOC "?"        ; Associate program "?" with this task
PRGRUN "?"          ; Run the line
```

There are two types of line numbers in use, user line numbers and axis processor line numbers.

Axis processor line numbers are only used by the axis processor. They are shown as the first number in each line, in the listing given by *PRGDUMP*. Axis processor line numbers start at 0 and increase one for each line downloaded. "User line numbers" are the line numbers assigned by the CNC compiler. These can be seen as the number in brackets on each line in the *PRGDUMP* program listing. These start at one and generally are incremented one for each line in the source. However, some single source lines compile into multiple axis processor lines. For example, a "G1 F4" will compile into two axis processor lines. In these cases the multiple axis processor lines will be given the same user line number.

The axis processor uses user line numbers within the context of a step (see the *TSKPRG* command). A step command will execute all lines with the same user line number. However, when downloading a normal program, the first user line number is always assumed to be 1.

When the user downloads to a queue, the user must provide the starting user line number in the PRGLOAD.

EXAMPLE:

```
ORIGINAL  (PRGTYPE LISTING)
        G1 F4   ; this will generate two statements: a G1 and a TASKPARM[35]=4
      ;         this line has no code on it
        $ glob[0]=0
COMPILED  (PRGDUMP LISTING)
        0 [1] TASKPARM[35] = 4.000000
        1 [1] G1
        2 [2] $DBL.GLOB0 = 0.000000
```

Programs can also be downloaded as circular queues. Queues are intended for situations where not all lines are available at once. Lines can be compiled and downloaded, then at a later time more lines can be compiled and appended to the program.

An important feature of queues is they are circular, so if the downloading is properly coordinated with the execution, then queues can execute programs of infinite length. After each line is executed, it is discarded, this makes a line available to download another line. If lines are executed faster than they are downloaded, the queue becomes empty.

The user must declare the size of a circular queue. When the axis processor has reached the end of the queue, then it starts executing the queue at the beginning. After each line is executed, that line becomes available for loading of another line. Therefore, execution will not end until an explicit M02 is seen. The axis processor indicates if the queue is full, or empty in the status.

However, queues have certain disadvantages: any statement with a jump cannot be downloaded into a queue, because the jump may refer to a line that is not currently in the queue. This includes all if, while, and repeat statements, as well as M47 and M30 codes. Here is an example of how a queue can be used, a series of lines are compiled and downloaded.

```
PRG1 "$global0=1"
PRGL "?" 3 0        ; Queue has 3 lines, download this as userline 1 in the queue
PRG1 "$global0=2"
PRGL "?" 0 1        ; Download this as userline 2 in the queue
PRG1 "$global0=3"
PRGL "?" 0 2        ; Download this as userline 3 in the queue
TSKA "?"            ; Associate
PRGD "?"            ; Dump out all the source (3 lines above)
TSKPRG E 1          ; Execute, stepover, first line of the queue program. This will free a
                    ; queue line to load another line.
PRG1 "$global0=4"
PRGL "?" 0 2        ; Download this as userline 4 in the queue
```

4.6. Programming Commands

The AerDebug commands are listed alphabetically in Table 4-2 in four groups, along with the parameters and a short description of the command. More help is available on-line, by using the "?" (refer to Sections 4.3.3. and 4.6.1.). AerDebug.exe ignores case.

Table 4-2. AerDebug Commands

Basic Commands	Description	Parameters
?	Display help (List of commands).	None
AX	Changes the default axis (if no parameter shows the default axis).	[Axis number]
CMDERR	Display the last command error.	Axis number
CMDLAST	Retrieve the last command from the command line buffer.	None
DOWNLOAD	Load firmware into axis card.	None
EXIT	Quits the AerDebug application.	None
MEM	Selects MEMORY command prompt mode.	None
OUTON	Sets output to be written to the specified to file.	Filespec
OUTOFF	Disables output from being written to a file.	None
OUTPAUSE	Suppresses/unsupresses output dumping to file(toggle).	None
PLAY	Executes the commands from within the specified file.	Filespec
PLYREWIND	Rewinds the PLAY file to the start.	None
RDO		
RGINFO	Display operating system registry information.	[device_id, card_num]
QUIT	Exits the AerDebug application.	None
RESET	RESET the Axis Processor card.	None
TK	Changes the default task (if no parameter shows the default task).	Task number
Memory Commands	**Description**	**Parameters**
!	Specifer to memory operations to operate on memory without interaction with the Axis Processor firmware.	RB through ML shown below.
!TI	Tests the PC interrupt. Reports "no error" or error message, if any.	None
DB	Display 128 bytes beginning at the specified address.	Address
DW	Display 64 words beginning at the specified address.	Address
DL	Display 32 longwords beginning at the specified address.	Address
MB	Monitor byte at specified address.	Address.
MW	Monitor word at specified address.	Address
ML	Monitor longword at specified address.	Address
RB	Read a byte from the specified address.	Address
RW	Read a word from the specified address.	Address
RL	Read a longword from the specified address.	Address
WB	Write the byte at the specified address.	Address, Data
WW	Write the word at the specified address.	Address, Data
WL	Write the longword at the specified address.	Address, Data

Table 4-2. AerDebug Commands (Cont'd)

Axis Commands	Description	Parameters
CONFIGD2A	Configure a DAC channel for this axis.	channel #
CONFIGENCODER	Configure an encoder feedback channel for this axis.	channel #, lines, bounded
CONFIGHENCODER	Configure an encoder feedback channel with hall effect sensors for this axis.	See command description
CONFIGRESOLVER	Configure a resolver feedback channel for this axis.	See command description
CONFIGHRESOLVER	Configure a resolver feedback channel with hall effect sensors for this axis.	See command description
CONFIGREAD	Configure an axis from an .INI file.	filespec
CONFIGWRITE	Write the axis configuration to an .INI file.	filespec
DCAX	Axis data center data.	None
DRVINFO	Display axis configuration information.	None
GETPROG	Display programming error and clear error condition.	None
INFO	Axis configuration information.	None
Task Commands	**Description**	**Parameters**
DIR	Directory of downloaded programs.	None
DUMPTABLE	Dumps CAM tables	Table number
DUMPERROR	Dumps error calibration table	Table number
ENABLEPENDANT	Enables the pendant with a given channel number	Channel number
EXELINE	Compile & Execute a single CNC program line.	CNC program line
EXEPRG	Compiles, loads, associates and executes a program.	Filespec
IOGET	Display the value of a virtual I/O point ([BI]/BO/RI/RO).	type, point
IOMON	Monitor the value of a virtual I/O point ([BI]/BO/RI/RO).	type, point
IOSET	Set the value of a virtual I/O point ([BI]/BO/RI/RO).	data
MABORT	Aborts motion on the current selected axis.	None
MABSOLUTE	Moves the axis to the absolute position.	Position, Speed
MALTHOME	Used reference the axis to an absolute reference point.	Direction, Speed
MFREERUN	Starts the currently selected axis moving in the specified direction.	Direction, Speed
MHALT	Decelerates motion on the currently selected axis to zero.	None
MHOLD	Feedholds the motion in progress on the currently active axis	None
MHOME	Homes the currently active axis in the specified direction.	Direction, Speed
MINCREMENTAL	Moves the currently axis in a specified incremental distance.	Distance, Speed
MINFEEDSLAVE	Starts the currently active axis moving the in the specified distance.	Distance, Speed
MNOLIMITHOME	Homes the specific axis in the direction and velocity specified.	Direction, Speed
MOSCILLATE	Continuously cycles the specified distance at a specific velocity.	Distance, Speed
MQABSOLUTE	Queued version of the *MABSOLUTE* function.	Position, Velocity
MQFLUSH	Clears the 16 level deep axis queue for the current axis.	None
MQHOLD	Places the queue for the selected axis into the hold state.	None
MQINCREMENTAL	Queued version of the *MQABSOLUTE* command.	Distance, Speed
MQUICKHOME	Starts the currently selected axis homing in the specified direction.	Direction, Speed

Table 4-2. AerDebug Commands (Cont'd)

Task Commands	Description	Parameters
MQRELEASE	Restarts the specified axis queue that was halted.	None
MRELEASE	Resumes the motion that was in progress on the selected axis.	None
PARMGET	Display the value of a parameter (G/T/M/A).	type, parameter
PARMMON	Monitor the value of a parameter (G/T/M/A).	type, parameter
PARMSET	Set the value of a parameter (G/T/M/A).	type, parameter
PRG1	Compile a single CNC program line.	CNC program line
PRGCMPL	Compile a CNC program.	Filespec
PRGDUMP	Display internal program code lines.	Filespec
PRGERRS	Shows program compile errors.	Filespec
PRGINFO	Displays program information.	Filespec
PRGLOAD	Loads a CNC program into the Axis Processor.	Filespec
PRGRUN	Runs a CNC program.	Filespec
PRGSTATS	Shows program compile status.	Filespec
PRGTYPE	Shows program test lines.	Filespec
PRGUNLOAD	Frees a CNC program.	Filespec
PSODOWNLOAD	Load firmware into PSO card	None
SPENDANTTEXT	Set text line on teach pendant	Ch, Line, Text
TSKASSOC	Associates a CNC program with the current task.	Filespec
TSKDEASSOC	Deassociates a CNC program from the current task.	None
TSKINFO	Displays information on the current task.	None
TSKPRG	Control program execution A/[E]/R/S/L.	mode, [param]
TSKRESET	Resets the current task	None
VAGET	Display axis point variable [G]/T.	number
VCGET	Display variables for task subroutine call stack.	None
VDGET	Display the value of a variable [G]/T/P/S.	type, #
VDMON	Monitor a variable (double) [G]/T/P/S.	type, #
VDSET	Set a variable (double) [G]/T/P/S.	type, #, value
VSGET	Display a string variable [G]/T/P.	type, #
VSMON	Monitor a string variable [G]/T/P.	type, #
VSSET	Set a string variable [G]/T/P.	type, #, text
WAIT	Wait on status.	[!] condition
WRITESERIAL	Writes to serial port	None
ZMONITOR	Display monitor data.	None
ZONGOSUB	Display ongosub data.	None

4.6.1. ? *or* (command) ?

The *?* command displays a list of valid AerDebug commands. For more help on an individual command type the command followed by a ?.

EXAMPLES:

?	; display all help
CMDERR ?	; display help on CMDERR

4.6.2. ! (memory_command)

The *!* specifier is used before the memory commands, such as *DB*, *DW*, *DL*, *MB*, *MW*, *ML*, *RB*, *RW*, *RL*, *WB*, *WW*, and *WL* specifying that the memory operation should not be performed via the firmware resident on the UNIDEX 600 Series controller. Instead, the operation will be performed by configuring the U600 memory interface hardware directly and performed solely under the control of the PC software. This permits lower level testing and debugging of the controller not normally performed by the user

EXAMPLES:

!DB 80c	; display byte of memory at 80c
!MW 10000	; monitor word of memory at 10000
!WL 20000 abce4231	; write long word of memory at 20000 to ; abce4231

4.6.3. ! TI

Tests the PC interrupt. Reports "no error" or error message, if any.

EXAMPLE:

!TI

4.6.4. AX (axis_number)

The *AX* command is used to display or change the current axis that the AerDebug commands act upon. Entering the *AX* command without an axis number will display the current axis number that the command will act upon. Specifying an axis number following the *AX* command will instruct AerDebug to direct all further commands to the specified axis. The range of valid axes is 1 through 16.

EXAMPLES:

AX	; displays current axis mode
AX 2	; sets current axis mode to axis 2

4.6.5. CMDERR (axis_number)

The *CMDERR* command displays the last command error that occurred for the specified axis number. The range of valid axes is 1 through 16. This information is not intended for the user. It is for debugging purposes only, since it displays the command error as an opcode and sub opcode with the byte count of the data transfer between the PC and UNIDEX 600 Series controller card.

Example:

```
CMDERR 3                    ; show last command error for axis 3
```

4.6.6. CMDLAST (axis_number)

The *CMDLAST* command displays the last command for the specified axis number. The range of valid axes is 1 through 16. This information is not intended for the user. It is for debugging purposes only, since it displays the command error as an opcode and sub opcode with the byte count of the data transfer between the PC and UNIDEX 600 Series controller card.

EXAMPLE:

```
CMDLAST 4                   ; show last command for axis 4
```

4.6.7. CONFIGD2A (D2A_channel_number)

The *CONFIGD2A* command allows the user to assign an analog output from a Digital-to-Analog-Converter (DAC) to an axis to be used as the command output of that axis. The valid range of DAC channels is determined by the number of DAC channels present in the user's system.

A UNIDEX 600/650 controller has 4 channels onboard with 4 additional channels provided by each additional encoder expansion card. Their channel numbers are determined by the expansion board number. Expansion board number one will be channels 5 through 8, board two will be channels 9 through 12, etc.

Attempting to configure an axis for a DAC channel that is not present will cause a programming error. This error will be indicated by the *GETPROG* command after the bad configuration attempt.

EXAMPLE:

```
CONFIGD2A 4                 ; current axis will be assigned D/A channel 4 to drive
                            ; its motor
```

4.6.8. CONFIGENCODER (encoder_ch) (lines_per_revolution) (bounded)

The *CONFIGENCODER* command allows the user to assign an encoder feedback channel to an axis to be used as the position and/or velocity feedback for that axis. The valid range of encoder feedback channels is determined by the number of encoder feedback channels present in the user system. The lines per revolution parameter indicates the number of lines per revolution of the encoder times 4. The UNIDEX 600 Series controller electronically multiplies the effective line count of the encoder by 4, so the lines per revolution entered should always be four times the physical line count of the encoder. The bounded parameter is used to activate software end of travel limits by entering a 1 for this parameter (0 to disable them).

A UNIDEX 600/650 controller has 4 channels onboard with 4 additional channels provided by each additional encoder expansion card. Their channel numbers are determined by the expansion board number. Expansion board number one will be channels 5 through 8, board two will be channels 9 through 12, etc.

Attempting to configure an axis for an encoder channel that is not present will cause a programming error. This error will be indicated by the *GETPROG* command after the bad configuration attempt.

EXAMPLE:

```
CONFIGENCODER 2     4000 0          ;current axis will be assigned
                                    ;encoder channel 2 to receive
                                    ;position/velocity feedback from,
                                    ;feedback will be 4000 counts per
                                    ;motor revolution, software limits
                                    ;are disabled
```

4.6.9. CONFIGHENCODER (encoder_ch) (lines_per_rev) (hall_lines) (com_offset) (comm_ch) (bounded)

The *CONFIGHENCODER* command allows the user to assign an encoder feedback channel to an axis to be used as the position and/or velocity feedback for that axis. The valid range of encoder feedback channels is determined by the number of encoder feedback channels present in the user's system. The lines per revolution parameter indicates the number of lines per revolution of the encoder times 4. The UNIDEX 600 Series controller electronically multiplies the effective line count of the encoder by 4, so the lines per revolution entered should always be four times the physical line count of the encoder. The *hall_lines* parameter specifies the number of encoder lines per electrical cycle of the motor. The *com_offset* parameter allows the resolver to be aligned to the motor phasing by specifying an offset to produce the required mechanical alignment. This offset is specified by a 14 bit number (0-16384), with 360 degrees being equal to 16384. The *comm_ch* parameter specifies the Hall effect feedback channel. The *bounded* parameter is used to activate software end of travel limits by entering a 1 for this parameter (0 to disable them).

A UNIDEX 600/650 controller has 4 channels onboard with 4 additional channels provided by each additional encoder expansion card. Their channel numbers are determined by the expansion board number. Expansion board number 1 will be channels 5 through 8, board 2 will be channels 9 through 12, etc.

Attempting to configure an axis for an encoder channel that is not present will cause a programming error. This error will be indicated by the *GETPROG* command after the bad configuration attempt.

EXAMPLE:

```
CONFIGENCODER 2     4000 1000 8192 2 0     ;current axis will be
                                           ;assigned encoder channel
                                           ;2 to receive
                                           ;position/velocity
                                           ;feedback from, feedback
                                           ;will be 4000 counts per
                                           ;motor revolution, there
                                           ;are 1000 encoder lines
                                           ;per electrical cycle, the
                                           ;resolver is 180 electrical
                                           ;degrees (8192) out of
                                           ;phase with the motor, hall
                                           ;effect channel 2 is used,
                                           ;software limits are
                                           ;disabled
```

4.6.10. CONFIGHRESOLVER (resolver_ch) (resolution) (hall_lines) (com_offset) (comm_ch)(bounded)

The *CONFIGHRESOLVER* command allows the user to assign a resolver feedback and Hall effect feedback channel to an axis to be used as the position and/or velocity feedback for that axis. The valid range of resolver feedback channels is determined by the number of resolver feedback cards present in the user's system. The *resolution* parameter specifies the desired counts per revolution of the motor that has been configured for the R/D card. This is entered as 10, 12, 14 or 16. These numbers represent binary powers of two (i.e., 2^{10}, 2^{12}, 2^{14}, or 2^{16}), which produce 1024, 4096, 16384 or 65536 counts per motor revolution. The *hall_lines* parameter specifies the number of encoder lines per electrical cycle of the motor. The *com_offset* parameter allows the resolver to be aligned to the motor phasing by specifying an offset to produce the required mechanical alignment. This offset is specified by a 14 bit number (0-16384), with 360 degrees being equal to 16384. The *comm_ch* parameter specifies the Hall effect feedback channel. The *bounded* parameter is used to activate software end-of-travel limits by entering a 1 for this parameter (0 to disable them).

UNIDEX 600 Series controllers have 4 channels provided by each resolver (R/D) card. Their channel numbers are determined by the R/D board number. Resolver board number one will be channels 1 through 4, board two will be channels 5 through 8, etc.

Attempting to configure an axis for a resolver channel that is not present will cause a programming error. This error will be indicated by the *GETPROG* command after the bad configuration attempt.

EXAMPLE:

```
CONFIGHRESOLVER 3 12 1000 8192 3 1   ;current axis will be
                                     ;assigned resolver
                                     ;channel 3 to receive
                                     ;position/velocity
                                     ;feedback from, feedback
                                     ;will be 12 bit (4096
                                     ;counts per motor
                                     ;revolution) there are 1000
                                     ;encoder lines per
                                     ;electrical cycle, the
                                     ;resolver is 180 electrical
                                     ;degrees (8192) out of
                                     ;phase  with the motor,
                                     ;hall effect channel 3 is
                                     ;used, software limits are
                                     ;enabled
```

4.6.11. CONFIGREAD (filespec)

The *CONFIGREAD* command configures an axis from the specified .INI file. The .INI file is written with the *CONFIGWRITE* command after manually configuring the axis the first time.

EXAMPLE:

```
AX 1                        ; Set axis to configure
CONFIGREAD AxisCfg.ini      ; configure axis 1 from AxisCfg.ini file
```

4.6.12. CONFIGRESOLVER (resolver_ch) (resolution) (poles) (com_offset)(bounded)

The *CONFIGRESOLVER* command allows the user to assign a resolver feedback channel to an axis to be used as the position and/or velocity feedback for that axis. The valid range of resolver feedback channels is determined by the number of resolver feedback cards present in the user's system. The *resolution* parameter specifies the desired counts per revolution of the motor that the user's R/D card has been configured for. This is entered as 10, 12, 14, or 16. These numbers represent binary powers of two (i.e., 2^{10}, 2^{12}, 2^{14} or 2^{16}), which produce 1024, 4096, 16384 or 65536 counts per motor revolution. The *com_offset* parameter allows the resolver to be aligned to the motor phasing by specifying an offset to produce the required mechanical alignment. This offset is specified by a 14 bit number (0-16384), with 360 degrees being equal to 16384. The *bounded* parameter is used to activate software end-of-travel limits by entering a 1 for this parameter (0 to disable them).

UNIDEX 600 Series controllers have 4 channels provided by each resolver (R/D) card. Their channel numbers are determined by the R/D board number. Resolver board number one will be channels 1 through 4, board two will be channels 5 through 8, etc.

Attempting to configure an axis for a resolver channel that is not present will cause a programming error. This error will be indicated by the *GETPROG* command after the bad configuration attempt.

EXAMPLE:

```
CONFIGRESOLVER 3 12 4 8192 1    ;current axis will be assigned
                                ;resolver channel 3 to receive
                                ;position/velocity feedback from,
                                ;feedback will be 12 bit (4096
                                ;counts per motor revolution), it is
                                ;driving a 4 pole motor, the
                                ;resolver is 180 electrical degrees
                                ;(8192) out of phase with the
                                ;motor, software limits are enabled
```

4.6.13. CONFIGWRITE (filespec)

The *CONFIGWRITE* command writes the currently selected axis configuration to the specified .INI file. The *CONFIGREAD* command may be used to restore the axis configuration from the file.

EXAMPLE:

```
CONFIGWRITE AxisCfg.ini      ; write currently selected axis
                             ; configuration to AxisCfg.ini file
```

4.6.14. DB (address)

The *DB* command displays the value of a byte at the specified address.

EXAMPLE:

```
DB 80c                       ; display byte of data at 80c
```

4.6.15. DCAX

The *DCAX* command displays the status, position, velocity, and type of axis for the currently selected axis (via the *AX* command). The status displayed is the fault, servo, and motion status for the axis. The type of axis indicates a linear or rotary axis as follows:

0 : Linear axis

1 : Rotary axis with modulo position rollover

2 : Rotary axis without modulo position rollover

EXAMPLE:

```
DCAX                         ; display status on current axis
```

4.6.16. DIR

The *DIR* command displays a directory of the programs on the UNIDEX 600 Series controller and the state of the program (compiled, downloaded, associated).

EXAMPLE:

```
DIR
```

4.6.17. DOWNLOAD

The *DOWNLOAD* command loads the axis firmware into the UNIDEX 600 Series controller card and initialize it to it's power up state. If the firmware has been previously downloaded the *RESET* command must be used to reset the axis processor card before another download takes place.

EXAMPLE:

```
DOWNLOAD          ; load axis processor firmware
```

4.6.18. DRVINFO

This command displays information on the configuration of the device driver.

EXAMPLE:

```
DRVINFO          ; display device driver configuration
```

4.6.19. DUMPTABLE (table number)

The *DUMPTABLE* command dumps CAM tables. See AerCamTablexxx function in the *UNIDEX 600 Library Reference Manual, P/N EDU156.*

EXAMPLE:

```
DUMPTABLE 0      ; dump CAM table
```

4.6.20. DUMPERROR (table number)

The *DUMPERROR* command dumps an error calibration table. This command has one parameter that is the table number (which is zero based indexed). See the AerAxisCalxxx functions in the *UNIDEX 600 Library Reference Manual, P/N EDU156.*

EXAMPLE:

```
DUMPERROR0       ; dump error table
```

4.6.21. DW (address)

The *DW* command displays the value of a word at the specified address.

EXAMPLE:

```
DW 80c           ; display word of data at 80c
```

4.6.22. DL (address)

The DL command displays the value of a long word at the specified address.

EXAMPLE:

```
DL 80c                   ; display a Long word of data at 80c
```

4.6.23. ENABLEPENDANT (channel) (mode 1)

The *ENABLEPENDANT* command enables the pendant with a given channel number. See AerPendantSetModexxx function in the *UNIDEX 600 Library Reference Manual, P/N EDU156.*

EXAMPLE:

```
ENABLEPENDANT    ; enable teach pendant
```

4.6.24. EXELINE ("command string")

The *EXELINE* command compiles and executes a single CNC program line. The command string to be executed must be placed within quotes.

Example:

```
EXELINE "BIND X"
```

4.6.25. EXEPRG (filespec)

The *EXEPRG* command compiles, loads, associates and executes a CNC program. This is the equivalent to the following command sequence: *PRGCMPL*, (*PRGUNLOAD*), *PRGLOAD*, *TSKD*, *TSKA*, *AND TSKP E*. If the program successfully completes execution, the message "No Error" will be displayed.

EXAMPLE:

```
EXEPRG C:\U600\PGM\TEST.PGM                ; run test.pgm
```

4.6.26. EXIT

The *EXIT* command terminates the AerDebug.exe application, as does the *QUIT* command.

EXAMPLE:

```
EXIT            ; Quit AerDebug
```

4.6.27. GETPROG

The *GETPROG* command displays the last programming error for the current axis, if any. Also, any error present will be cleared.

EXAMPLE:

```
GETPROG      ; display last error, if any!
```

4.6.28. INFO

The *INFO* command displays the DAC channel number assigned to the current axis and the encoder feedback channel and type of feedback (resolver, encoder, etc.) configured for the axis.

EXAMPLE:

```
INFO
```

4.6.29. IOGET (type) (point_number)

The *IOGET* command displays the value of a virtual I/O point. Inputs and outputs may be displayed of register and binary (bit) types. Binary types refer to bits, having two possible states, a logic 1 or 0. Registers are 16 bits having a valid data range of 0 through 65535. These 4 types are represented as follows:

BI : Binary inputs
BO : Binary outputs
RI : Register inputs
RO : Register outputs

The valid range of the virtual I/O point is dependent on the type. Register types have a valid range of 0 through 127. Binary (bit) types have a valid range of 0 through 511. Each type does not share this range among inputs and outputs, so there are 128 register inputs, 128 register outputs, 512 binary inputs, and 512 binary outputs. If a virtual I/O point number is not specified, all I/O points of the type specified will be displayed.

EXAMPLES:

```
IOGET BI 3          ; read binary input 3
IOGET BO 127        ; read state of binary output 127
IOGET RI 511        ; read register input 511
IOGET RO 0          ; read state of register output 0
```

4.6.30. IOMON (type) (point_number)

The *IOMON* command monitors the state of a virtual I/O point, updating the display approximately every 100 msec. (10 times /second). Inputs and outputs may be monitored of register and binary (bit) types. Binary types refer to bits, having two possible states, a logic 1 or 0. Registers are 16 bits, having a valid data range of 0 through 65535. These 4 types are represented as follows:

BI : Binary inputs
BO : Binary outputs
RI : Register inputs
RO : Register outputs

The valid range of the virtual I/O point is dependent on the type. Register types have a valid range of 0 through 127. Binary (bit) types have a valid range of 0 through 511. Each type does not share this range among inputs and outputs, so there are 128 register inputs, 128 register outputs, 512 binary inputs, and 512 binary outputs.

EXAMPLES:

```
IOMON BI 3          ; monitor binary input 3
IOMON BO 127        ; monitor the state of binary output 127
IOMON RI 511        ; monitor register input 511
IOMON RO 0          ; monitor the state of register output 0
```

4.6.31. IOSET (type) (point_number) (value)

The *IOSET* command writes to a virtual I/O point. Inputs and outputs may be of register and binary (bit) types. Virtual inputs may be set with this command, however, setting a virtual input that is mapped to a physical hardware bit will be overwritten by the state of the physical hardware bit within 1 msec. on the occurrence of the next I/O scan update. Binary types refer to bits having two possible states, a logic 1 or 0. Registers are 16 bits, having a valid data range of 0 through 65535. These 4 types are represented as the following:

BI : Binary inputs

BO : Binary outputs

RI : Register inputs

RO : Register outputs

The valid range of the virtual I/O point is dependent on the type. Register types have a valid range of 0 through 127. Binary (bit) types have a valid range of 0 through 511. Each type does not share this range among inputs and outputs, so there are 128 register inputs, 128 register outputs, 512 binary inputs and 512 binary outputs.

EXAMPLES:

```
IOSET BI 73 1          ; set the state of binary input 73 to 1
IOSET BO 0 0           ; set the state of binary output to 0
IOSET RI 511  65535    ; set the state of register input 511 to 65535
IOSET RO 0 12345       ; set the state of register output 0 to 12345
```

4.6.32. MABORT

The *MABORT* function aborts the motion on the current selected axis. However, this does not apply to motion of the axis within a CNC program. It applies only to the motion started by the move or motion commands beginning with an 'M', such as: *MABSOLUTE*, *MHOME*, etc. The axis will stop abruptly by having its commanded position set to the current position, creating instantaneous deceleration causing position over-shoot beyond the current position. The axis will then settle back to that position at where it was commanded to stop. The axis accelerates and decelerates in the mode, rate/time defined by the *Accel*, *Decel*, *AccelMode*, *DecelMode*, *AccelRate* and *DecelRate* axis parameters.

EXAMPLE:

```
MABORT                 ; stop motion on the current axis
```

4.6.33. MABSOLUTE (position) (speed)

The *MABSOLUTE* command moves the axis to the absolute position specified by the position parameter at the speed specified by the speed parameter. The absolute position is relative to the home position (see *MALTHOME, MHOMEQUICK*, etc.) or the position the axis was at when powering up the system. The machine absolute position registers are set to zero at power up, then they are reset to zero at the completion of a homing cycle.

The position parameter is in machine steps, it may be specified as positive or negative, and is in machine steps per second. The axis must be enabled before commanding it to move. This command should not be used if the axis is executing a command from within a CNC program. The axis will accelerate and decelerate in the mode, rate/time defined by the *Accel, Decel, AccelMode, DecelMode*, *AccelRate,* and *DecelRate* axis parameters.

EXAMPLE:

```
PARMSET A DRIVE 1          ; enable the drive!
MABSOLUTE 10000 2000       ; move to absolute 10,000 machine steps
                           ; position at 2,000 steps per second
```

4.6.34. MALTHOME (direction) (speed)

The *MALTHOME* command references the axis to an absolute reference point. The direction parameter is either 1 or -1 to specify a clockwise or counter-clockwise homing direction. The speed parameter is in machine steps per second. The axis must be enabled before commanding it to move. This command should not be used if the axis is executing a command from within a CNC program. The axis will accelerate and decelerate in the mode, rate/time defined by the *Accel, Decel, AccelMode, DecelMode*, *AccelRate* and *DecelRate* axis parameters.

The axis will proceed in the specified direction until encountering the home limit. Then it will reverse direction, waiting for the home limit to become false, at that point, it begins looking for the reference pulse (encoder marker pulse or resolve null). When it finds the reference pulse, it stops at that position and sets the axis position register equal to the *HOMEOFFSET* axis parameter. The user may set the *HOMEOFFSET* axis parameter in user units by first defining the axis type using the *Type* axis parameter and entering the home offset into the *HomeOffsetInch* (or *HomeOffsetDeg* for rotary axes) machine parameter. This will overwrite the *HOMEOFFSET* axis parameter, setting it to the correct value for the desired offset in machine steps.

EXAMPLE:

```
PARMSET A DRIVE 1          ; enable the drive!
MALTHOME -1 2000           ; home direction is counter-clockwise,
                           ; move at 2,000 steps per second
```

4.6.35. MFREERUN (direction) (speed)

The *MFREERUN* command starts the currently selected axis moving in the specified direction, at the specified speed. The direction is specified as –1 or 1 for counterclockwise or clockwise motion. The speed parameter is in machine steps per second. The axis must be enabled before commanding it to move. This command should not be used if the axis is executing a command from within a CNC program. The axis will accelerate and decelerate in the mode, rate/time defined by the *Accel*, *Decel*, *AccelMode*, *DecelMode*, *AccelRate* and DecelRate axis parameters.

This command is typically used to begin continuous motion on an axis without end-of-travel limits, such as a spindle.

EXAMPLE:

```
PARMSET A DRIVE 1          ; enable the drive!
MFREERUN 1 50000           ; freerun clockwise at 50,000 steps per
                           ; second
```

4.6.36. MHALT

The *MHALT* function decelerates the motion on the current selected axis to zero. However, this does not apply to motion of the axis within a CNC program. It applies only to the motion started by the move or motion commands beginning with an 'M', such as: *MABSOLUTE*, *MHOME*, etc. The axis will decelerate in the mode, rate/time defined by the *Decel*, *DecelMode* and *DecelRate* axis parameters.

EXAMPLE:

```
MHALT                      ; decelerate motion on the current axis to a stop
```

4.6.37. MHOLD

The *MHOLD* command feedholds the motion in progress on the currently active axis by decelerating the axis to a stop in the current mode (linear/sinusoidal, rate/time). The current mode is determined by the *decelmode* axis parameter. The current deceleration rate or time is determined by the *decelrate* and *decel* axis parameters, respectively. To restart the move that was in progress, the *MRELEASE* command should be used to accelerate the axis back up to the programmed velocity based upon the axis acceleration parameters. This function causes any motion commands issued to the specified axis before the function is called to be placed into a queue that is one level deep. After the releasing the feedhold, the last commanded move executes and ignores any others. It applies only to the motion started by the move or motion commands beginning with an 'M', such as: *MABSOLUTE*, *MHOME*, etc.

EXAMPLE:

```
MHOLD                      ; feedhold the currently selected axis
.
.
MRELEASE                   ; accelerate back up to speed of previous motion
```

4.6.38. MHOME (direction) (speed)

This command starts the currently active axis homing in the specified direction at the specified velocity. The home procedure is defined as follows: The axis begins moving in the specified home direction. When the axis crosses the home limit, it continues moving into the home limit until encountering the first encoder marker pulse (or resolver null), and set the position register equal to the *HOMEOFFSET* axis parameter. If it reaches an end-of-travel limit, it reverses direction. If it reaches a home limit switch, it will go past it, reverse direction, and come at it from the specified direction. If another end-of-travel limit is encountered, a homing fault occurs. The velocity is specified in machine steps per second. The direction is specified as –1 or 1 for counterclockwise or clockwise motion. The axis will accelerate and decelerate at the currently selected modes (linear/sinusoidal) and rates/times.

The *HOMEOFFSET* axis parameter may be set in user units by first defining the axis type using the *Type* axis parameter and entering the home offset into the *HomeOffsetInch* (or *HomeOffsetDeg* for rotary axes) machine parameter. This will overwrite the *HOMEOFFSET* axis parameter, setting it to the correct value for the desired offset in machine steps.

EXAMPLE:

```
PARMSET A DRIVE 1          ; enable the drive!
MHOME -1 2000              ; home direction is counter-clockwise,
                           ; move at 2,000 steps per second
```

4.6.39. MINCREMENTAL (distance) (speed)

This command starts the currently active axis moving the specified incremental distance at the specified speed. This command will abort any move in progress and begin the new move from the current position. A move currently executing will immediately take on the new velocity. The direction is specified as –1 or 1 for counterclockwise or clockwise motion. The speed is specified in machine counts per second. The axis will accelerate and decelerate at the currently selected mode (linear/sinusoidal) and rate/time. The specified drive must be enabled and the axis must not be in the sync mode or a programming error will occur.

EXAMPLE:

```
PARMSET A DRIVE 1          ; enable the drive!
MINCREMENTAL -10000 2000   ; move to -10,000 machine steps, at
                           ; 2,000 steps per second
```

4.6.40. MINFEEDSLAVE (distance) (speed)

This command starts the currently selected axis moving the specified distance at the specified velocity. A move currently executing will have its move increment summed with the distance specified in this function. The velocity is specified in machine steps per second. The axis will accelerate and decelerate in the currently selected modes (linear/sinusoidal) and rates/times. The specified drive must be enabled and the axis must be in the sync mode or a programming error occurs. This function should be used only on axes that are in sync or cam table mode. The motion generated by this command will be added to the motion output by the cam table.

EXAMPLE:

```
MINFEEDSLAVE 50000 10000    ;move the current axis 50,000 at 10,000
                            ;steps per second
```

4.6.41. MNOLIMITHOME (direction) (speed)

This command homes the specified axis in the direction and velocity specified. The home procedure is defined as follows: the axis begins moving in the specified home direction and stops on the next encoder marker pulse (or resolver null) and sets the position register equal to the *homeoffset* parameter. The velocity is specified in machine steps per second. The axis accelerates and decelerates at the currently selected modes (linear/sinusoidal) and rates/times. The specified drive must be enabled and the axis must not be in the sync mode or a programming error occurs. The direction is specified +/- by entering 1/-1 respectively for the direction parameter.

EXAMPLE:

```
MNOLIMITHOME 0 2000         ; home CCW at 2,000 steps per second
```

4.6.42. MOSCILLATE (distance) (speed)

This command continuously cycles the specified distance at the specified velocity. The sign of the distance parameter determines the initial direction of the cycle. The drive must be enabled. The *MABORT* command stops this command. The distance and velocity are in machine steps. The axis accelerates and decelerates at the currently selected modes (linear/sinusoidal) and rates/times.

EXAMPLE:

```
MOSCILLATE 25400 12700      ; cycle 25,400 steps at 12,700 steps per
                            ; second
```

4.6.43. MQABSOLUTE (position) (velocity)

This command is a queued version of the *MABSOLUTE* function. There is a 16 level queue for each axis. If a move is in progress, the queued move will not begin to execute until the current move is completed. If the queue is empty the move will begin immediately. The currently selected axis moves to the specified absolute position at the specified velocity. The velocity is specified in machine steps per second. The axis accelerates and decelerates at the currently selected modes (linear/sinusoidal) and rates/times. The specified drive must be enabled and the axis must not be in the sync mode or a programming error occurs.

EXAMPLE:

```
MQABSOLUTE 1000 10000      ; move to 1,000 at 10,000 steps per second
MQABSOLUTE 2000 10000      ; move to 2,000 at 10,000 steps per second
MQABSOLUTE 3000 10000      ; move to 3,000 at 10,000 steps per second
```

4.6.44. MQFLUSH

This command clears the 16 level deep axis queue for the currently selected axis. All commands that were in the queue will not be executed.

EXAMPLE:

```
MQFLUSH                    ; clear the queue for the current axis
```

4.6.45. MQHOLD

This command places the queue for the currently selected axis into the hold state upon completion of the current motion command. The current command will be completed and the queue can be restarted by using the *MQRELEASE* command.

EXAMPLE:

```
MQHOLD                     ; halt the execution queue for this axis
```

4.6.46. MQINCREMENTAL (distance) (speed)

This command is a queued version of the *MQABSOLUTE* command. There is a 16 level queue for each axis. If a move is in progress, the queued move will not begin to execute until the current move is complete. If the queue is empty the move will begin immediately. The currently selected axis moves the specified incremental distance at the specified speed. The velocity is specified in machine steps per second. The axis accelerates and decelerates at the currently selected modes (linear/sinusoidal) and rates/times. The specified drive must be enabled and the axis must not be in the sync mode or a programming error occurs.

EXAMPLE:

```
MQINCREMENTAL 50000 10000  ; move 50,000 steps at 10,000 steps
                           ; per second
```

4.6.47. MQUICKHOME (direction) (speed)

This command starts the currently selected axis homing in the specified direction at the specified velocity. The quick home function does not complete a normal home cycle, but moves in the specified home direction and stops at the home limit. The velocity is specified in machine steps per second. The axis accelerates and decelerates at the currently selected modes (linear/sinusoidal) and rates/times. The specified drive must be enabled and the axis must not be in the sync mode or a programming error occurs. The direction is specified as 1/-1 to move CW/CCW (+/-).

EXAMPLE:

```
MQUICKHOME -1 2000          ; quickhome CCW at 2,000 steps per second
```

4.6.48. MQRELEASE

This command restarts the specified axis queue that had been halted from executing motion commands contained within its 16 level deep queue by the *MQHOLD* command.

EXAMPLE:

```
MQRELEASE          ; restart the axis command queue
```

4.6.49. MRELEASE

This command resumes the motion that was in progress on the currently selected axis before calling the *MHOLD* command. This is done by accelerating the axis to the programmed velocity in the current mode (linear/sinusoidal, rate/time). The current mode is determined by the *accelmode* axis parameter. The current acceleration rate and time is determined by the *accelrate* and *accel* axis parameters respectively. This command has no effect on an axis that is not in the feedhold mode.

EXAMPLE:

```
MRELEASE
```

4.6.50. MB (address)

The *MB* command monitors the value of a byte at the specified address.

EXAMPLE:

```
MB 80c                    ; monitor a byte of data at 80c
```

4.6.51. MEM

The *MEM* command selects the memory command prompt mode that is used for executing memory commands.

EXAMPLE:

```
MEM                       ; switch to memory mode
```

4.6.52. ML (address)

The *ML* command monitors the value of a long word at the specified address.

EXAMPLE:

```
ML 80c          ; monitor a Long word of data at 80c
```

4.6.53. MW (address)

The *MW* command monitors the value of a word at the specified address.

EXAMPLE:

```
MW 80c          ; monitor a Word of data at 80c
```

4.6.54. OUTON (filespec)

The *OUTON* command echoes all output sent to the data screen to the specified file. This includes the prompt, commands typed, bad commands, and the terminating *OUTOFF* command. When executing the *OUTON* command, the prompt will change from:

AX1>

to

AX1 ['filename']>

indicating the filename within brackets between the current prompt (*AX#* or *MEM*) and the >.

EXAMPLE:

```
OUTON myfile.dat        ; echo output to myfile.dat
```

4.6.55. OUTOFF [Z]

The *OUTOFF* command will terminate echoing of the output to the data screen to the specified file with the *OUTON* command. An optional Z parameter may be specified to suspend echoing output to the file. After pausing the output to the file, it may be reactivated by using the *OUTON* command without specifying an output file.

EXAMPLE:

```
OUTOFF          ; disable output echoing
OUTOFF Z        ; pause output echoing
```

4.6.56. OUTPAUSE

The *OUTPAUSE* command temporarily suspends echoing output to the file. After pausing the output to the file it may be reactivated by using the *OUTPAUSE* command to toggle echoing back to the output file.

EXAMPLE:

```
OUTPAUSE        ; toggle state of echoing to output file
```

4.6.57. PARMGET (type) (parameter_name)

The *PARMGET* command returns the value of the specified parameter of the specified type. The types of parameters are Global, Task, Machine, and Axis, which are specified as G, T, M, and A respectively. If a parameter name is not specified, all parameters and their values will be displayed.

EXAMPLE:

```
PARMG A POS          ; display value of current axis position
PARMG G              ; display values of all Global parameters
```

For a complete list and description of parameters, refer to Appendix C: Parameters.

4.6.58. PARMMON (type) (parameter_name)

The *PARMMON* command monitors the value of a parameter, updating the display approximately every 100 msec. (10 times /second). The types of parameters are Global, Task, Machine, and Axis, which are specified as G, T, M, and A, respectively.

The parameter names of each type are shown in Table 4-2 under the *PARMGET* command. For a comprehensive description of each parameter refer to the *UNIDEX 600 Series Library Reference Manual, P/N EDU156* or the *U600 MMI online help file.*

EXAMPLE:

```
PARMMON A POS            ; monitor the position of the current axis
```

4.6.59. PARMSET (type) (parameter_name) (value)

The *PARMSET* command sets the value of the specified parameter of the specified type. The types of parameters are Global, Task, Machine and Axis, which are specified as G, T, M and A respectively. The parameter names of each type are shown in Table 4-2 under the *PARMGET* command. For a comprehensive description of each parameter refer to the *UNIDEX 600 Series Library Reference Manual, P/N EDU156 or the U600 MMI online help file.*

EXAMPLE:

```
PARMSET A KP 10      ; set current axis Kp gain to 10
```

4.6.60. PLAY (filespec) [S] [P]

The *PLAY* command executes the AerDebug commands contained within a text file. Blank lines within the file will be ignored. More *PLAY* commands may be contained within the specified text file (nesting is permitted). The optional second parameter 'S', allows the play file to be executed in the single step mode. In the single step mode each command within the play file will be executed following any key press, except the ESC key. Also, in the single step mode, the play file execution may be terminated by pressing the ESC key. The optional third parameter 'P', will disable the use of the ESC key to abort the play file execution.

EXAMPLE:

```
PLAY setup.ply S     ; execute the commands in the file
                     ; setup.ply in the single step mode
```

4.6.61. PLYREWIND

The *PLYREWIND* command rewinds or restarts the play file to the beginning. The next command executed would be the first command within the play file. This allows a play file to be executed continuously in a loop until the ESC key is pressed.

Example of a play file:

```
PARMSET A AUX 1          ; set auxiliary (mode) output to 1
PARMSET A DRIVE 1        ; enable drive
PARMSET A AUX 0          ; set auxiliary (mode) output to 0
PARMSET A DRIVE 0        ; disable drive
PLYREWIND                ; repeat till ESC key pressed
```

4.6.62. PRG1 ("command string")

The *PRG1* command compiles a single CNC program line. The CNC program line must be within quotes.

EXAMPLE:

```
PRG1 "BIND X"                ; bind the X axis
```

4.6.63. PRGCMPL (filespec) (option)

The *PRGCMPL* command compiles a CNC program. This function *does not* download the compiled file to the axis processor. The compiled program is held in RAM and can be downloaded to the axis processor via the *PRGLOAD* command. The filespec specifies the CNC program (and path, if its not in the current directory) to compile. The option flag is a bitmask of the following options:

Pre-process the file only	1
Do not read/write object files from/to disk	2
Read the object file from the disk	4
Create a listing file	8

Preprocessing produces only a listing that can be viewed with a *PRGTYPE* command, no object code will be generated.

The option flag being a bitmask implies that multiple options may be specified by adding the decimal values of the desired option flags together to produce the value of the option parameter that will produce these options. For example, if it were desired to create a listing file and not to write the object file to the disk, the option parameter would be set 10(8+2=10).

EXAMPLE:

```
PRGCPMPL \U600\TEST.PGM 10   ; compile test.pgm with options
                             ; described in text above
```

4.6.64. PRGDUMP (filespec)

The *PRGDUMP* command displays the source code of a CNC program as it currently exists on the axis processor card. Contrast this with the *PRGTYPE* command that shows the CNC program as it exists on the PC processor. *PRGTYPE* shows the original source code while *PRGDUMP* shows a textual decompiling of the current object code on the axis processor. The user needs only to compile to obtain a *PRGTYPE* listing, but must compile and download to obtain a *PRGDUMP* listing.

If a successful download was done just after the last compile of a program, then source code on the axis processor should look virtually identical to the original source code (on the PC), although there may be minor differences in spacing and grouping of terms in expressions.

EXAMPLE:

```
PRGDUMP \U600\TEST.PGM      ; display object code dump of test.pgm
```

4.6.65. PRGERRS (filespec)

The *PRGERRS* command displays any compile errors/warnings found when compiling the CNC program.

EXAMPLE:

```
PRGERRS \U600\TEST.PGM              ; any error's?
```

4.6.66. PRGINFO (filespec)

The *PRGINFO* command returns information on a program that has been downloaded to the axis processor card. It will display the number of program lines, double variables, labels, and the program status.

EXAMPLE:

```
PRGINFO \U600\TEST.PGM              ; tell me about my program!
```

4.6.67. PRGLOAD (filespec) [num_lines] [userline_offset]

The *PRGLOAD* command downloads a compiled program to the axis processor. If no *num_lines* are provided, then the program is download normally. However, if the number of lines parameter is provided, then the compiled program is downloaded as a circular queue. If *num_lines* parameter is non-zero, the axis processor allocates a new queue of *num_lines* size and downloads the program as the first lines in the queue. However, if *num_lines* parameter is zero, the axis processor appends the program onto an existing program queue. In either case the program lines will be placed down with user line numbers starting at the *userline_offset* parameter. For an example of how *PRGLOAD* works with queues, see the section on queues, under the Running CNC Programs Chapter.

EXAMPLE:

```
PRGLOAD \U600\TEST.PGM      ;download test.pgm to axis processor
```

4.6.68. PRGRUN (filespec) [line_number]

The *PRGRUN* command executes a CNC program that has been downloaded to the axis processor by the *PRGLOAD* command. The line number parameter may be optionally specified to begin execution of the program on a line other than line number one.

EXAMPLE:

```
PRGRUN \U600\TEST.PGM 5          ; run test.pgm beginning on line 5
```

4.6.69. PRGSTATS (filespec)

The *PRGSTATS* command returns the compiler status of the specified CNC program. This status includes errors, warnings, and compile time.

EXAMPLE:

```
PRGSTATS \U600\TEST.PGM          ; has program been compiled?
```

4.6.70. PRGTYPE (filespec)

The *PRGTYPE* command displays the CNC program lines from the specified program of the program that has been successfully compiled. This is not the downloaded object program on the axis processor (see *PRGDUMP*), it is the source lines from the user's program.

EXAMPLE:

```
PRGTYPE \U600\TEST.PGM      ;display CNC program lines
```

4.6.71. PRGUNLOAD (filespec)

The *PRGUNLOAD* command removes the specified CNC program from the axis processor's memory freeing the space for other programs.

EXAMPLE:

```
PRGUNLOAD \U600\TEST.PGM          ; unload program form memory
```

4.6.72. PSODOWNLOAD

The *PSODOWNLOAD* command loads the PSO firmware into the PSO card. The name of the PSO image will be specified in the AerReg program. If the base address of the PSO card (specified in AerReg) is non-zero, then this is automatically done when the axis processor firmware is downloaded. Normally, this does not need to be done manually.

EXAMPLE:

```
PSODOWNLOAD                  ; Load firmware
```

4.6.73. QUIT

The *QUIT* command terminates the AerDebug.exe application, as does the *EXIT* command.

4.6.74. RB (address)

The *RB* command reads the value of a byte at the specified address.

EXAMPLE:

```
RB 80c                    ; read a byte of data at 80c
```

4.6.75. RDO

This command executes the *RESET* command and the *DOWNLOAD* command sequentially.

EXAMPLE:

```
RDO
```

4.6.76. RESET

The *RESET* command resets the axis processor card to its power up default state. After a reset, no communication with the axis processor is possible until a *DOWNLOAD* command is executed.

EXAMPLE:

```
RESET                     ; reset the axis processor card
```

4.6.77. RGINFO [device_id] [card_num]

The *RGINFO* command displays the UNIDEX 600 Series controller information contained within the operating systems registry.

EXAMPLE:

```
RGINFO                    ; display registry information
```

4.6.78. RL (address)

The *RL* command reads the value of a long word at the specified address.

EXAMPLE:

```
RL 80c                    ; read a long word of data at 80c
```

4.6.79. RW (address)

The *RW* command reads the value of a word at the specified address.

EXAMPLE:

```
RW 80c                    ; read a word of data at 80c
```

4.6.80. SPENDANTTEXT (channel) (line #) (text)

The *SPENDANTTEXT* command sets the text line in the teach pendant. See AerPendantSetTextxxx function in the *UNIDEX 600 Library Reference Manual, P/N EDU156.*

EXAMPLE:

```
SPENDANTTEXT                ; Set text line.
```

4.6.81. TK (task_number)

The *TK* command is used to display or change the current task that the AerDebug commands will act upon. Entering the *TK* command without a task number will display the current task number that the command will act upon. Specifying a task number following the *TK* command will instruct AerDebug to direct all further commands to the specified task. The range of valid tasks are 1 through 4.

EXAMPLE:

```
TK                          ; displays current task mode
TK 2                        ; sets current task to 2
```

4.6.82. TSKASSOC (filespec)

The *TSKASSOC* command associates the specified CNC program with the current task.

EXAMPLE:

```
TSKASSOC \U600\TEST.PGM     ; associate test.pgm with the current
                            ; task
```

4.6.83. TSKDEASSOC

The *TSKDEASSOC* command deassociates the specified CNC program from the current task.

EXAMPLE:

```
TSKDEASSOC          ; deassociate currently associated program
                    ; from current task
```

4.6.84. TSKINFO

The *TSKINFO* command displays the task information for the current task. This information includes status information indicating if a program has been associated with the task, modes of the task (english/metric, absolute/incremental, etc.), faults, current CNC line number, and priority level.

EXAMPLE:

```
TSKINFO                     ; display current task info
```

4.6.85. TSKPRG (mode) (execution_type or line_number)

The *TSKPRG* command allows program execution to be controlled. The program may be Executed, Aborted, Reset, Stopped or have the Line number to execute set. These modes are set by specifying the mode parameter as E, A, R, S, or L, respectively. The second parameter is only specified for the Execute and Line number modes. The Execute mode may optionally specify an execution mode as normal - 0 (default), step into - 1 or step over - 2. The Line number mode may specify the line number to set by the second parameter.

EXAMPLE:

```
TSKPRG E          ; execute program in normal mode
TSKPRG E 1        ; execute program in step into mode
TSKPRG A          ; abort current program
TSKPRG R          ; reset current program
TSKPRG S          ; stop the current program
TSKPRG L 5        ; set current program line number to 5
```

4.6.86. TSKRESET

This command resets the current task. This includes deassociating any program, clearing task faults, clearing program call stack, and Rtheta, Preset, and fixture offsets are removed.

4.6.87. VAGET (type) (number)

The *VAGET* command displays the specified Global or Task axis point variable. The type of axis point variable is specified as G or T for Global or Task variables. By default, there are 10 of each axis point variables in the range of 0 through 9, determined by the *NumTaskAxisPts* and the *NumGlobalAxisPts* parameters. If the variable number is omitted, all ten of the specified type of axis point variables will be displayed.

EXAMPLE:

```
VAGET G 1         ; display Global axis point variable 1
VAGET T           ; display all Task axis point variables
```

4.6.88. VCGET

The *VCGET* command displays the variables for the task subroutine call stack.

EXAMPLE:

```
VCGET             ; display the variable for the subroutine
                  ; call stack
```

4.6.89. VDGET (type) (number)

The *VDGET* command displays the value of the specified double variable. There are four types, Global, Task, Program, and call Stack variables represented by G, T, P and S respectively, all of which will hold floating point numbers in the range of 1.7E-308 to 1.7E+308. By the default value of the *NumGlobalDoubles* and the *NumTaskDoubles* parameters, there are ten of each Global and Task variables, each numbered 0 through 9. There are four Program variables, numbered 0 through 3. There are twenty-six call Stack parameter variables, numbered 0 through 25. If no variable number is specified, all variables of that type will be displayed.

EXAMPLE:

```
VDGET G 0          ; display Global double variable 0
VDGET T 9          ; display Task double variable 9
VDGET P            ; display all Program double variables
VDGET S 1          ; display call Stack double variable 1
```

4.6.90. VDMON (type) (number)

The *VDMON* command monitors the value of a double variable, updating the display approximately every 100 msec. (10 times /second). The type of double variables are Global, Task, Program and call Stack, which are specified as G, T, P, and S, respectively, all of which will hold floating point numbers in the range of 1.7E-308 to 1.7E+308. There are ten of each Global and Task variables, each numbered 0 through 9, determined by the default settings of the *NumGlobalDoubles* and the *NumTaskDoubles* parameters. There are four of the Program variables, each numbered 0 through 3. There are twenty-six of the call Stack double variables numbered 0 through 25.

EXAMPLE:

```
VDMON G 9          ; monitor Global double variable 9
VDMON T 0          ; monitor Task double variable 0
VDMON P 3          ; monitor Program double variable 3
VDMON S  25        ; monitor call Stack parameter double
                   ; variable 25
```

4.6.91. VDSET (type) (number) (value)

The *VDSET* command sets the value of the specified double variable. There are four types, Global, Task Program and call Stack variables represented by G, T, P and S respectively, all of which will hold floating point numbers in the range of 1.7E-308 to 1.7E+308. There are ten of each Global and Task variables, each numbered 0 through 9, determined by the *NumGlobalDoubles* and the *NumTaskDoubles* parameters. There are four Program variables, numbered 0 through 3. There are twenty-six call Stack parameter variables, numbered 0 through 25. If no variable number is specified, all variables of that type will be displayed.

EXAMPLE:

```
VDSET G 0 123.456       ; set Global double variable 0 to 123.456
VDSET T 9 56.7          ; set Task double variable 9 to 56.7
VDSET P 3 19.63         ; set Program double variable 3 to 19.63
VDSET S 1 32.45         ; set call Stack double variable 1 to
                        ; 32.45
```

4.6.92. VSGET (type) (number)

The *VSGET* command displays the specified string variable. There are three types, Global, Task, and Program string variables represented by G, T and, P respectively, all holding 128 character strings. There are ten Global, twenty Task string variables, each numbered 0 through the (maximum size -1), determined by the *NumGlobalStrings* and *NumTaskStrings* parameters. The number of program string variables is determined by the number defined within the current CNC program associated with the current task. If no variable number is specified, all variables of that type will be displayed.

EXAMPLE:

```
VSGET G 9               ; display Global string variable 9
VSGET T 19              ; display Task string variable 19
VSGET P 0               ; display Program string variable 0
```

4.6.93. VSMON (type) (number)

The *VSMON* command monitors the value of a string variable, updating the display approximately every 100 msec. (10 times /second). There are three types, Global, Task and Program string variables represented by G, T and P respectively, all holding 128 character strings. By default, there are ten Global, twenty Task string variables, each numbered 0 through the(maximum size -1), determined by the *NumGlobalStrings* and *NumTaskStrings* parameters. The number of Program string variables is determined by the number defined within the current CNC program associated with the current task.

EXAMPLE:

```
VSMON G 9               ; monitor Global string variable 9
VSMON T 19              ; monitor Task string variable 19
VSMON P 0               ; monitor Program string variable 0
```

4.6.94. VSSET (type) (number) ("string")

The *VSSET* command sets the specified string variable to the text string specified. There are three types, Global, Task, and Program string variables represented by G, T, and P, respectively, all holding 128 character strings. By default, there are ten Global, twenty Task string variables, each numbered 0 through the (maximum size -1), determined by the *NumGlobalStrings* and *NumTaskStrings* parameters. The number of Program string variables is determined by the number defined within the current CNC program associated with the current task.

EXAMPLE:

```
VSGET G 9          ; display Global string variable 9
VSGET T 19         ; display Task string variable 19
VSGET P 0          ; display Program string variable 0
```

The string must be within quotation marks if there are spaces within the string.

4.6.95. WAIT ([!] condition)

The *WAIT* command waits on any of the following conditions by specifying the string shown below for the desired condition. Also, the user may wait for the condition to become false by specifying a "!" or a "~" before the string. The case of the string condition parameter is insignificant.

ProgramAssociated
ProgramActive
ProgramExecuting
ImmediateCodeExecuting
SingleStepInto
SingleStepOver
InterruptFaultPending
InterruptCallBackPending
ProgramCleanup
EStopInputActive
FeedHoldInputActive
SpindleFeedHoldActive
MotionFeedHoldActive
MotionContinuous

EXAMPLES:

```
WAIT   ProgramAssociated     ; wait till a program has been associated
                             ; with this task
WAIT ! ProgramExecuting      ; wait till the program is done executing
```

4.6.96. WB (address) (value)

The *WB* command writes the value-specified byte to the specified address.

EXAMPLE:

```
WB 80c 55          ; write 55 to 80c
```

4.6.97. WRITESERIAL (channel) (text)

The *WRITESERIAL* command writes same text to a serial port. See the AerSerialWritexxx function in the *UNIDEX 600 Library Reference Manual, P/N EDU156.*

EXAMPLE:

```
WRITESERIAL  0     ; "test" write to serial port
```

4.6.98. WW (address) (value)

The *WW* command writes the value-specified word to the specified address.

EXAMPLE:

```
WW 80c 55AA                    ; write AA55 to 80c
```

4.6.99. WL (address) (value)

The *WL* command writes the value-specified long word to the specified address.

EXAMPLE:

```
WL 80c 123455AA          ; write 1234AA55 to 80c
```

4.6.100. ZMONITOR

The *ZMONITOR* command displays all the monitor conditions defined within the current CNC program associated with the current task. The maximum number of conditions permitted is determined by the *MaxMonitorData* task parameter.

EXAMPLE:

```
ZMONITOR        ; display all the current monitor
                ; conditions defined
```

4.6.101. ZONGOSUB

The *ZONGOSUB* command displays all the ongosub conditions defined within the current CNC program associated with the current task. The maximum number of conditions permitted is determined by the *MaxOnGosubData* task parameter.

EXAMPLE:

```
ZONGOSUB        ; display all the current ongosub
                ; conditions defined
```

4.7. Command to Library Cross Reference

This section contains the command to library function cross-references, refer Table 4-3, Table 4-4, and Table 4-5 for all cross-references.

Table 4-3. Basic Command to Library Function Cross Reference

Basic Commands	Description	Library Function
?	Display help (List of commands).	Not Applicable
!TI	Test the PC interrupt.	Not Applicable
AX	Changes the default axis (if no parameter shows the default axis).	Not Applicable
CMDERR	Display the last command error.	AerDrvGetLastCmdErr
CMDLAST	Retrieve the last command from the command line buffer.	AerDrvGetLastCmd
DOWNLOAD	Load firmware into axis card.	AerSysDownLoad
DRVINFO	Displays information on the device driver.	AerDrvGetDebugInfo
EXIT	Quits the AerDebug application.	Not Applicable
MEM	Selects MEMORY command prompt mode.	Not Applicable
OUTON	Sets output to be written to the specified to file.	Not Applicable
OUTOFF	Disables output from being written to a file.	Not Applicable
OUTPAUSE	Suppresses/unsupresses output dumping to file (toggle).	Not Applicable
PLAY	Executes the commands from within the specified file.	Not Applicable
PLYREWIND	Rewinds the PLAY file to the start.	Not Applicable
QUIT	Exits the AerDebug application.	Not Applicable
RESET	RESET the Axis Processor card.	AerSysReset
RGINFO	Displays information on the Operating System registry.	AerRegGetDeviceInfo
TK	Changes the default task (if no parameter shows the default task).	Not Applicable

Table 4-4. Axis Command to Library Function Cross Reference

Axis Commands	Description	Library Function
CONFIGD2A	Configure a DAC channel for this axis.	AerConfig
CONFIGENCODER	Configure an encoder feedback channel for this axis.	AerConfigEncoder
CONFIGHENCODER	Configure an encoder feedback channel with hall effect sensors for this axis.	AerConfigEncoderHall
CONFIGRESOLVER	Configure a resolver feedback channel for this axis.	AerConfigResolver
CONFIGHRESOLVER	Configure a resolver feedback channel with hall effect sensors for this axis.	AerConfigResolverHall
CONFIGREAD	Configure an axis from an .INI file.	AerConfigReadPacket AerConfig
CONFIGWRITE	Write the axis configuration to an .INI file.	AerConfigGet AerConfigWritePacket
DCAX	Axis data center data.	AerDCGetAxisDirect
GETPROG	Display programming error.	AerProgGet
INFO	Axis configuration information.	AerConfigGet

Table 4-5. Task Command to Library Function Cross Reference

Task Commands	Description	Library Function
DIR	Directory of downloaded programs.	AerProgramGetHandle
DUMPTABLE	Dumps CAM tables.	AerCamTable
DUMPERROR	Dumps error calibration table.	AerAxisCal
ENABLEPENDANT	Enables the pendant.	AerPendantSetMode
EXELINE	Execute a single CNC program line.	AerCompilerDestroy AerCompilerCreate AerCompilerCompileLine AerCompilerRunImmediate AerCompilerDestroy
EXEPRG	Compiles, loads, associates and executes a program.	AerCompilerDestroy AerCompilerCreate AerCompilerCompileProg AerCompilerDownload AerTaskProgramAssociate AerTaskProgramExecute
IOGET	Display the value of a virtual I/O point ([BI]/BO/RI/RO).	AerVirtGetBinaryInput AerVirtGetBinaryOuput AerVirtGetRegisterInput AerVirtGetRegisterOutput
IOMON	Monitor the value of a virtual I/O point ([BI]/BO/RI/RO).	AerVirtGetBinaryInput AerVirtGetBinaryOuput AerVirtGetRegisterInput AerVirtGetRegisterOutput
IOSET	Set the value of a virtual I/O point ([BI]/BO/RI/RO).	AerVirtSetBinaryInput AerVirtSetBinaryOuput AerVirtSetRegisterInput AerVirtSetRegisterOutput
PARMGET	Display the value of a parameter (G/T/M/A).	AerParmGlobalGetValue AerParmTaskGetValue AerParmMachineGetValue AerParmAxisGetValue
PARMMON	Monitor the value of a parameter (G/T/M/A).	AerParmGlobalGetValue AerParmTaskGetValue AerParmMachineGetValue AerParmAxisGetValue
PARMSET	Set the value of a parameter (G/T/M/A).	AerParmGlobalSetValue AerParmTaskSetValue AerParmMachineSetValue AerParmAxisSetValue
PRG1	Compile a single CNC program line.	AerCompilerCreate AerCompilerCompileLine
PRGCMPL	Compile a CNC program.	AerCompilerCreate AerCompilerCompileProg
PRGDUMP	Dump program machine code lines.	AerProgramGetInfo AerProgramGetLabel AerProgramGetLine
PRGERRS	Shows program compile errors.	AerCompilerErrsGetNumOf AerCompilerErrGetData AerCompilerErrGetText
PRGINFO	Displays program information.	AerProgramGetInfo
PRGLOAD	Loads a CNC program into the Axis Processor.	AerCompilerDownload
PRGRUN	Runs a CNC program.	AerTaskProgramSetLineUser AerTaskProgramExecute

Table 4-5. Task Command to Library Function Cross Reference (cont'd)

Task Commands	Description	Library Function
PRGSTATS	Shows program compile status.	AerCompilerGetStatus
PRGTYPE	Shows program test lines.	AerCompilerGetLineText
PRGUNLOAD	Frees a CNC program.	AerProgramFree
PSODOWNLOAD	Loads the PSO firmware	
SPENDANTTEXT	Sets the text line in the teach pendant.	AerPendantSetText
TSKASSOC	Associates a CNC program with the current task.	AerTaskProgramAssociate
TSKDEASSOC	Deassociates a CNC program from the current task.	AerTaskProgramDeAssociate
TSKINFO	Show information on current task.	AerDCGetTaskDirect
TSKPRG	Control program execution A/[E]/R/S/L.	AerTaskProgramExecute AerTaskProgramAbort AerTaskProgramReset AerTaskProgramStop AerTaskProgramSetLineUser
TSKRESET	Resets the current task.	AerTaskReset
VDGET	Display the value of a variable (double word) [G]/T/P/S.	AerVarGlobalGetDouble AerVarTaskGetDouble AerVarProgramGetDouble AerVarStackGetDouble
VDMON	Monitor a variable (double word) [G]/T/P/S.	AerVarGlobalGetDouble AerVarTaskGetDouble AerVarProgramGetDouble AerVarStackGetDouble
VDSET	Set a variable (double word) [G]/T/P/S.	AerVarGlobalSetDouble AerVarTaskSetDouble AerVarProgramSetDouble AerVarStackSetDouble
VSGET	Display a string variable [G]/T/P.	AerVarGlobalGetString AerVarTaskGetString AerVarProgramGetString
VSMON	Monitor a string variable [G]/T/P.	AerVarGlobalGetString AerVarTaskGetString AerVarProgramGetString
VSSET	Set a string variable [G]/T/P.	AerVarGlobalSetString AerVarTaskSetString AerVarProgramSetString
WAIT	Wait on status.	AerDCGetTaskDirect
WRITESERIAL	Writes same text to a serial port	AerSerialWrite
ZMONITOR	Display monitor data.	AerTaskMonitorGetData
ZONGOSUB	Display ongosub data.	AerTaskOnGosubGetData

∇ ∇ ∇

CHAPTER 5: AERTUNE

In This Section:

5.1. Introduction

The AerTune program is a utility for visually observing and fine-tuning the performance of the motion generated by a UNIDEX 600 Series controller. The utility allows multiple windows to be displayed, providing a separate window for each axis. Each window allows a single step or continuous cycle command to be generated for an axis to simulate a typical move profile for the users desired application. This will allow the response of the axis to be graphically displayed and adjusted accordingly. All U600 Series controllers use the same PIDF servo loop with velocity and acceleration feedforward.

5.2. The Main Window of the AerTune Program

The menu selections for the main window will be described in the following sections.

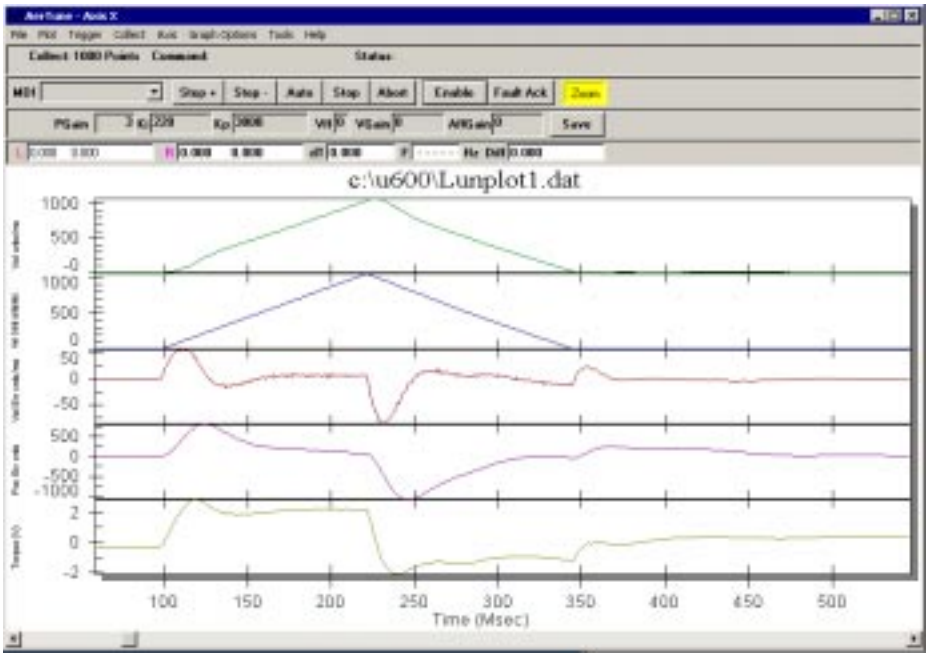


Figure 5-1. AerTune Main Window

5.2.1. The Help Menu

The Help menu displays the online help for AerTune.

5.2.2. The File Menu

The AerTune utility allows the user to observe and fine-tune the performance of the axes. This is accomplished by allowing the user to command motion and/or capture data from the axis. The AerPlot utility may be used to plat multiple axes simultaneously. See the DATASTART command for triggering data collection from a CNC program. This utility may not collect data while AerPlot, AerTune, AerPlot3D, or AerPlotIO are collecting data.

The File menu allows plots to be saved, loaded, and printed. The Plot Comment selection allows a comment to be displayed for the plot, below the axis name. This comment is not saved to the .dat file when the plot is saved, but will be visible when the plot is printed.

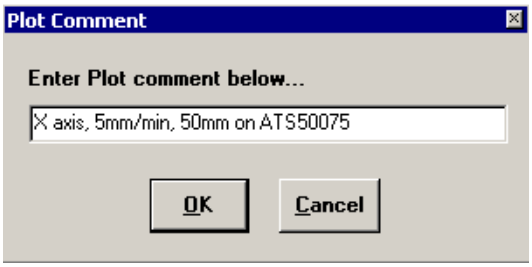


Figure 5-2. Plot Comment

5.2.3. The Plot Menu

The data that may be displayed from the AerTune's plot menu includes (* - indicates default selections):

* Velocity Feedback
* Velocity Command
* Velocity Error
Position Feedback
Position Command
* Position Error
* Torque
Acceleration

5.2.4. The Collect Menu

The Collect menu allows the number of samples to be defined, also determining the time period over which the data acquisition takes place. The choices are:

100, 250, 500, 1000, 5000, 8000

5.2.5. The Graph Options Menu

The Graph Options menu has four selections.

The Grid Lines selection allows a grid to be displayed on the X, Y, or both axes.

The Mark Data points selection allows dots to be displayed for each sample point.

The Units menu allows the units for linear, rotary, and analog inputs to be selected. The linear axes may be displayed as machine steps (counts), mm, mm/1000, or inches, inches/1000. Rotary axes may be displayed as machine steps (counts) or degrees.

The analog inputs may be displayed as machine steps (counts) or volts. The Time Scale selection allows the time or X axis of the plot to be displayed as seconds, seconds/1000 (milliseconds), or by sample number.

The Zoom menu allows the Zoom feature to be activated, disabled, (so that the cursor functions may be used) and to Un-Zoom. These features are available via the Zoom button also. To Un-Zoom using the Zoom button, click the right mouse button on the Zoom button.

5.2.6. The Trigger Menu

The Trigger menu allows the sample rate to be defined. The sample rate determines the number of milliseconds between each sample. This also determines the period of time that AerPlot will collect data for, based upon the number of points defined on the collect menu.

The Collect On Set of Data menu selection will collect the specified number of points and update the display.

The Collect Data Continuous menu selection will collect the specified number of points, update the screen, then repeat the cycle until halted by the user.

The Collect Halt Data Continuous menu selection will terminate the Continuous Data Collection mode.

5.2.7. The Axis Menu

The Axis menu allows one of the axes to be selected for display. The axes are listed by number in the same order they are listed on the position display or axes configuration page. To display multiple axes simultaneously, use the AerPlot utility.

5.2.8. The Tools Menu

The Tools menu allows the Status, Control, and Cursor toolbars to be displayed across the top of the plot, below the menu. Sec_Num_FFTAnalysis

The FFT Analysis menu selection - see Section 5.3.2.

The Hard Reset menu selection allows the controller to be reset to its power-up state. This selection should not be used.

The Fault Acknowledge selection will attempt to acknowledge and clear any faults that are present. The Fault Ack button, on the status tool bar, will also attempt to clear the faults.

The Auto Tune menu selection – see Section 5.4.

5.3. Using AerTune

Within AerTune there are various controls for exercising the axis. These include Step+ and Step– buttons, for moving the axis the incremental distance specified by the Step Move parameters. There is also an Auto button for initiating an auto cycle of the axis, which will cycle the axis back and forth by the amount in the distance entry field. The initial direction is determined by the button selected, either plus or minus. The axis may be stopped during a move by selecting the Halt button. A button is also provided for enabling/disabling the axis. The Clear button will clear any faults and their associated messages that may occur while exercising the axis.

The Step Move parameters allow the user to define a move profile typical to the desired application. The Distance and Speed are specified in machine steps and machine steps per second. All other Move parameters are the respective axis parameters for the axis. For additional information on the axis parameters, refer to Appendix C: Parameters.

The *ACCELMODE/DECELMODE* parameters define the acceleration and deceleration as Linear/1-Cosine) and Time/Rate based, as follows:

0 – (1-Cosine) – Time Based

1 – Linear Ramping – Time Based

2 – (1-Cosine) Ramping – Rate Based (Recommended)

3 – Linear Ramping – Rate Based

The *ACCEL/DECEL* entry fields specify the Acceleration/Deceleration time in milliseconds when Time based ramping is active. The units are milliseconds (i.e., a one second ramp would be specified as 1000 [milliseconds]). The *ACCELRATE/DECELRATE* entry fields specify the Acceleration/Deceleration rate in machine counts per second^2 when Rate based ramping is active.

The Gain parameter fields (all Axis parameters) allow the user to adjust the response of the servo loop. The *PGain* parameter varies the position loop gain of the servo loop and minimizes position errors during acceleration and deceleration while at rest. The *Kp*

parameter varies the damping effect of the servo loop. The *Ki* parameter varies the integral gain of the servo loop for increasing stiffness, while in position, and improves settling time. The *VGain* parameter minimizes the position error during constant velocity in analog tachometer based systems and torque mode systems with large frictional loads. The *Vff* parameter enables velocity feedforward for the axis minimizing position error during constant velocity. The *AffGain* parameter provides acceleration feedforward for the axis, which minimizes position error during acceleration and deceleration.

5.3.1. Step Move Parameters

The distance field is the number of machine counts that the axis will move when the Step+ or Step– (or auto, cycle +/-) buttons are pressed. Speed is the velocity in machine counts/second that the axis will move at. We recommend using sinusoidal rate based accel and decel modes.

The mode (rate/time) for G0/Asynchronous motion, generated by AerTune, is determined by each axes ACCELMODE/DECELMODE axis parameters, which also select linear/sinusoidal profiles. The appropriate axis parameter (shown below) will then determine the acceleration/deceleration of the axes.

Acceleration Axis Parameters	Deceleration Axis Parameters
ACCELMODE	DECELMODE
ACCEL (time)	DECEL (time)
ACCELRATE	DECELRATE

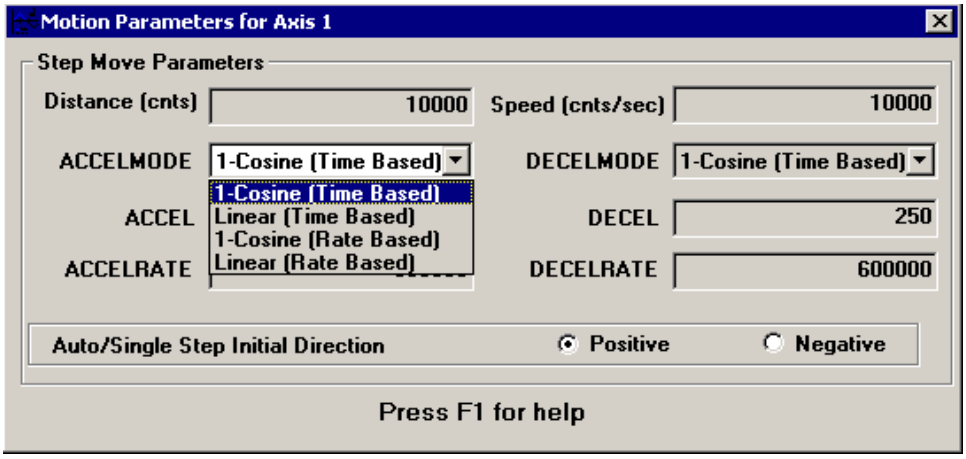


Figure 5-3. Step Move Parameters

5.3.2. FFT Analysis

Clicking the Analysis menu selection will graphically display the spectral frequency distribution of the item selected from the "Data to Analyze" menu. This will allow you to determine if there is a resonant frequency present in the servo loop and /or mechanics of the system. A peak, such as that shown in Figure 5-4, at approximately 400 Hz, is indicative of a resonance. In this case, a 375 Hz low-pass filter might be used. If there is a resonant frequency, see Identifying an Instability within the Servo Loop (Section 5.3.4.).

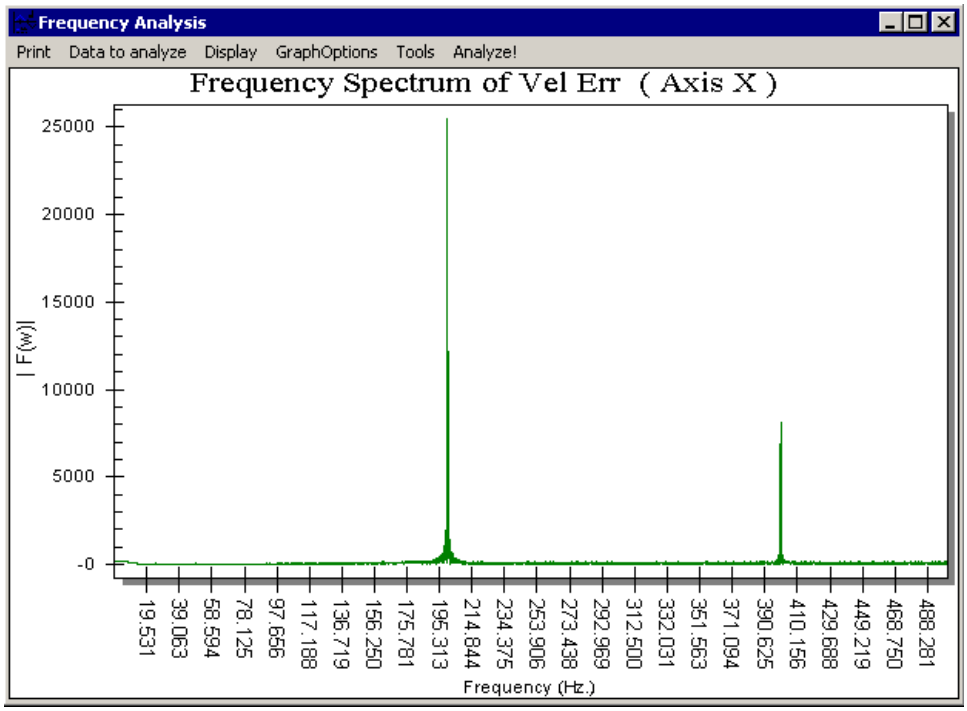


Figure 5-4. FFT Analysis

5.3.2.1. The FFT Analysis Window Menu Description

Print Menu — You may Export or Print the plot.

Data to Analyze — You may select the item to analyze:

Velocity Feedback, Velocity Command, Velocity Error, Position Feedback, Position Command, Position Error, or Torque

Display Menu — You may select the number of points to display on the plot:

64, 128, 256, 512, 1024, 2048, 4096, 8192

Tools Menu

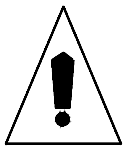
Disable Bias Correction
Low Pass Filter Data
Remove DC Bias from Position Data
Remove DC Bias from Torque Data

5.3.3. Determining the Maximum Acceleration of an Axis

The maximum acceleration of an axis may be calculated or determined empirically via the AerTune utility after your axis has been fully assembled (to other axes), configured, and tooling/parts loaded. The acceleration is increased until the axis produces a 10 volt torque command during acceleration. This indicates that the amplifier has saturated (full on), indicating that the axis is accelerating as fast as possible with the current load.

In the example below, we will determine the maximum acceleration in machine counts / second / second, since this can easily be converted to user units / second / second for the AccelRateIPS2 (AccelRateDPS2 for rotary axes) task parameter. You could also determine the maximum acceleration (in time) to a given velocity. However, this is dependent on the programmed speed.

Other system considerations like motors, ball-screws, bearings, encoders, stiffness, structural resonance, move profiles, and duty cycle may require a lesser value for optimum and reliable performance. Exceeding the maximum acceleration specifications of system components will cause damage.



1. From the "Plot" menu, select "Velocity Command", "Velocity Feedback", and "Torque" to be displayed.
2. Set the ACCELMODE and DECELMODE axis parameters for linear or sinusoidal rate-based acceleration. Rate-based sinusoidal acceleration is recommended.
3. Select "Update Step Move Parameters" from the "Trigger" menu.
4. Enter a speed and distance (in machine counts) to produce a trapezoidal velocity profile that will not encounter an end of travel limit. A trapezoidal velocity profile implies a profile at which the axis reaches constant velocity.
5. Press the Auto Cycle button.
6. Increase (or decrease) the ACCELRATE field on the "Update Step Move Parameters" window until the axis produces a 10 volt torque command during acceleration.
7. The value in the ACCELRATE field is the maximum rate at which the axis can accelerate with the current load.

5.3.4. Identifying an Instability within the Servo Loop

Using the AerTune utility:

1. Activate the "Cursor" toolbar from the "Tools" menu.
2. Select "100 points" from the "Display" menu.
3. Exercise (move) the axis in question.

You may now utilize the FFT Analysis (Section 5.3.2.) selection on the Tools menu of AerTune, or follow steps 4 and 5.

4. If a repeating sinusoidal waveform is now visible on the torque or velocity plots, click (left mouse button) on the peaks of two sequential sinusoids.
5. The frequency of the disturbance will be displayed within the "F" frequency box on the cursor toolbar.

Using the Aerotech Filter utility (Section 11.1):

6. Select the "Low Pass" radio button and enter a "Stop Freq" approximately 20 Hz below the frequency determined from step 5. For example, if step 5 identified a disturbance frequency of 170 Hz for the "Stop Freq".
7. Click the "Calculate Coeff." Button
8. The required values of the A1, A2, B0, B1, and B2 axis parameters will be displayed. Enter these values into the axis parameters for the axis in question.

You should now be able to successfully Auto Tune the axis.

5.3.5. Minimizing Position Error due to Torque Ripple or Amplifier Offsets

If the position error is a cyclic sinusoid repeating every electrical cycle of the motor, it is most likely due to offsets in the amplifier current loop, or possibly poor Hall effect alignment. One electrical cycle of the motor can be determined from Aerotech's motor specifications. Assuming that you do not have an inductive current probe, follow this procedure:

Cycle the axis through multiple electrical cycles while monitoring the position error, in AerTune:

1. Adjust the PHASEAOFFSET axis parameter for the axis +/- in increments of 50 till you minimize the position error as best you can.
2. Now do the same for the PHASEBOFFSET axis parameter for the same axis.
3. Repeat steps 1 and 2 with a +/- 25 increment of the parameters.
4. Repeat steps 1 and 2 with a +/- 10 increment of the parameters.
5. Repeat steps 1 and 2 with a +/- 5 increment of the parameters.

5.3.6. AC Brushless Motor Tuning Tip

Shown in Figure 5-5 is a tuning plot of an AC brushless motor. Note the ripple effect during the move. This is normal since AC brushless motors usually have a larger amount of torque ripple than DC brush motors. This torque ripple has been observed to be approximately 10 machine steps peak to peak with an unloaded BM Series motor. While tuning an axis driving a brushless motor, torque ripple can be identified as the cause of the disturbance by comparing one electrical cycle of the brushless motor.

One electrical cycle of the brushless motor can be calculated in machine steps by dividing the number of machine counts per revolution of the motor by the number of electrical cycles of the motor. A standard BM Series motor has a 1,000 line encoder mounted on the motor that the controller does times 4 multiplication on, producing 4,000 machine steps per revolution of the motor. Therefore, one electrical cycle of the motor is equal to:

4000 / (pole count) = machine steps per electrical cycle

See the online help file for the pole count of BM/BMS motors and the length on the electrical cycle of BLM linear motors.

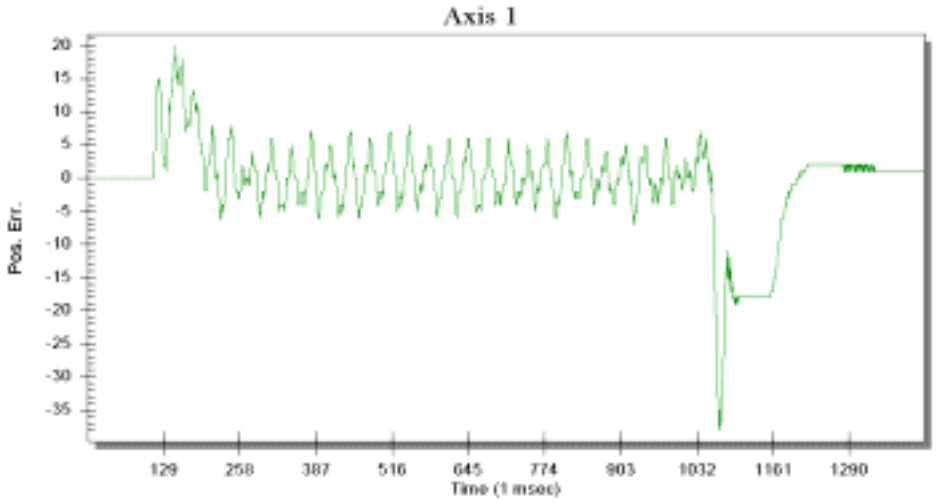


Figure 5-5. Torque Ripple Plot of an AC Brushless Motor

See Section 5.3.5. regarding minimizing torque ripple.

5.3.7. Computing Torque (Closed-Loop Torque Mode)

The torque applied to the motor (in torque mode) may be easily calculated for brush or brushless motors if you know a few parameter values and the K_t (motor torque constant). The UNIDEX 600/650 Controllers have a 16 bit Digital-to-Analog converter used to convert a signed 16 bit number (+32,767 through –32,767) to an analog voltage in the range of +10 volts through –10 volts. This voltage is applied to the command input of the servo amplifier (knowing the G_m (transconductance) value of the servo amplifier allows the current (Amps) output to be calculated). Aerotech's amplifiers typically (model dependant) have a peak output of 20 or 30 amps. Simply meaning, a +/- 10 volts in to the amplifier will be equal to +/- *x* amps, where *x* is the peak output current of the servo amplifier. This current produces a torque generated by the motor, equal to the current multiplied by the K_t of the motor. For example, if the peak output command (or instantaneous output command) from DAC was 16,385 (+5 volts), the amplifier produced 30 amps for an input of 10 volts, and the motor had a K_t of 16 oz-in (ounce-inches) per amp, then:

(5 volt command / 10 volt, max amps input command) * 30 amps (max I output) * 16 oz-in K_t = 240

or

5/10 * 30 *16 = 240

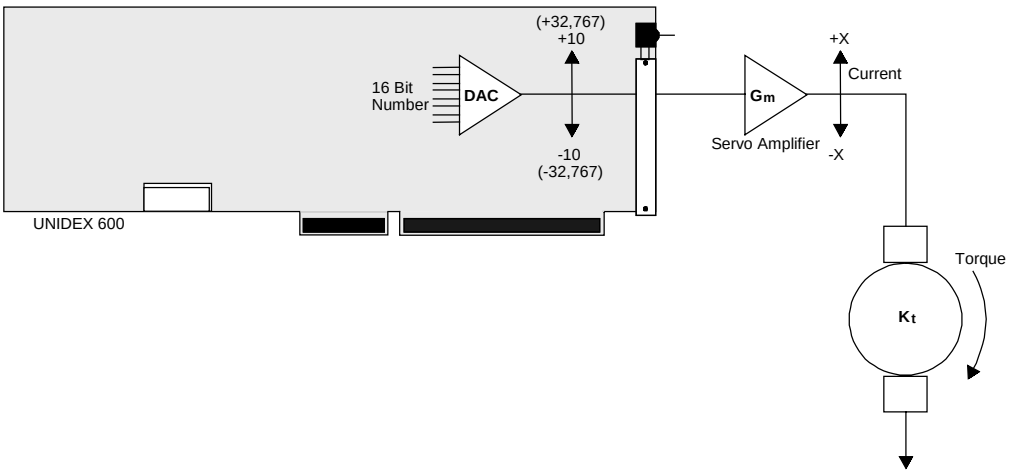


Figure 5-6. Closed-Loop Torque Mode

5.4. Servo Loop Auto Tuning

Slaved axes may be AutoTuned, but, the calculated gains will be entered into the master axis only. You must manually duplicate the gains into the slave axis parameters. A second AerTune window may be opened, and left open, to change the gains for the slave axis. The AutoTune feature of the U600 AerTune utility program provides the user with an automated servo loop tuning capability for torque (current) mode axes. This allows most users to quickly get their systems tuned and running quickly. The AutoTuning algorithm bases its calculations on user supplied tuning specifications and identified system parameters that are calculated during the AutoTuning cycle. When the AutoTuning cycle is complete, servo loop gains are returned that satisfy the given user requirements. An axis may be manually tuned by the procedure provided in Section 5.5.

The basic AutoTune cycle consists of defining the tuning parameters, starting the AutoTune cycle, and accepting the calculated gains.

Begin AutoTuning with a 10 Hz bandwidth and increase until the axis becomes unstable. After the axis becomes unstable, decrease the bandwidth by about 10%.

The AutoTune algorithm will be unable to calculate acceptable values for the servo loop gains if an instability exists, this may require manually tuning the axis, or, under severely unstable conditions, identifying and eliminating the instability.

In general, the default values are a good starting point for most typical systems (especially those using AC Brushless motors). One exception is that the Amplitude parameter should be set to produce at least +/- a half revolution of the motor (or +/- a half inch of linear travel), to achieve acceptable results from the AutoTuning algorithm, or, more accurately, a torque signal greater than 1 volt.

Air Bearing systems should use a .7-damping factor.
Ball Screw systems should use a .3-damping factor.

Most systems should be able to achieve about 30 Hz bandwidth.

Systems with a larger mass or high inertia may need to reduce the excitation frequency to .25 - .5 Hz. Smaller systems may need to increase this to 2 Hz.

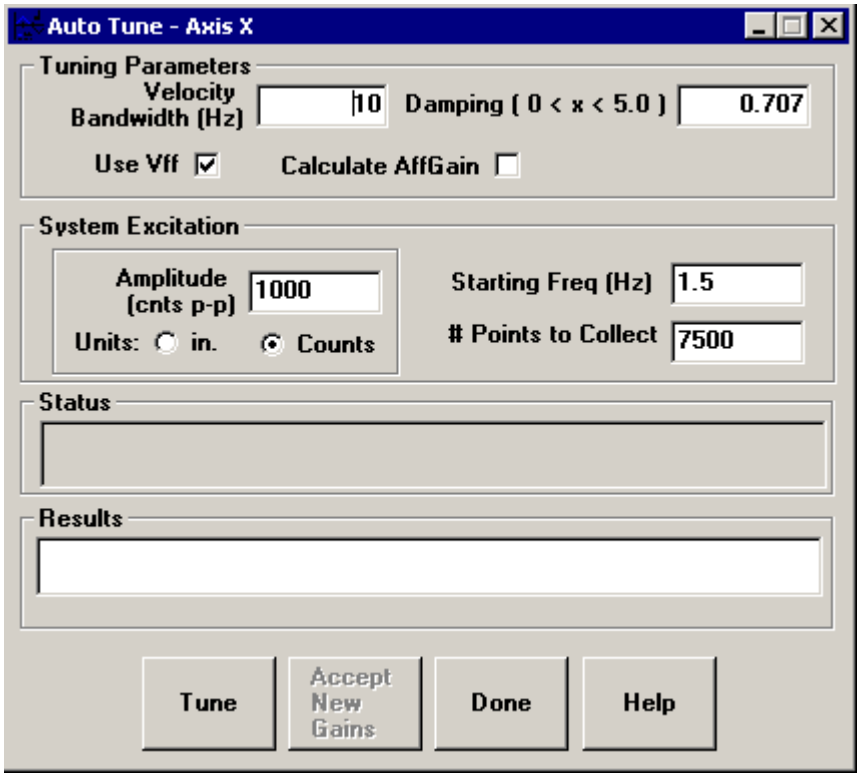


Figure 5-7. AutoTune Screen

5.4.1. Setting up AutoTune Parameters

Here are some general considerations for successful AutoTuning:

- The excitation parameters must generate approximately a 1 volt torque command.
- Axes with a large mass or high inertia require a lower excitation frequency of approximately 0.25 – 0.5 Hz.
- Axes with a small mass or low inertia require a higher excitation frequency of approximately 2.0 Hz.
- Air bearing axes should use a damping factor of .7
- Ball-screw driven axes should use a damping factor or .3
- Most axes should be able to achieve about 30 Hz bandwidth
- Begin at 10 Hz bandwidth, AutoTune, and then accept the calculated gains, increase the bandwidth, working up until each axis becomes unstable, then reduce the bandwidth to the previous stable value.

5.4.2. Excitation Parameters

In order to identify the necessary system parameters that the AutoTune algorithm needs (calculate the servo loop gains), a multiple frequency sinusoidal test input is used to excite the system. The following set up parameters are available:

5.4.2.1. Amplitude of Excitation in AutoTune

The Amplitude parameter sets the Peak-Peak envelope of the sinusoidal input. The system will move about its starting position by +/- 0.5 * Amplitude. The Amplitude can be displayed and entered either in raw machine counts or user units (inches/mm or degrees, depending on the axes selected). The units can be changed by clicking on the counts or user units button. Systems with high-resolution feedback devices will almost always require the amplitude to be raised so that the system will move an adequate amount and utilize a measurable amount of torque. In general, the AutoTune calculations will be more accurate the larger the amplitude is, chosen within the constraints of the system.

If the Velocity Trap (VELTRAP) axis parameter is set to a non-zero value, AutoTune will prevent System Excitation Parameters from being chosen that will violate the Velocity Trap value.

5.4.2.2. Excitation Amplitude will exceed Velocity Trap Limit

The Amplitude of Excitation in AutoTune has been entered as a value that would cause the VELTRAP axis parameter to be exceeded. The value will be re-adjusted to prevent this from occurring.

5.4.2.3. Units: Inches/Degrees or Counts

The units of the Amplitude of Excitation in AutoTune may be entered in machine counts or user units (inches or degrees), depending on the axis type. Click the desired radio button.

5.4.2.4. Starting Frequency for Excitation in AutoTune

The Starting Frequency parameter sets the starting frequency of the sinusoid test input. The units are in Hertz. The default value is usually a good starting point. In general, the higher the frequency of the input, the more accurate the AutoTune calculations will be. However, higher frequencies can cause high system velocities and high amplifier currents. If the frequency is too high, over-current errors, velocity traps, or position errors may result. In general, the starting frequency can be raised to 2.0 Hz, 2.5 Hz, or somewhat higher depending on the system. Systems that have high inertia, small amplifiers, or small motors may require the frequency to be reduced to 1 Hz or lower.

5.4.2.5. Ending Frequency for Excitation in AutoTune

The Ending Frequency for excitation in AutoTune will always be 4 times the Start Frequency. Additionally, there will be an intermediate frequency generated 2 times the Starting Frequency.

5.4.2.6. # Points to Collect in AutoTune

The # Points to Collect parameter determines how many data points that the AutoTune program collects. The Gain calculations will be more accurate with more data points collected. The number of points indirectly adjusts the length of time that the sinusoidal input is entered into the system. 7500 points result in a system excitation of 7.5 seconds in length (the data is sampled at 1 msec). The default number of points will work well in almost all cases, and there is a point of diminishing returns if too many points are collected. Two exceptions are where the Starting Frequency is very low, or, where AutoTune Convergence errors have been previously displayed. In these cases, more points may assist the AutoTuning algorithm to come to a correct set of servo loop gains.

5.4.3. Tuning Parameters

The following fields are available on the AutoTune screen.

5.4.3.1. Velocity Bandwidth

The Velocity Bandwidth parameter is used to set the desired closed-loop bandwidth of the Velocity Loop. The default value is a good starting point in most cases. For systems that require very fast movement or are very lightly damped, the bandwidth setting will most likely need to be increased. On very large systems (i.e., a large gantry system) or systems that do not require fast moves, the bandwidth may need to be reduced for optimum performance.

The Position Loop Bandwidth is automatically determined based on the desired Velocity Bandwidth setting and will be chosen to be approximately 20% of the velocity loop bandwidth. It has been found that this Position Loop Bandwidth will give very good performance in almost all cases. If needed, the user can manually raise or lower the PGAIN after AutoTuning has been completed to adjust the Position Loop bandwidth.

Look at bandwidth from the response time point-of-view. A good approximation for the rise time (10% to 90% of the move) of a system given a certain Position Loop bandwidth (w_{pos}) is t_{rise} (sec) =1.8 / w_{pos}. This presumes that the axis parameters such as ACCEL, DECEL, etc. are properly set, so as not to unnecessarily slow down the speed of motion.

5.4.3.2. Damping

The Damping factor parameter determines the damping of the system. This damping factor is sometimes referred to as Zeta (x) in control literature. A damping factor of 0 will result in a system with no damping, large overshoot, and very oscillatory behavior. A damping factor of 1 will result in smooth system response with very little, if any, overshoot. Damping factors >1 can also be used and will result in an over-damped system which will have no overshoot, but will also have a slower response.

Axes with larger inertia, such as an X axis with a Y and a vertically mounted Z axis, will have more optimal gains calculated by the algorithm if the damping factor is much higher (3.4 or greater), assuming the motor/drive can produce the required torque. Most likely, you will need to sacrifice damping and reduce the damping factor to 0.4.

Typically, the value of useful damping factors will be between 0.5 and 1.0. As the damping factor is reduced towards 0, the system step response will be quicker at the expense of larger overshoot and longer settling times. As the damping factor is increased

towards 1, the step response will slow down, but the overshoot and settling time will be smaller. A default damping factor of 0.707 is typically an optimal choice that gives a system a response that blends a small amount overshoot with fast response and quick settling time.

5.4.3.3. Use VFF

If this box is checked, AutoTune will set VFF to 1.

5.4.3.4. Calculate AFFGAIN

If this box is checked, AutoTune will calculate an AFFGAIN

5.5. Manual Servo Loop Tuning

This section and following subsections explain the procedures for tuning a UNIDEX 600 Series controller servo loop with and without tachometer feedback using the AerTune.exe utility. The UNIDEX 600 firmware/utility software package (UTIL600-NT) contains a graphical tool that can be used to display the effects of the servo loop gain settings.

Included in these sections are step-by-step procedures for tuning motors controlled by the UNIDEX 600, to yield optimal performance. Optimal performance can be defined two ways. First, it may be defined as the smallest amount of allowable position error according to user-defined tolerances while still having smooth motion. Alternatively, it may be defined as the smallest amount of position error that is allowable by user-defined tolerances while having minimal settling time without concern for smoothness of motion.

To produce smooth motion, the Velocity Loop takes precedence, and an attempt will be made to have a tight Velocity Loop and a tight Position Loop. To have very little settling time, the Position Loop takes precedence and an attempt will be made to have a tight Position Loop. The UNIDEX 600 uses a dual control loop having an inner velocity loop and an outer position loop. The servo loops update time is defined by the *Enable1KHzServo* global parameter, which allows either a 1 kilohertz or 4 kilohertz servo loop update rate. Refer to Figure 5-8 (Torque Mode), Figure 5-9 (Open-Loop Velocity Mode), and Figure 5-10 (Closed-Loop Velocity Mode) for block diagrams of the servo loop.

Before tuning an axis, the motor and feedback device must be properly configured.

For additional information, refer to the following documentation:
Chapter 2: Getting Started, Section 2.2.1. Axis Configuration
Appendix C: Parameters
U600 Hardware Manual P/N EDU154, Chapter 4: Technical Details.

The AutoTuning algorithm, available from the Tools menu, may be used to automatically tune the servo-loop. If your axis is configured as a gantry, the resultant servo-loop gains must be manually copied to the slaves' axis parameters.

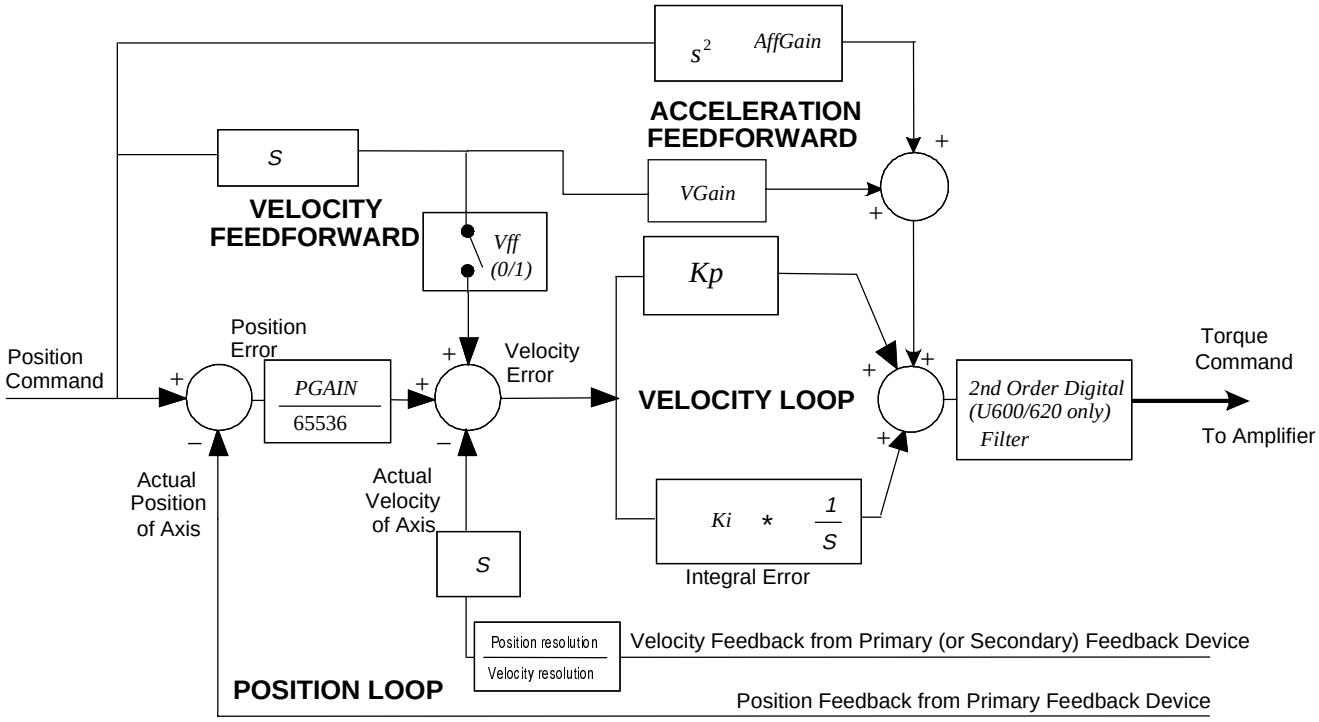


Figure 5-8. Servo Loop Diagram (Torque Mode)

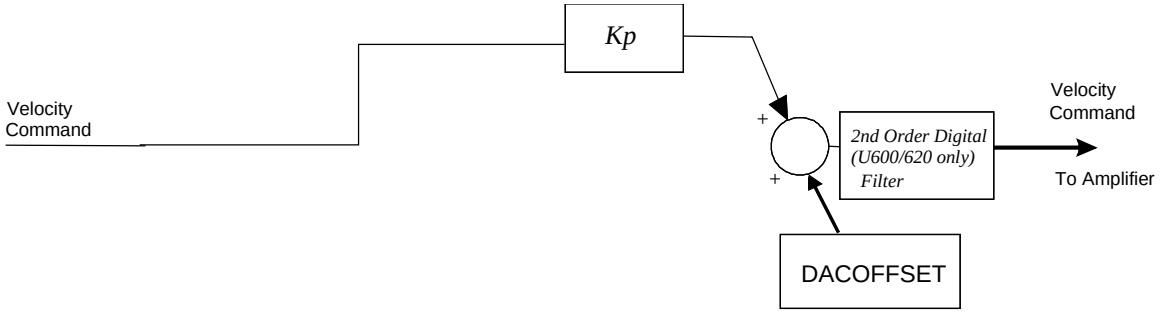


Figure 5-9. Servo Loop (Open-Loop Velocity Mode)

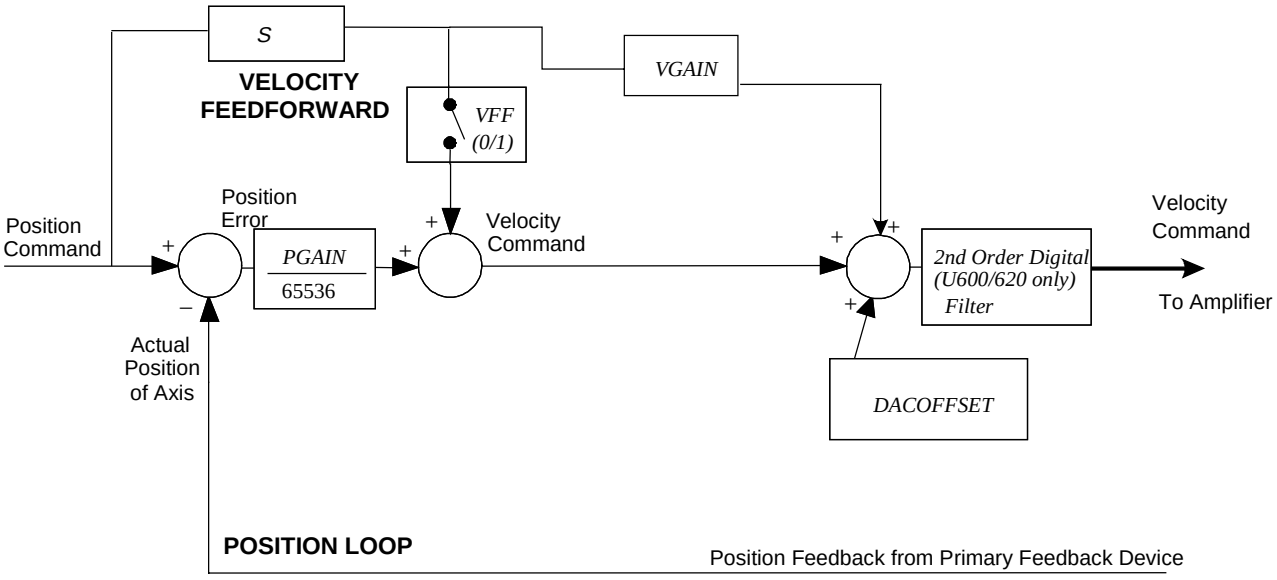


Figure 5-10. Servo Loop (Velocity Mode, Closed Loop)

5.5.1. Kp - Proportional Gain

This is the Proportional Gain. It is part of the Velocity Loop in the UNIDEX 600's Servo Loop. This parameter reduces the amount of velocity error. Also, this gain has a dampening, or stabilizing effect in the servo loop. This is the first servo loop parameter to adjust.

5.5.2. Ki - Integral Gain

This is the Integral Gain. It is part of the Velocity Loop in the UNIDEX 600's Servo Loop. This parameter reduces the amount of velocity error. Moreover, it removes steady-state position errors at the end of a move. This is the second servo loop parameter to adjust.

5.5.3. PGain - Position Gain

This is the Position Gain. It is the only parameter in the Position Loop of the UNIDEX 600's Servo Loop. This parameter reduces the amount of position error and decreases the settling time. This is the third servo loop parameter to adjust.

5.5.4. Vff - Velocity Feedforward Gain

This is the Velocity Feedforward Gain. It is the only parameter in UNIDEX 600's velocity feedforward loop. This parameter is either 1, enabling velocity feedforward, or 0 to disable it. This parameter is used to minimize position errors proportional to velocity.

5.5.5. AffGain - Acceleration Feedforward Gain

This is the Acceleration Feedforward Gain. It is the only parameter in the acceleration feedforward loop in the UNIDEX 600's servo loop. This parameter is used to remove position error during the acceleration and deceleration phase of a move. Normally this parameter has a magnitude of less than 200. The user may not have to set this servo loop parameter greater than zero.

5.5.6. Alpha - AffGain Filter

The Alpha parameter is used within the acceleration feedforward portion of the servo loop. It is responsible for filtering the effect of the AffGain parameter reducing the noise introduced into the servo loop by the rapid velocity changes the AffGain produces. The Alpha parameter has inverse scaling, 65,536 produces minimum filtering and 1, maximum filtering

5.5.7. VGain - Constant Velocity Gain

This parameter is used primarily for tachometer based (velocity loop) systems to reduce the position error (following error) during constant velocity. It can also be used in torque mode systems with large frictional loads. The VGain parameter is multiplied by the commanded velocity to produce a voltage proportional to velocity that is added to the DAC output value to the servo amplifier that minimizes the following error.

5.6. Tuning Procedure for Torque (Current) Mode Servo Loops

Figure 5-11 shows the overall tuning process with the AerTune utility. The tuning process discussed in this section was performed using the "X" (lower) Axis of an Aerotech ATS3220140P X-Y open frame table, with a BM130 AC brushless motor and an AS32030 amplifier at 160VDC. The table had no mounted load, except for that of the Y axis that is physically part of the table. The user's system may behave differently and have different values for servo loop gains. However, the overall process is the same and the same process can be repeated for other axes. The user will essentially follow the procedure below and use it as a guide when tuning the UNIDEX 600 Servo Loop. This procedure does not apply to motors with tachometers.

1. Exercise the axis through a move profile typical to your application.
2. Observe the servo loop performance with the AerTune utility.
3. Make a decision on whether to increase or decrease the value of the servo loop gain parameter or proceed to the next servo loop parameter.
4. Repeat.

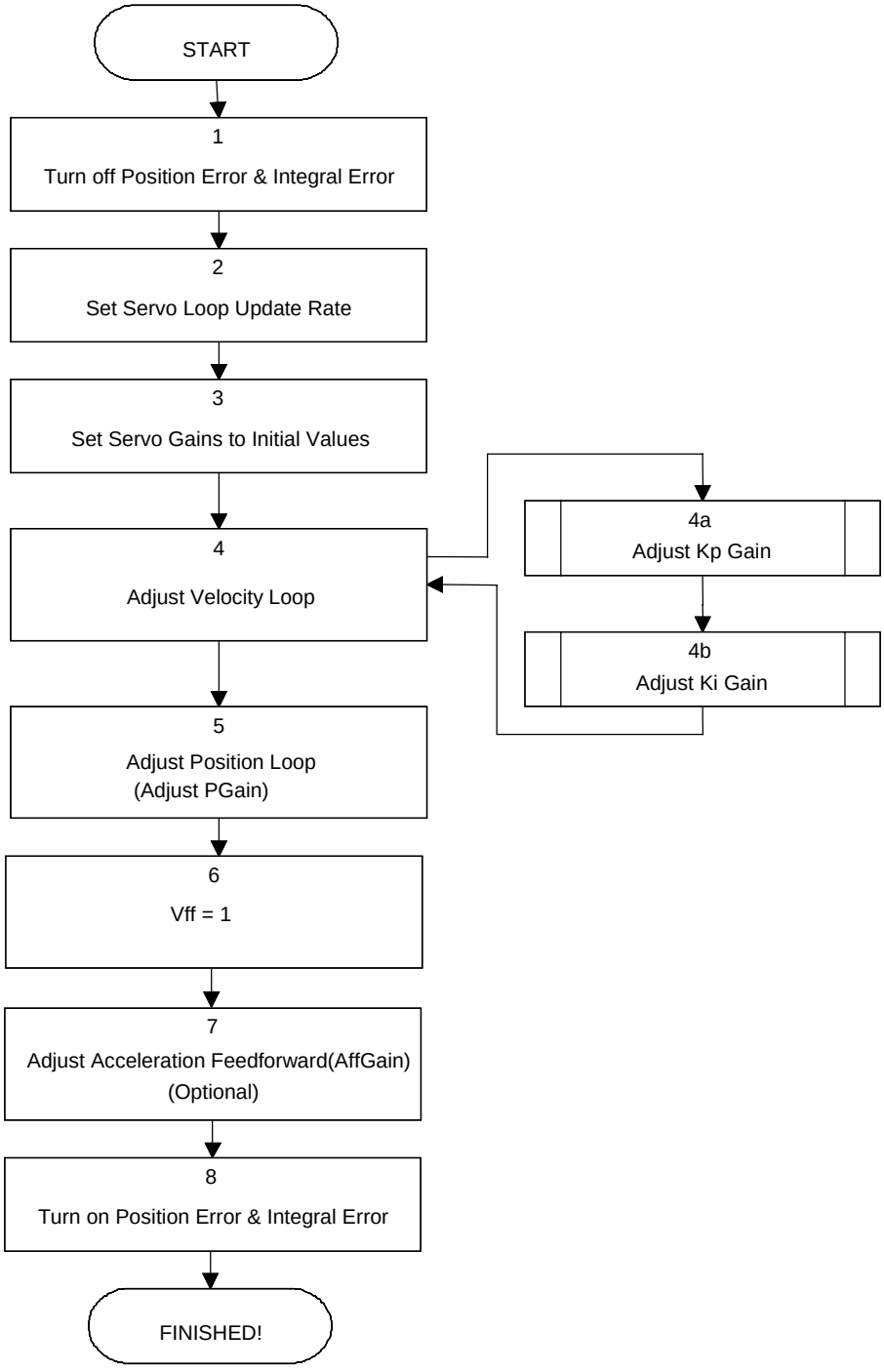
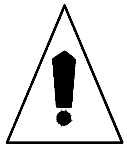


Figure 5-11. Flowchart of Overall Tuning Process

The following is a step-by-step procedure for tuning motors without tachometers.



Please read each step thoroughly before performing the task.

1. Turn off the "Position Error" and "Velocity Error" bits in the *FAULTMASK* axis parameter, by starting the AerDebug utility. Download the axis firmware if this has not been done. Select the axis that needs tuned, with the *AX #* command. Record the current value of the *FAULTMASK* axis parameter by using the *ParmGet* command as follows:

   ParmGet A FaultMask

A value will be displayed after entering the preceding command. Record this value to restore it after tuning the axis.

To disable these faults set the *FAULTMASK* axis parameter to 8398 as follows:

ParmSet A FaultMask 8398

2. Set the servo loop parameters to the initial values according to Table 5-1.

Table 5-1. Initial Torque Mode Servo Loop Parameter Values

PGain	Ki	Kp	Vff	VGain	AffGain	DACOFFSET	Alpha
0	0	1000	0	0	0	0	65536

PGAIN may need to be greater than 0 for short, quick move profiles.

3 Prepare the AerTune utility for tuning by performing the following functions.

   a. Press the maximize button on the title bar so that the window fills the entire screen.
   b. Use the Select pull-down menu to select the axis to be tuned.
   c. Use the Show pull-down menu and select Velocity Command, Velocity Error, and Position Error.
   d. Set the Distance and Speed entry fields for a typical move profile; define the desired ACCELMODE and DECELMODE and enter the appropriate Accel/Decel Rate/Times into the dialog boxes.

When the user selects the "Step+" or "Step-" button, the axis moves the specified distance and direction determined by the Step buttons (positive [+] or negative [-]).

4. Adjust the Velocity Loop using Kp. The PGain and Ki have been set to zero (0) to eliminate the Position Loop. Thus, the only servo loop gain having any effect is Kp.

Even though the user may only be concerned with how well the axis positions, the Velocity Loop cannot be overlooked, as it is the basis for positioning because

Distance = Velocity * Time

The better an axis tracks velocity, the closer it will be to its commanded position.

The objective while adjusting Kp is to minimize the velocity error. Typically the error may be reduced to roughly 5-20 machine counts. The velocity error will not be eliminated completely, Ki will help to accomplish this. Also, most users desire the axis to have a high degree of stiffness. As Kp is increased, observe that the motor shaft (or drive screw) becoming increasingly stiff, or resistant, to external forces. The user may cautiously attempt to turn the shaft manually, and it will become increasingly harder to turn, even though it will not return to its original position (because the position loop gain (PGain) is zero).

No screeching or squealing should be heard from the motor when it is stationary. Noise indicates that Kp is set too high, causing oscillation. It may screech a little during the move, but not while at a rest.

If the motor doesn't move, Kp is too low. Increase the value of Kp and try again by pressing the "Step+" or "Step-" buttons.

If the user is fine tuning the servo loop gains that Aerotech provided for the system, use the existing Kp as your starting point.

Once the motor has begun cycling, a plot will display similar to Figure 5-12. From the graph, it can be seen that there are 75 to 85 machine counts of velocity error. "Kp" should be increased to reduce the amount of velocity error. After repeating this process a few times, the velocity error will look similar to Figure 5-13. From this graph the user can observe that the average velocity error during the move is about 20 machine counts. Likewise, the axis does not oscillate when it is stationary.

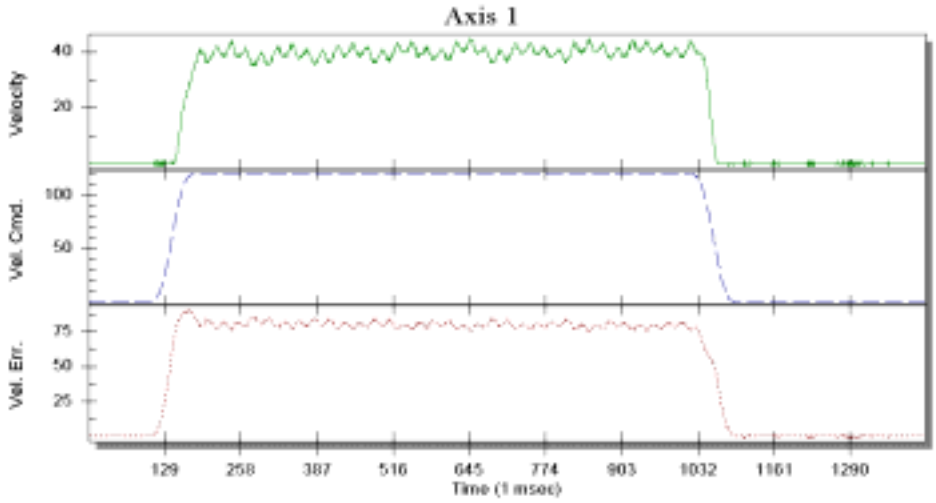


Figure 5-12. Unacceptable Velocity Error

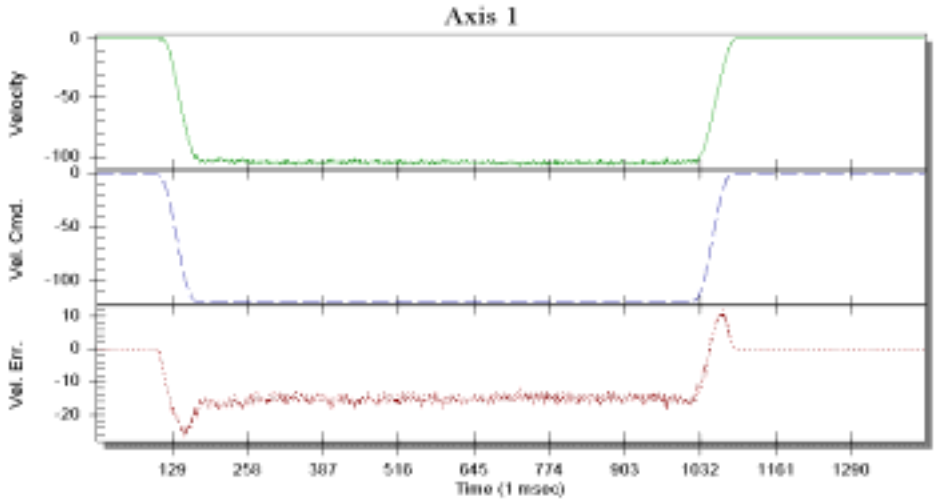


Figure 5-13. Acceptable Velocity Error (While Adjusting Kp)

Begin adjusting Ki with an initial value of 25. The main objective while increasing Ki is to reduce velocity error and position error. As Ki is increased, the error is reduced. However, the position error should not cross the "zero line", indicating position overshoot, otherwise it will increase the settling time of the axis. At this point, Ki has reached its optimal value. Also, a very large Ki will introduce a low frequency oscillation in the position error. From the perspective of the load, this is an unwanted vibration, that may be unacceptable to the user, refer to Figure 5-14.

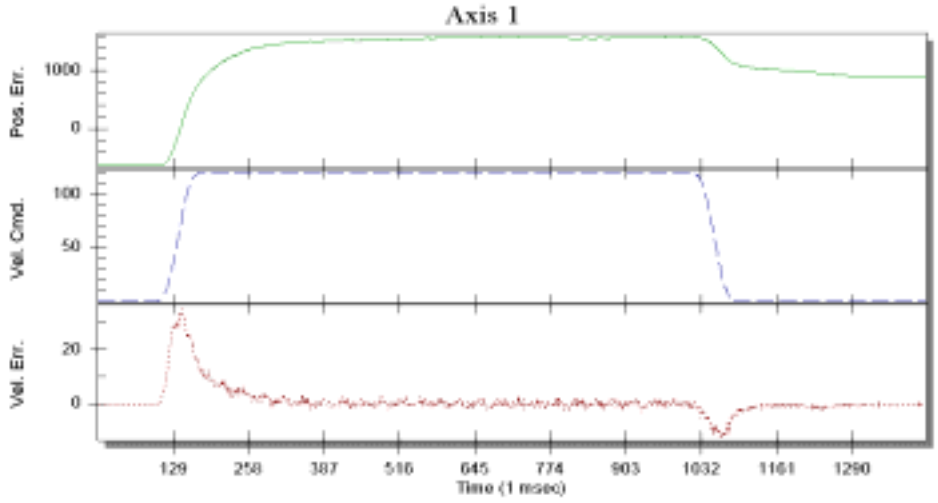


Figure 5-14. Unacceptable Position Error (While Adjusting Ki)

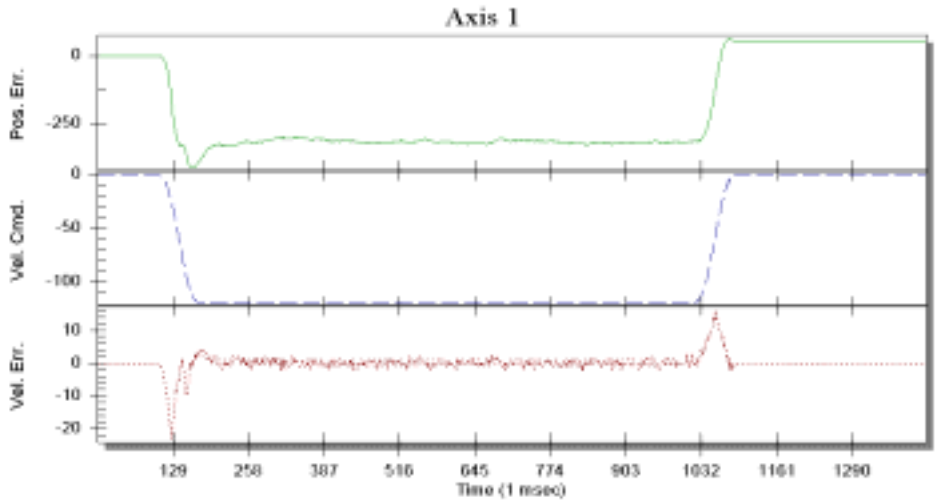


Figure 5-15. Acceptable Position Error (While Adjusting Ki)

Depending on the application, the user does not need to be too concerned with vibration in the axis. However, when the position error crosses through zero, it indicates the velocity loop gain is too high, especially if maintaining position is important to the user. This zero crossing (or position overshoot) causes the settling time to increase and the PGain within the position loop to be increased to keep the motor on its desired trajectory. The two servo loop gains are in effect working against each other instead of complimenting each other.

Shown in Figure 5-15 is a graph with Ki adjusted more optimally. Observing the position error, it is smoother and the position error does not cross through zero indicating positional overshoot. Likewise, the velocity error has been reduced.

5. Adjust the servo loops position gain by varying the PGain. Use an initial value of 1-4 for PGain. As PGain is increased, observe that the position error is reduced. The objective is to adjust PGain until the position error is within an acceptable range for the user or an oscillation occurs.

As previously mentioned, if PGain is too high, the user will encounter a high frequency oscillation (axis vibrates strongly). This causes the UNIDEX 600 to generate an *IAvgLimit* fault, which indicates that the continuous current rating of the motor has been exceeded (the *IAvgLimit* fault is essentially a software fuse).

Shown in Figure 5-16 is a plot of an axis with PGain adjusted optimally. From the plot it shows that settling time is minimal. In other words, their is no damped oscillation at the end of the commanded move; so the axis is "in position" at the end of the commanded move. For comparison, Figure 5-17 illustrates a plot where PGain is too high.

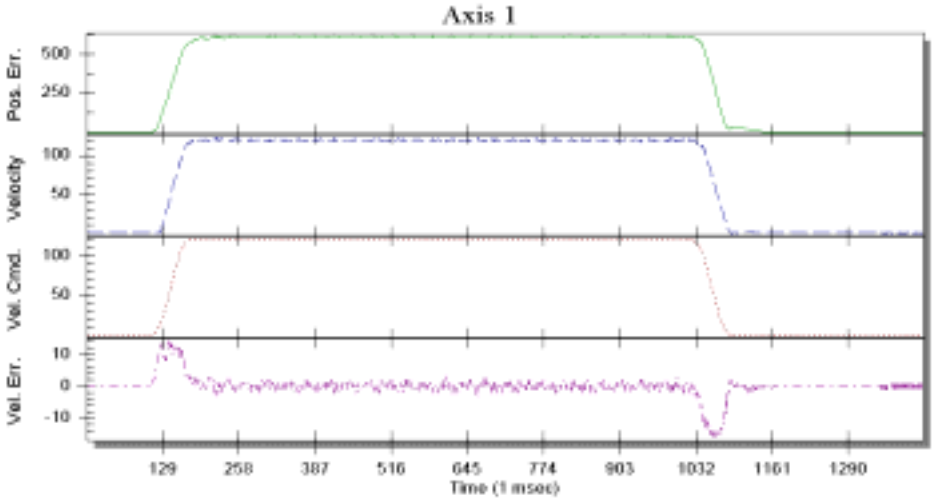


Figure 5-16. Plot Showing an Appropriate Value for PGain

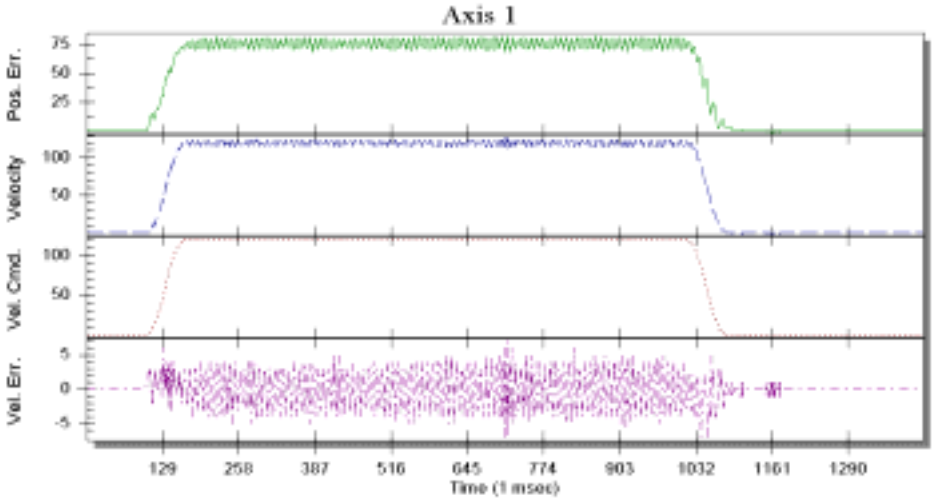


Figure 5-17. Plot Showing Overall Effects when PGain is High

6. Set Vff = 1 to enable velocity feedforward.

The position error has been minimized during the constant velocity portion of the move, refer to Figure 5-18.

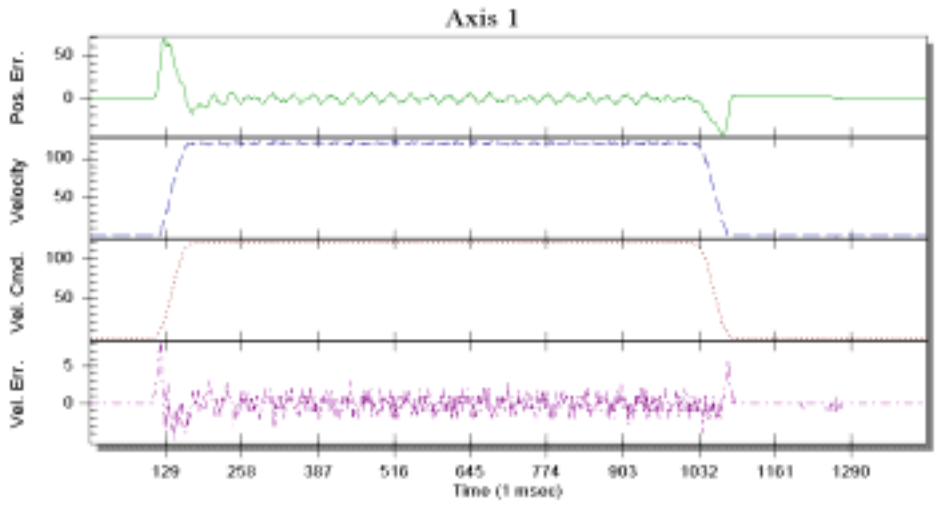


Figure 5-18. Plot Showing Velocity Feedforward Enabled (Vff=1)

7. Minimize position error during acceleration and deceleration by increasing the AffGain servo loop parameter.

A normal value for AffGain is typically less than 200. The AffGain servo loop parameter reduces the large position error visible during acceleration and deceleration. Increasing the AffGain will increase the noise produced by the axis due to the attempt of the servo loop to compensate for velocity changes. This noise may be reduced by varying the *Alpha* parameter. The *Alpha* parameter filters the effect of the AffGain parameter minimizing the noise created by the velocity changes. The *Alpha* parameter is inversely scaled. Setting the parameter to 65,536 produces no filtering and maximum filtering is produced by setting the parameter to 1. Figure 5-19 shows the optimal AffGain setting for out motor/drive/table combination. The Alpha servo loop parameter was not modified from its default setting of 65,536 providing no filtering of the AffGain servo loop parameter.

Adjusting the AffGain servo loop parameter is optional. The user's application may not require it.

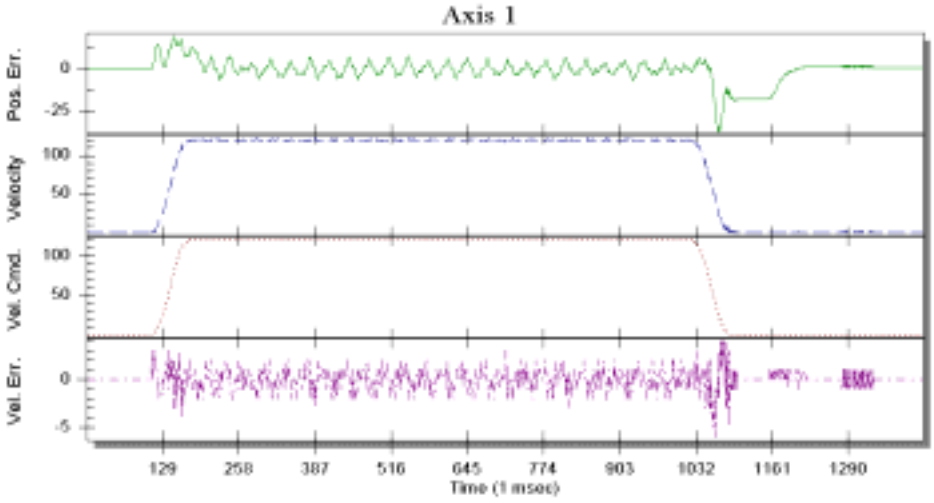


Figure 5-19. Plot Showing Optimal AffGain Adjustment

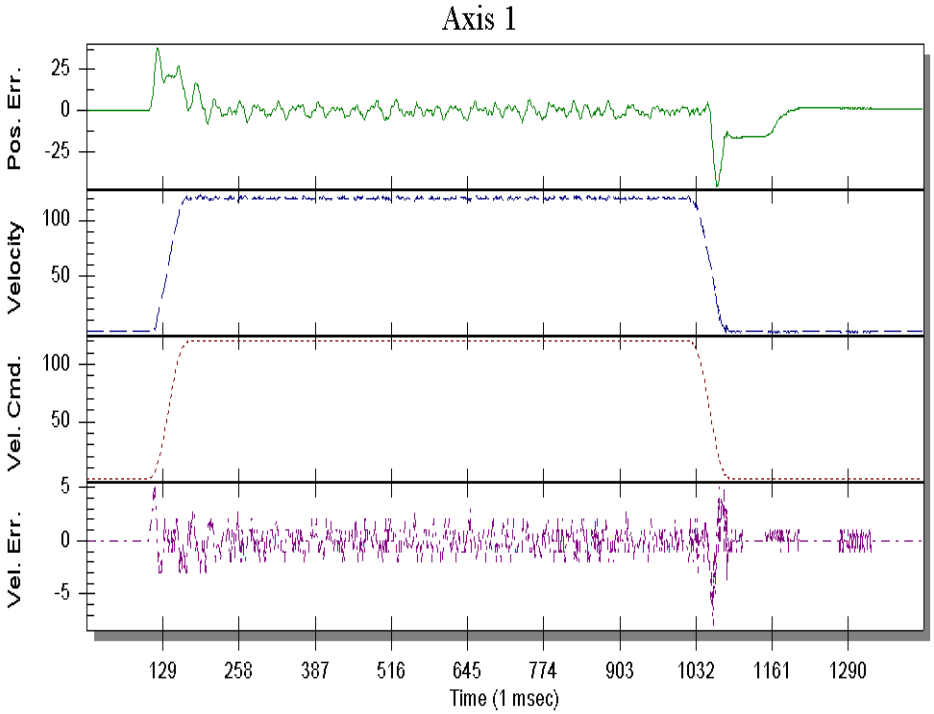


Figure 5-20. Plot Showing Final Performance of ATS3220140P X axis table, with a BM130 motor and an AS32030 amplifier

The final performance of the X axis of the ATS3220140P open frame table is shown in Figure 5-20. The distance of the move in the plot is 110,000 machine steps (110mm). The velocity of the move in the plot is 120,000 machine steps per second (120mm, or 4.7 inches per second). In summary, the points that should be noted include the actual

velocity (and position) of the axis following the commanded velocity (and position) within several machine counts during the constant velocity portion of the move. In addition, it should also be noted that, during acceleration and deceleration of the axis, no ringing or position overshoot following the end of the move. Finally, note the time it takes for the axis to settle to the desired positional accuracy when the velocity command reaches zero at the end of the move. Should smoothness of motion be a performance criterion, it may be desirable to have a more constant velocity (and position) error indicating less fluctuation in the velocity of the axis providing for smoother motion of the axis. The servo loop gain parameters used are:

Kp = 3000	Distance = 110,000
Ki = 250	Speed = 120,000
PGain = 3	AcelMode = 0
AffGain = 60	DecelMode = 0
Alpha = 0	AccelRate = 1,000,000
Vff = 1	DecelRate = 1,000,000
VGain = 0	Accel = 80
	Decel = 80

8. Turn the Position Error and Integral Error Traps on by returning to the AerDebug utility. Turn the "Position Error" and the "Velocity Trap" back on by using the *ParmSet* command as follows:

```
ParmSet A FaultMask ####
```

"####" represents the number returned by the *ParmGet* command in step 2 of this tuning procedure.

This will re-enable these fault conditions.

5.7. Tuning With Tachometer Feedback

The UNIDEX 600 Series controller configures easily for controlling motors with external tachometers providing velocity feedback. To configure the controller for an external tachometer based Velocity Loop, the inherent digital Velocity Loop operation within the controller must be disabled. This is done by setting the digital servo loop proportional gain (Kp) and the integral gain (Ki) to zero. The servo system's Velocity Loop must be adjusted by the potentiometers on the amplifier for the particular motor/tachometer/amplifier/load combination.

When configured this way, the analog outputs of the UNIDEX 600 that normally provide current commands to amplifiers now deliver velocity commands to amplifiers accepting tachometer feedback.

In this configuration, the servo system has the following characteristics.

- The amplifier is configured to accept tachometer based velocity feedback
- The amplifier regulates the Velocity Loop of the servo system. Velocity Loop regulation is accomplished by the Pre-amp section of the amplifier
- The proportional (Kp) and integral (Ki) gain parameters in the UNIDEX 600 controller's servo loop have been set to zero (0), disabling its digital Velocity loop functionality
- The controller is now commanding velocity to the amplifier instead of commanding torque.

5.7.1. Vff - Velocity Feed Forward

The Following Error (position error) that occurs while the axis is moving may be reduced significantly by setting the velocity feed forward gain to one. When the velocity feed-forward function is enabled (i.e., Vff = 1), an added voltage is summed with the velocity command to the amplifier. This signal is proportional to the Velocity Command.

5.7.2. VGain - Constant Velocity Gain

The Following Error (position error) that occurs while the axis is moving at a constant velocity may be reduced by setting the VGain parameter to a non-zero value. This causes the velocity command to be increased proportionally by the commanded velocity, which is scaled by the VGain servo loop parameter.

5.7.3. Servo Parameter Setup for Tachometer Feedback

When configuring a servo loop containing external velocity feedback from a tachometer, the servo gain values shown in Table 5-2 are used.

Table 5-2. Servo Loop Axis Parameters for Tachometer based systems

Parameter	Name	Value	Comments
Position Gain	PGain	Adjust per application	Should be maximized for servo stability and acceptable position error (following error) levels.
Integral Gain	Ki	Always 0	Unused.
Proportional Gain	Kp	Always 0	Unused.
Velocity Feedforward	Vff	Optional	Minimizes following error (position error) of the servo system.
Acceleration Feedforward	AffGain	Always 0	Unused.
Acceleration Feedforward Gain Filter	Alpha	Always 0	Unused.
Constant Velocity Gain	VGain	Adjust per application	Should be maximized for acceptable position error (following error) levels during constant velocity.
Offset to Null Digital to Analog converter offset	DACOffset	Adjust per application	Should be set non-zero to null any offset in the velocity output command that will introduce a velocity offset into the system.

5.7.4. The Servo Loop Parameters for Tachometer-Based Systems

The servo loop parameters have slightly different meanings with tachometer feedback. The following sections describe the parameter definitions.

5.7.4.1. PGain - Position Gain

The Position Gain is the only gain in the Position Loop in the UNIDEX 600's Servo Loop. This gain reduces the amount of position error and decreases the settling time. It is the first servo loop parameter to adjust.

5.7.4.2. Vff - Velocity Feedforward Gain

The Velocity Feedforward Gain is the only gain in the Velocity Feedforward Loop in the UNIDEX 600's Servo Loop. This gain reduces the amount of position error for systems with a tachometer. It is set to one after preliminary tuning is done.

5.7.4.3. Kp - Proportional Gain

Kp is the proportional gain used in systems with tachometers. It must be set to zero (0).

5.7.4.4. Ki - Integral Gain

Ki is the integral gain used in systems with tachometers. It must be set to zero (0).

5.7.4.5. AffGain - Acceleration Feedforward Gain

The Acceleration Feedforward Gain is the only gain in the Acceleration Feedforward Loop in the UNIDEX 600's Servo Loop. For systems with tachometers, it must be set to zero (0).

5.7.4.6. VGain - Constant Velocity Gain

The VGain servo loop parameter is used to reduce the amount of position error during constant velocity mode.

5.8. Tuning Tachometer Loops

The following procedure is a guide for tuning motors with tachometers. Figure 5-21 shows the overall tuning process. The following procedure can be used as a guide when tuning the UNIDEX 600 Servo Loop. The tuning process discussed in this section was performed using the "X" (lower) Axis of an Aerotech ATS3220140P X-Y open frame table, with a 1035DC brush motor and an DS16020 amplifier at 40VDC. The user's system may behave differently and have different values for servo loop gains. However, the overall process is the same and the same process can be repeated for other axes. When adjusting each of the servo loop gains, the user will essentially be following the procedure below:

1. Exercise the axis through a move profile typical to your application.
2. Observe the servo loop performance with the AerTune utility.
3. Make a decision on whether to increase or decrease the value of the servo loop gain parameter or proceed to the next servo loop parameter.
4. Repeat.

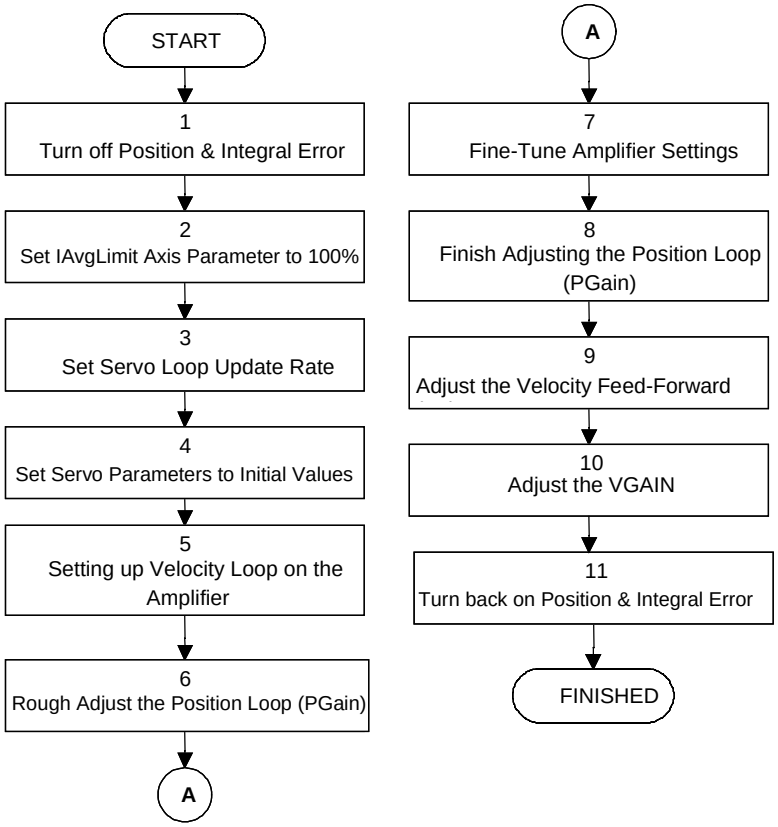


Figure 5-21. Flowchart of Overall Tachometer Tuning Process

The following is a step-by-step procedure for tuning motors with tachometers.

Please read each step thoroughly before performing the task.

1. Turn off the "Position Error" and "Velocity Error" bits in the FaultMask axis parameter. Start the AerDebug utility. Download the axis firmware if this has not been done. Select the axis that you wish to tune, with the AX # command. Record the current value of the *FAULTMASK* axis parameter by using the *ParmGet* command as follows:

   ParmGet A FaultMask

A value will be displayed after entering the preceding command. Record this value to restore it after tuning the axis.

To disable these faults set the *FAULTMASK* axis parameter to 8398 as follows:

   ParmSet A FaultMask 8398

2. Set the *IAVGLIMIT* parameter to 32,767 (100%) while in the velocity mode, scales the maximum commanded velocity to +/- 10 volts (+/- 32,767).

Some low resolution systems (600 line encoders, etc.) or high inertia systems or low velocity systems perform better at a lower update rate such as 1 kHz. If the user doesn't know what to use for this parameter then an update rate of 4 kHz should be used. However, an update rate of 1 kHz can be used. If the update rate is changed, the tuning process must be repeated. The servo loop update rate can be changed to 1 kilohertz by setting the Enable1kHzServo global parameter to 1.

3. Set servo loop parameters to the initial values shown in Table 5-3.

Table 5-3. Initial Servo Parameter Values - Tachometer Tuning

VGain	PGain	Ki	Kp	Vff	DACOffset	Alpha	AffGain
0	0	Always 0	Always 0	0	0	Always 0	Always 0

4. Adjust the Velocity Loop on the amplifier.

If the user has a non-Aerotech amplifier, the manufacturer should provide information for configuring the amplifier to accept a Velocity Command and explain how to optimize the Velocity Loop.

If the user has an Aerotech DS16020/DS16030 servo amplifier, the Velocity Loop is adjusted through the following steps.

   a. Select a fuse to protect the motor for the continuous current rating of the motor and insert it in the appropriate fuse holder of the amplifier. Refer to Figure 5-22 for the location of the fuse holder.

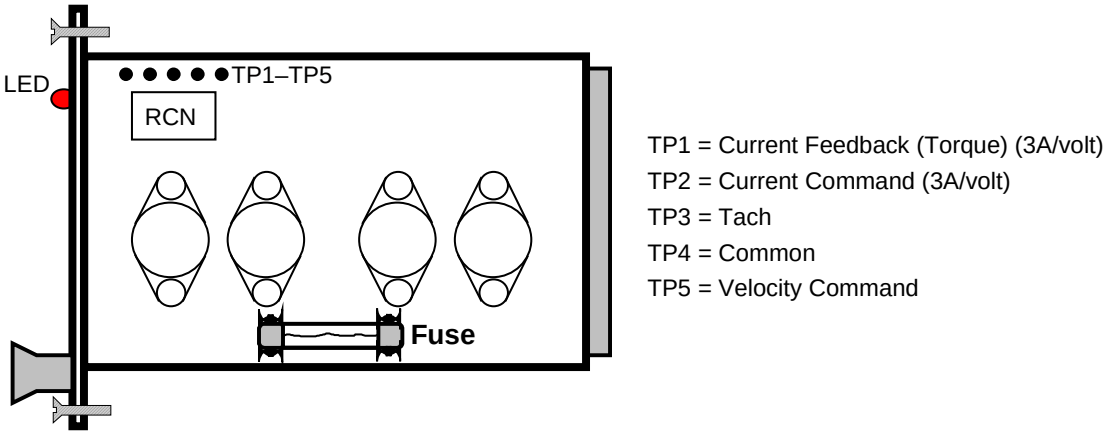


Figure 5-22. Cross-Section of the DS16020/16030 Amplifier

b. Make rough adjustments to the potentiometers on the Aerotech DS16020/16030 servo amplifier as shown in Figure 5-23, then run the axis at its maximum speed. This maximum speed will be provided to the user by Aerotech if the user purchased a complete system from Aerotech. Otherwise, the user will have to calculate the maximum speed. While running the axis at maximum speed, adjust the input potentiometer on the amplifier so that the torque plot in AerTune (torque actually indicates velocity when in the velocity command mode) indicates approximately 26,200 (8 volts).

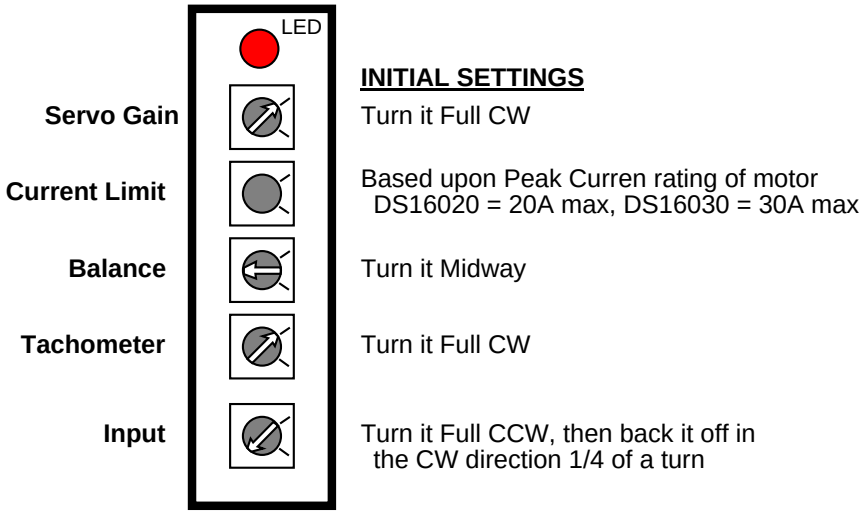


Figure 5-23. Amplifier Potentiometer Layout

The initial setting of the Current Limit potentiometer is based upon the peak current rating of the motor. If the user has a motor with a 10A peak current rating and a DS16020, which has a maximum current output of 20A, set the Current Limit potentiometer to midway for a representation of 10A. Then back it off 1/8 turn in the CW direction. Full CW sets the minimum current and full CCW the maximum.

c. Adjust the Servo Gain (AC gain) potentiometer on the amplifier by first enabling the axis and then turning the potentiometer CCW until the motor oscillates (i.e., the axis vibrates). The motor will produce a screeching sound when it oscillates. Back the gain off by turning it CW until the oscillation stops. Make another 1/8 turn CW from that position so it's not on the borderline of having the motor oscillate.

5. Prepare the AerTune utility for tuning by performing the following steps.
   a. Press the maximize button on the AerTune Window so that its window fills the entire screen.
   b. Use the Select pull-down menu to select the axis to be tuned.
   c. Use the Show pull-down menu, to select Velocity Command and Position Error.
   d. Set the Distance and Speed entry fields for a typical move profile. Define the desired AccelMode and DecelModes and enter the appropriate Accel/Decel Rate/Times into the dialog boxes.

When the user selects the "Step+" or "Step-" button, the axis moves the specified distance and direction determined by the Step buttons (positive [+] or negative [-]).

6. Adjust the PGain servo loop parameter such that the position error is at zero at or near the same time the Velocity Command is at zero. The adjustment to PGain is made by entering a value in the PGain box. If PGain is set too high, the position error will oscillate and the motor will vibrate. The user is not striving to reduce the position error, although that will happen. However, the axis needs to be rough tuned because the following step will be to fine-tune the potentiometers on the amplifier.

Since all Servo Gains are set to zero, the user must set the PGain to an initial value, otherwise the axis won't move. A typical initial starting value is 5.

Once the axis begins moving, Pgain should be increased until the acual velocity tracks the commanded velocity, with minimal position overshoot.

If the motor doesn't move, then PGain is too low. Increase the value of PGain and try again by pressing the "Step+" or "Step-" button.

The axis may tend to drift away on its own when it is enabled. Adjusting the *DACOffset* axis parameter will null the offset causing the drift.

If the user is fine tuning the servo loop gains that Aerotech has setup for the system, use the existing PGain as the starting point.

As PGain increases, the position error will begin to be at zero or near the end of the commanded move. The axis is now roughly tuned, so continue with the following step.

7. This step requires fine tuning the amplifier settings. First, adjust the Balance pot on the amplifier in order to remove any DC offset in the position error. Press the Auto button to cycle the axis. While the axis is moving, adjust the Balance pot and remove any DC offset in the position error. Press the Halt button when the task is done. Ideally the position error will be symmetrical in the positive and the negative directions. However, it is most important that the position error is at zero when the move is complete.

Second, the user will fine tune the Current Limit pot on the Aerotech DS16020/16030 amplifier after commanding the motor to move short, fast moves and observing the current feedback from TP1 on the amplifier with an oscilloscope. In order to do this, perform the following steps.

a. Connect the Oscilloscope leads to TP1 (current feedback) and TP4 (common) on the amplifier.

b. Set the Distance and Speed entry fields to represent a typical short fast move. The idea here is to command the motor to move faster than it is capable, so that the amplifier will saturate. This implies that the amplifier is full on, that is, commanding maximum current. This allows the user to adjust the current limit potentiometer that limits the maximum current that the amplifier can produce.

c. Press the Auto button and allow the axis to cycle.

If the user cycles the axis too long at short fast speeds, the fuse will blow, so be prepared to make the adjustment fairly quickly. The user may desire to use the Step buttons so there is a delay between moves preventing the fuse from blowing quickly.

d. While the axis is moving, adjust the Current Limit pot to limit the current to either 4 times the continuous current rating of the motor or the peak current rating of the motor, whichever is less.

The current feedback on TP1 is 3 amps per volt, so a 2 volt signal on the Oscilloscope would represent 6 amps. Press the Halt button when complete. Figure 5-24 illustrates what is seen after one move.

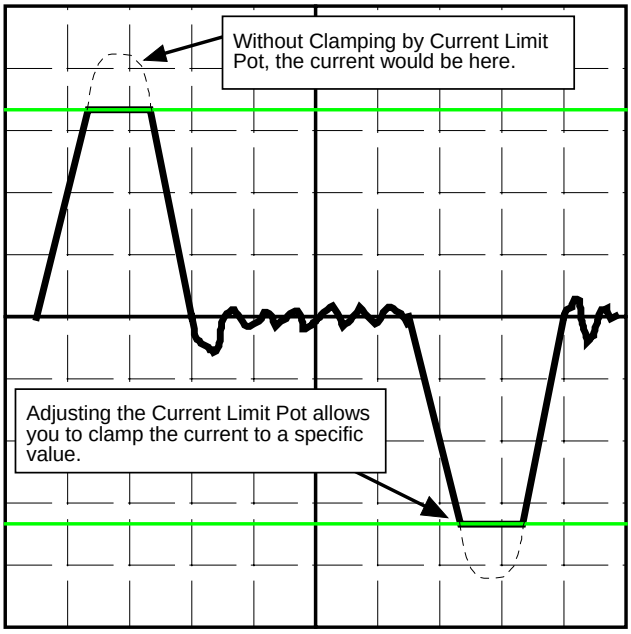


Figure 5-24. Oscilloscope Showing Current Feedback for One Move

Third, if necessary, the user may have to fine-tune the Input pot if unable to achieve maximum speed for the motor. To fine-tune the Input pot, perform the following procedure.

a. Connect the Oscilloscope to TP5 (velocity command) and TP4 (common) on the amplifier.

b. Set the Distance and Speed entry fields to represent a typical move at 1/2 of the maximum speed.

c. Press the Auto button and allow the axis to cycle.

d. While the axis is moving, adjust the Input pot so that when the motor is moving at 1/2 speed the Velocity Command on TP5 is 4 volts.

e. Press the Halt button when completed.

8. Finish, by adjusting the Position Loop (PGain) where the main concern is to strive for a smooth variation in the position error, and to have the position error reach zero at or near the same time the Velocity Command ends. After repeating the process of starting and stopping the axis and adjusting PGain, the axis velocity should track the commanded velocity fairly well and the position error should be at zero at the end of the commanded move. Increasing the PGain will lower the axis settling time.

If the PGain parameter is too high, the motor will oscillate.

9. Enable the Velocity feedforward parameter to reduce the position error (if desired). It is OK to allow some following error in the system. However, if using multiple axes for simultaneous contoured moves, it is desirable that each axis will have the same following error.

10. Increase the VGain servo loop parameter to minimize the position error during the constant veloicty portion of the move. Typically, VGain will be set less than 500.

11. Turn the Position Error and Integral Error Traps on by returning to the AerDebug utility and using the *ParmSet* command as follows:

    ParmSet A FaultMask ####

"####" represents the number returned by the *ParmGet* command in step 2 of this tuning procedure.

This will re-enable these fault conditions.

∇ ∇ ∇

CHAPTER 6: AERPLOT

In This Section:

6.1. Introduction

The AerPlot program allows the user to display up to six plots simultaneously – a mix of axes and/or analog user input information form the U600 Series controller card. This information is displayed in a visual format with a user definable time base reference. This is useful for debugging axes performance as well as monitoring the users analog inputs connected to the U600 Series controller. Figure 6-1 illustrates the AerPlot Screen.

AerPlot provides the capability to load, save, and print captured data. Data points may be captured at a user defined specified rate as the time between points, in milliseconds. The number of points collected and the acquisition type (single or continuous) of the data may also be captured. AerPlot will also display which is collected under CNC program control via the data.

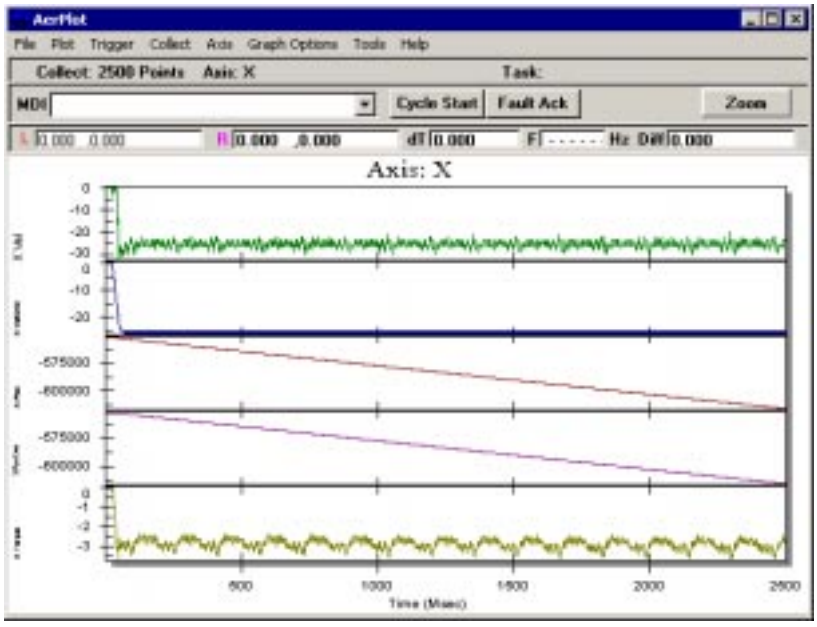


Figure 6-1. AerPlot Screen

6.2. File Menu

The File pull-down menu allows acquired data acquisitions to be saved, loaded, and printed, as well as the settings for the current configuration to be saved. The File menu also allows a comment to be displayed on the plot. This comment is visible on the printed plot.

6.3. Plot Menu

The Plot menu allows up to 6 plots to be selected for display. Each item selected in the Plot area of the Plot Selection Window (see Figure 6-2) will be displayed for each axis checked in the Axis area of the Window. Additionally, any analog inputs checked in the System Data area will also be displayed.

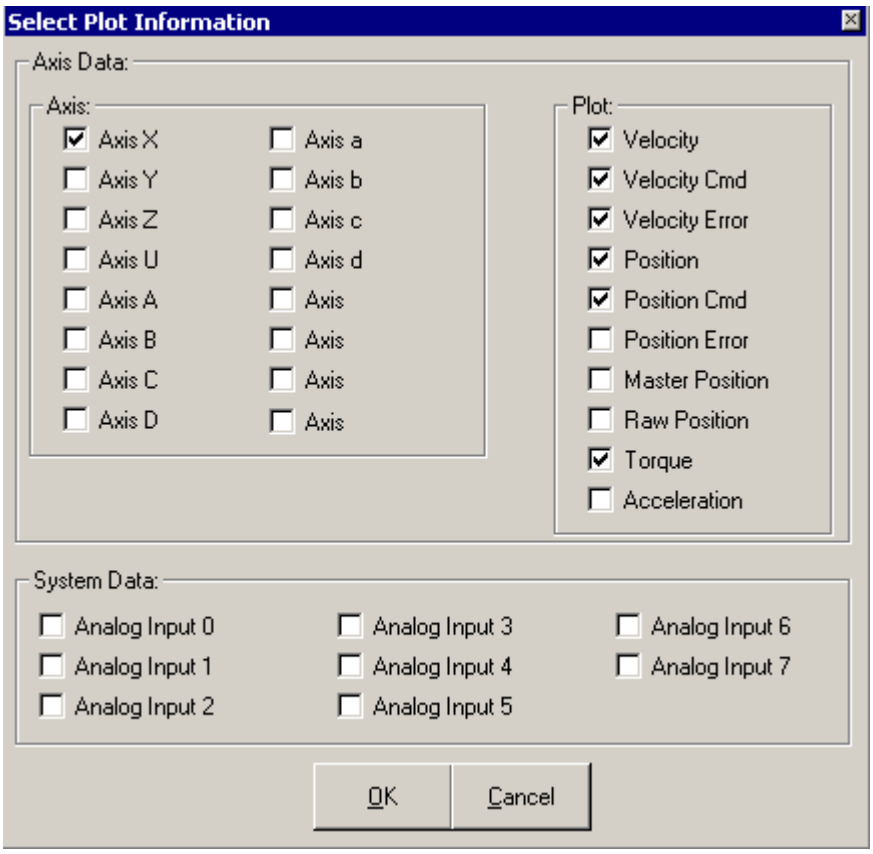


Figure 6-2. Plot Selection Window

6.4. Trigger Menu

The Trigger menu allows the sample rate to be defined. The sample rate determines the number of milliseconds between each sample. This also determines the period of time that AerPlot will collect data, based upon the number of points defined on the collect menu.

The Collect One Set of Data menu selection will collect the specified number of points and update the display.

The Collect Data Continuous menu selection will collect the specified number of points, update the screen, then repeat the cycle until halted by the user.

The Collect Halt Data Continuous menu selection will terminate the Continuous Data Collection mode.

6.5. Collect Menu

The Collect menu allows the number of samples to be defined, which also determines the time period over which the data acquisition takes place. The choices are:

100
250
500
1000
2500
5000
8000
User defined (up to 32,000 samples)

6.6. Axis Menu

The Axis menu allows up to 6 plots to be selected for display. It will display the Plot Selection Window, just like the Plot Menu (see Figure 6-2).

6.7. Graph Options Menu

The Graph Options menu has four selections.

The Grid lines selection allows a grid to be displayed on the X, Y, or both axes.

The Mark Data Points selection allows dots to be displayed for each sample point.

The Units menu allows the units for linear, rotary, and the analog inputs to be selected. The linear axes may be displayed as machine steps (counts), mm, mm/1000, or inches, inches/1000. Rotary axes may be displayed as machine steps (counts) or degrees.

The analog inputs may be displayed as machine steps (counts) or volts. The Time Scale selection allows the time of X axis of the plot to be displayed as seconds, seconds/1000 (milliseconds), or by sample number.

The Zoom menu allows the Zoom feature to be activated, disabled (so that the cursor functions may be used), and to Un-Zoom. These features are available via the Zoom button also. To Un-Zoom using the Zoom button, click the right mouse button on the Zoom button.

6.8. Tools Menu

The Tools menu allows the Status, Control, and Cursor toolbars to be displayed across the top of the plot, below the menu. The FFT analysis selection allows the spectral frequency distribution of a selected item from the "Data to Analyze" menu to be graphically displayed. This will allow you to determine if there is a resonant frequency present in the servo loop and/or mechanics of the system. See the AerTune chapter for more information.

The Hard Reset menu selection allows the controller to be reset to its power-up state. This selection should not be used.

The Fault Acknowledge selection will attempt to acknowledge and clear any faults that are present. The Fault Acknowledge button on the status tool bar will attempt to clear the faults also.

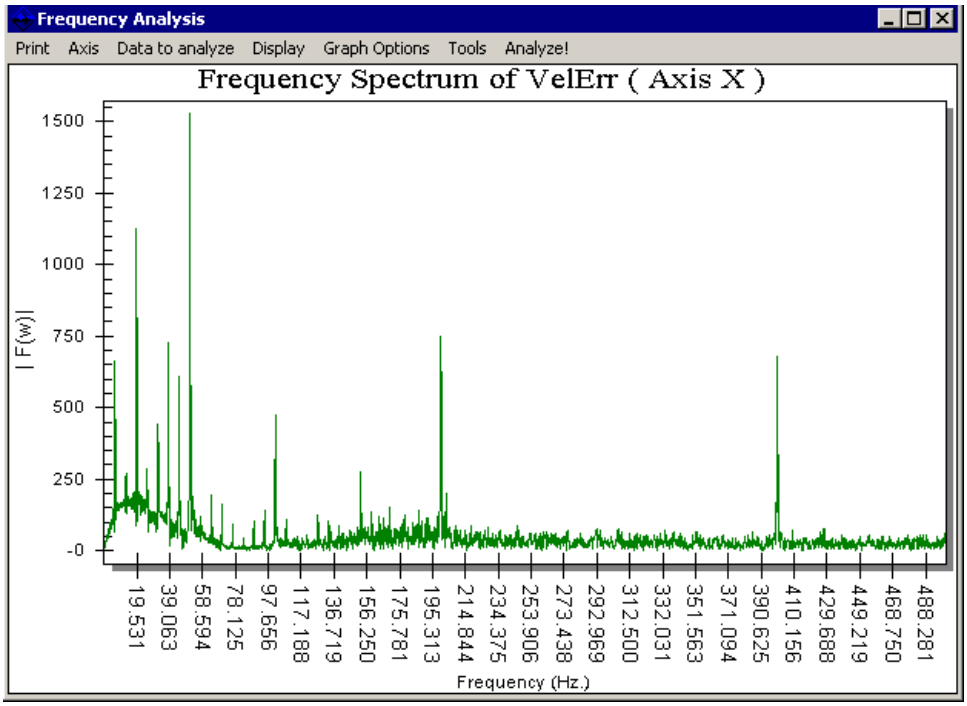


Figure 6-3. FFT Analysis Window

6.8.1. The FFT Analysis Window Menu Description

Print Menu - You may Export or Print the plot.

Data to Analyze - You may select the item to analyze:

Velocity Feedback, Velocity Command, Velocity Error, Position Feedback, Position Command, Position Error, Torque

Display Menu- You may select the number of points to display on the plot:
64, 128, 256, 512, 1024, 2048, 4096, 8192

Tools Menu - Disable Bias Correction
Low Pass Filter Data
Remove DC Bias from Position Data
Remove DC Bias from Torque Data

6.9. Help Menu

The Aerotech U600 Help menu selection will display the information in the online help file for AerPlot.

The About UNIDEX 600 AerPlot will display version information for AerPlot.

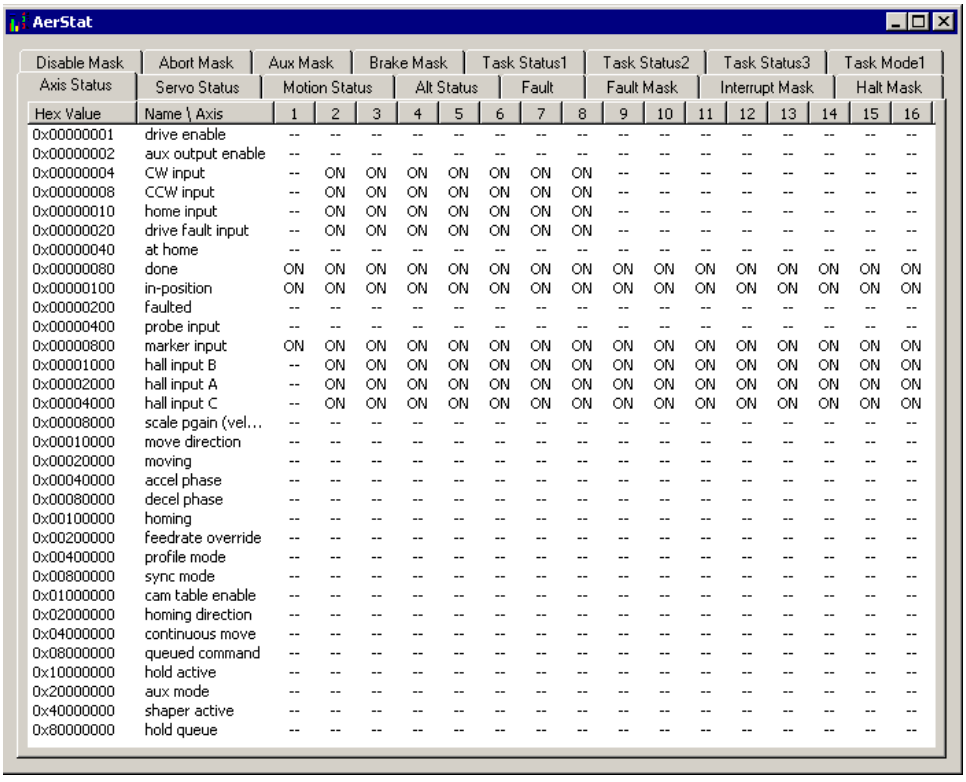
∇ ∇ ∇

CHAPTER 7: AERSTAT

In This Section:

7.1. Introduction

The AerStat utility is a debugging tool that displays the status of all 16 axes of the controller. This is accomplished by displaying the various axes' status, fault masks, faults, Task Status, and Task Mode parameters. This is very useful for initial machine setup and testing, since it displays the state of drive signals, including axes hardware limits, drive enables, drive faults, hall effect signals, and probe input. Pressing the F1 key will display information about all of the parameters displayed by AerStat.



Hex Value	Name \ Axis	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0x00000001	drive enable	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
0x00000002	aux output enable	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
0x00000004	CW input	--	ON	ON	ON	ON	ON	ON	ON	--	--	--	--	--	--	--	--
0x00000008	CCW input	--	ON	ON	ON	ON	ON	ON	ON	--	--	--	--	--	--	--	--
0x00000010	home input	--	ON	ON	ON	ON	ON	ON	ON	--	--	--	--	--	--	--	--
0x00000020	drive fault input	--	ON	ON	ON	ON	ON	ON	ON	--	--	--	--	--	--	--	--
0x00000040	at home	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
0x00000080	done	ON	ON	ON	ON	ON	ON	ON	ON	ON	ON	ON	ON	ON	ON	ON	ON
0x00000100	in-position	ON	ON	ON	ON	ON	ON	ON	ON	ON	ON	ON	ON	ON	ON	ON	ON
0x00000200	faulted	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
0x00000400	probe input	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
0x00000800	marker input	ON	ON	ON	ON	ON	ON	ON	ON	ON	ON	ON	ON	ON	ON	ON	ON
0x00001000	hall input B	--	ON	ON	ON	ON	ON	ON	ON	ON	ON	ON	ON	ON	ON	ON	ON
0x00002000	hall input A	--	ON	ON	ON	ON	ON	ON	ON	ON	ON	ON	ON	ON	ON	ON	ON
0x00004000	hall input C	--	ON	ON	ON	ON	ON	ON	ON	ON	ON	ON	ON	ON	ON	ON	ON
0x00008000	scale pgain (vel...	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
0x00010000	move direction	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
0x00020000	moving	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
0x00040000	accel phase	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
0x00080000	decel phase	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
0x00100000	homing	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
0x00200000	feedrate override	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
0x00400000	profile mode	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
0x00800000	sync mode	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
0x01000000	cam table enable	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
0x02000000	homing direction	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
0x04000000	continuous move	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
0x08000000	queued command	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
0x10000000	hold active	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
0x20000000	aux mode	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
0x40000000	shaper active	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
0x80000000	hold queue	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

Figure 7-1. AerStat Screen

7.2. Overview

AerStat has no controls, only folder tabs allowing the user to select the parameter to be displayed. A false signal or state is indicated by a '-' and a true state is indicated by a 'ON' indicator. The unused axes may be slid off the screen to remove useless information by grabbing the vertical bar to the right of the axes number with the left mouse button, and sliding it to the right of the screen. The information on the screen is updated at a rate of 250 milliseconds (4 times per second). Refer to Appendix C: Parameters for a full description of parameters.

∇ ∇ ∇

CHAPTER 8: AERREG

In This Section:

8.1. Introduction

AerReg is Aerotech's operating system registry editor program that allows registry information to be created or edited by the user without knowing the required structure of the registry database.

When the UTIL600 software is installed, it will start the AerReg utility so that you may enter the registry settings. However, until the PC is rebooted, the device driver will not be loaded. That means that you can not click the Test Card button until after the first time the PC is rebooted following the installation of the UTIL600 software.

8.2. Editing Registry Entries

To edit an existing UNIDEX 600 Series controller registry entry click on the '+' sign to the left of the UNIDEX 600 entry (see Figure 8-1). This will numerically display the cards present in your PC that, in almost all cases, is one.

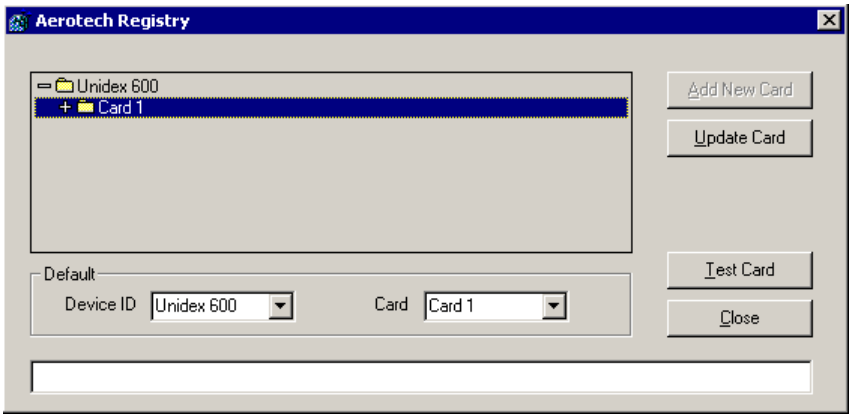


Figure 8-1. AerReg Screen

8.2.1. Finding and/or Creating a "Card 1" Entry

A "Card 1" entry is required by Aerotech's software. If a '+' sign is displayed next to the "UNIDEX 600" entry, click on the '+' sign to display the "Card 1" entry. If there is no '+' sign displayed, or "Card 1" entry, highlight "UNIDEX 600" and select the "Add New Card" button, otherwise, highlight "Card 1" and select the "Update Card" button.

8.2.2. Modifying the "Card 1" Entry

After finding/creating the "Card 1" entry above, and proceeding to the next screen, you must enter the I/O Base address as defined by the jumpers on the controller card. You must also enter the IRQ number defined by the jumpers on the controller card. See the U600 Hardware manual (EDU154) for more information. All other fields should default to the proper values, with the exception of the "Optional PSO Support" field.

Selecting the 'Browse' button for an entry expecting a firmware image file (type *.IMG) displays a file selection box allowing the user to select a file from the default directory location or locate a file in another path.

If you have a PSO-PC card, you must enter the base address of the PSO-PC card as defined by the jumpers on the card. See the PSO-PC manual (EDO105) for more information. The PSO image file name will default to the correct value.

To edit the database entry for that card, highlight an entry and click the update button. A dialog box will appear, allowing all of the entries to be modified.

After updating/creating a new entry, click 'OK' to return to the main screen. Click 'OK' again to close the program, after double checking the data entered into the registry.

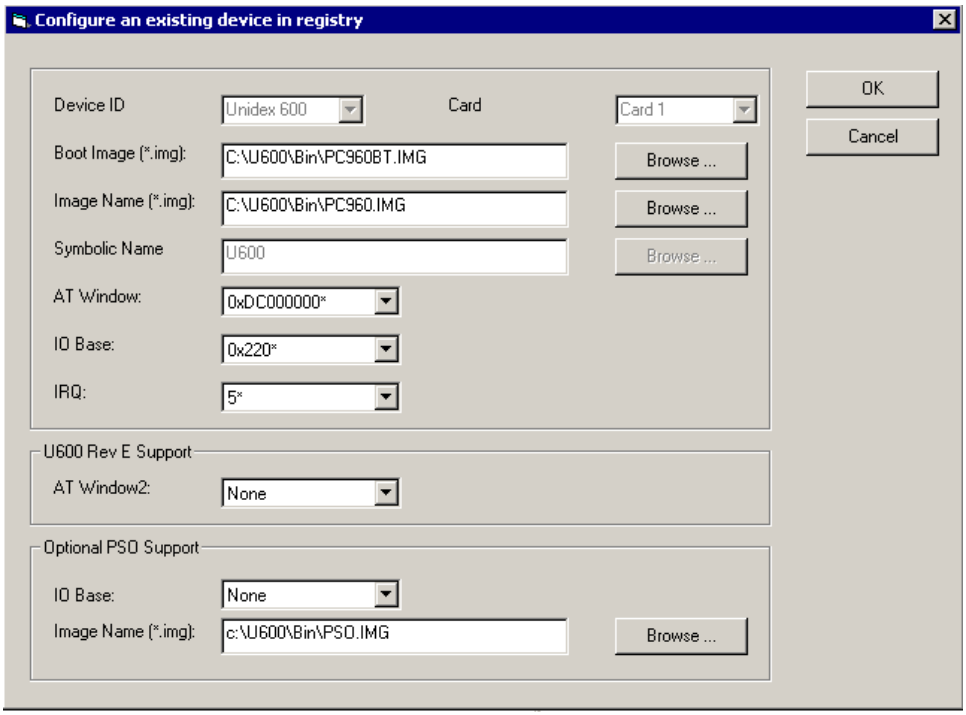


Figure 8-2. AerReg Registry Editor Screen

The default registry entry's with their corresponding default values for:

The UNIDEX 600/620 controllers are:

Image	=	C:\U600\BIN\PC960.IMG
Boot Image	=	C:\U600\BIN\PC960BT.IMG
Device Driver	=	C:\U600\BIN\U600.VXD (WIN 95) [U600 (WIN NT)]
IO Base	=	0x220
ATWindow	=	0xDC000000
IRQ	=	0x5

PSO (the default values for the PSO-PC, if present, are):

IO Base	=	0x310
Image Name	=	C:\U600\BIN\PSO.IMG

The IOBase address and IRQ values are determined by hardware jumpers on the UNIDEX 600/620 controller card.

∇ ∇ ∇

CHAPTER 9: AERPLOT3D

In This Section:

9.1. Introduction

The AerPlot3D utility allows the user to continuously plot the motion (tool path(in either a 2D or 3D format. This utility may not collect data while AerPlot, AerTune, or AerPlotIO are collecting data. The following items may be plotted:

Velocity Command
Velocity Feedback
Velocity Error
Position Feedback
Position Command
Position Error
Torque
Acceleration

In 2D plotting, two items may be selected for each axis. In 3D plotting, only 1 item per axis may be plotted. The data plots may be saved and later reloaded into AerPlot3D.

9.1.1. File Menu

The select setup menu selection allows you to select one of the five previously defined configurations defined via the Save Setup menu selection, as shown in Figure 9-1.

The Save Setup menu selection allows up to five configurations to be defined for AerPlot3D, each with a comment describing their configuration.

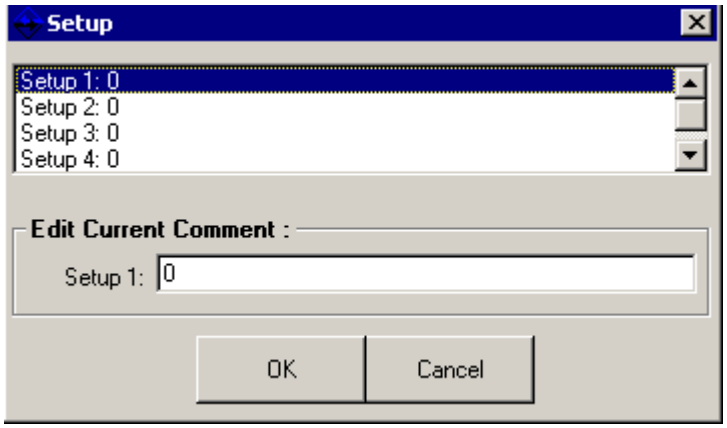


Figure 9-1. The Setup Screen of AerPlot3D

The read/write data file menu selection allows the plot to be saved to a data file that may be reloaded at a later time.

9.1.1.1. Writing a Plot Data File

To write data to a file,

1. Select the read/write data file menu selection
2. Enter a filename or select the Browse button to overwrite an existing file
3. Select the Write button
4. Select the Start button from the menu on the main window
5. When data collection is complete, select "Stop Write"
6. Select the X box in the upper right of the Data File window to close it

9.1.1.2. Reading and Displaying a Plot Data File

To read data from a file,

1. Select the read/write data file menu selection
2. Enter a filename or select the file via the Browse button
3. Select the Read button
4. Select the X box in the upper right of the Data File window to close it

The Default Setup menu selection will restore the AerPlot3D default configuration.

The Exit menu selection will exit AerPlot3D.

9.1.2. Plot Type Menu

The Plot Type menu allows you to select either 2D plotting or 3D plotting. This also defines the number of axes that may be selected for display via the Axis Select selection of the Setup menu.

9.1.3. Setup Menu

The Setup menu has 4 selections.

The Axis Select menu will display the axis select window (shown in Figure 9-2). The number of axes that may be selected is determined by the Plot Type menu selection. Each axis may have its position feedback, position command, position error, velocity feedback, velocity command, velocity error, torque, or acceleration selected for display.

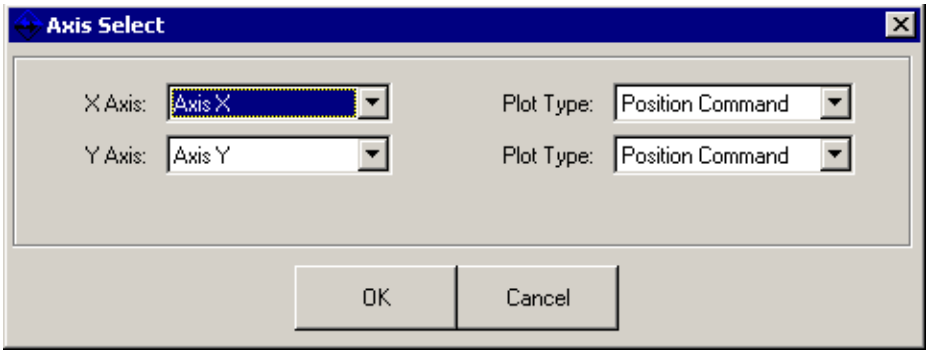


Figure 9-2. The Setup Screen of AerPlot3D

The Axis Min and Max menu selection allows the Min and Max range of the X and Y axes of the plot screen to be defined. See Auto Scaling, Section 9.2., for more information.

The Update and Sample Rates menu selections allow the rate at which the items selected for plotting will be sampled, and the rate at which the display will be updated, by AerPlot3D.

The Show Min and Max Plotted Values menu selection determines whether the Min and Max Value Plotted Window will be displayed (similar to Figure 9-3), allowing you to optionally click the Update X, Y, and Z buttons to rescale the displayed range of the plot.

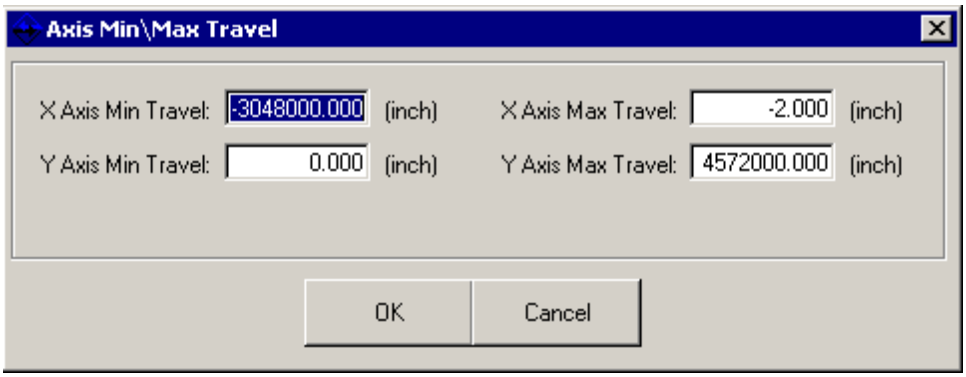


Figure 9-3. The Axis Min/Max Travel Screen of AerPlot3D

9.1.4. Colors Menu

The Colors menu allows you to define the displayed color for the following:

Desk Color
Graphic back color
Graphic fore color
Plotting color
Shadow color
Text color

The Default Colors menu selection allows changes in any of the above to be reset back to their default values.

9.1.5. Grid Lines Menu

The Grid Lines menu allows a grid to be displayed on the X, Y, or both axes.

9.1.6. Units Menu

The Units menu allows the units for the X, Y, and Z (if 3D mode is active) to be selected. Each may be displayed as machine steps (counts), or user units.

9.1.7. Start Menu

The Start menu will begin plotting the selected lines

9.1.8. Stop Menu

The Stop menu will discontinue plotting the selected item. If the "Show Min and Max Plotted Values" item is selected on the Setup menu, the Min and Max Value Plotted window will be displayed (similar to Figure 9-3), allowing you to optionally click the Update X, Y, or Z buttons to rescale the displayed range of the plot. Click the "Close" button when done.

9.1.9. Resume Menu

The Resume menu will continue plotting the selected data items if the Suspend menu has been selected.

9.1.10. Suspend Menu

The Suspend menu will suspend plotting of the selected data items. The Resume menu will continue plotting.

9.1.11. Help Menu

The Help menu has 2 selections.

The Aerotech UNIDEX 600 Help menu selection will display the information in the online help file for AerPlot3D.

The About U600 AerPlot3D will display version information for AerPlot3D.

9.2. Auto Scaling the Display in AerPlot3D

To automatically scale the display window in AerPlot3D (assuming that your CNC program is in G90 mode):

- Select Start form the menu bar to trigger AerPlot3D
- Run your CNC program
- Select Stop form the menu bar
- Click the "Update X" and "Update Y" (and "Update Z" where applicable) buttons
- Select "Start", again, to trigger AerPlot3D
- Run your CNC program a second time

∇ ∇ ∇

CHAPTER 10: AERPLOTIO

In This Section:

10.1. Introduction

The AerPlotIO utility will display the state of the virtual binary inputs/outputs and registers versus time. Designed to be a logic analyzer, it can collect four 32-bit words of data. Each 32-bit word can represent 32 binary inputs/outputs, or 2 register inputs/outputs. The number of binary traces that may be selected will be limited if all of the bits selected do not fall within four 32-bit blocks.

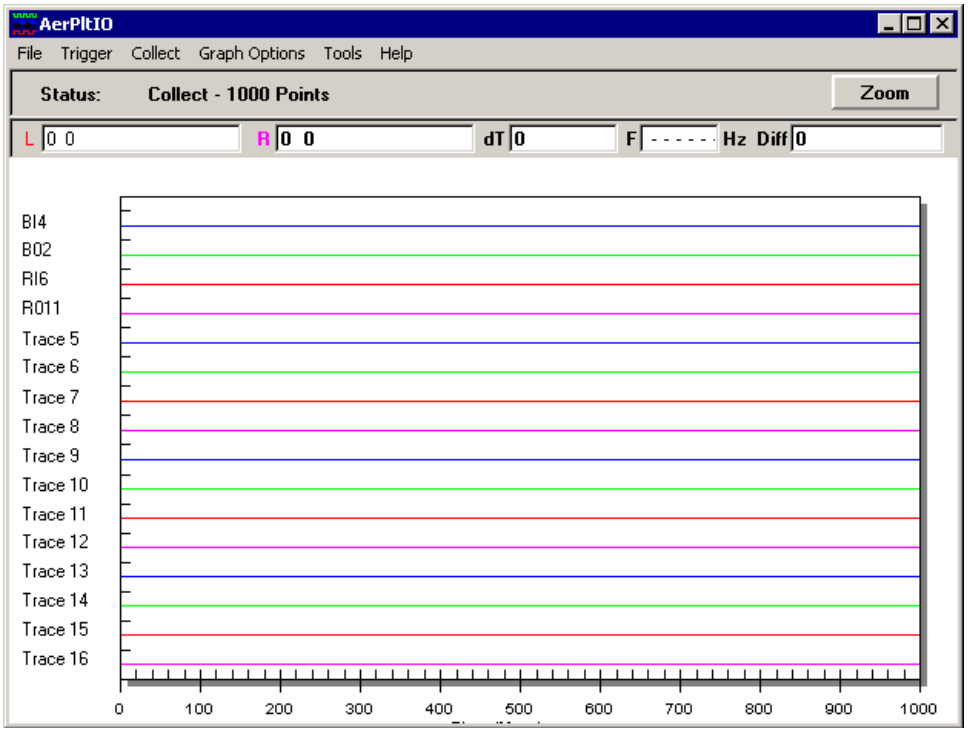


Figure 10-1. AerPlotIO Screen

To add a trace:

Double click on the trace label (i.e., Trace 1). This will display a screen that will allow you to select which binary input, binary output, register input, or register output to display. When trace has been added, the name will be changed from Trace X to the name of the binary input/output or register input/output that was chosen to be displayed for that trace.

When register inputs or register outputs are displayed, the actual value of the register input/output is not displayed. If a register input/output does not change value, a 1 is displayed. If a register input/output changes, a 0 will be displayed for one millisecond to represent the fact that the register input/output changed. The value of the register input/output at any given point in time can be determined using the cursors.

File Menu	-	Allows the user to Save, Export, Load, or Print the Plot File
Trigger Menu	-	Allows the user to collect one set of data, collect data continuously, halt continuous data collection, update the conditional triggering, and update the sample rate
Collect Menu	-	Allows the user to specify how many points to collect
Graph Options	-	Allows the user to use graph lines, mark data points, change the time scale, or zoom the graph
Tools	-	Allows the user to control whether the status and the cursors window can be seen
Help	-	Allows the user to activate the U600 help file

To set up conditional triggering:

In order to enable conditional triggering, the conditional triggering enabled check box must be checked. 16 binary inputs, 16 binary outputs, a register input, or a register output may be used for conditional triggering.

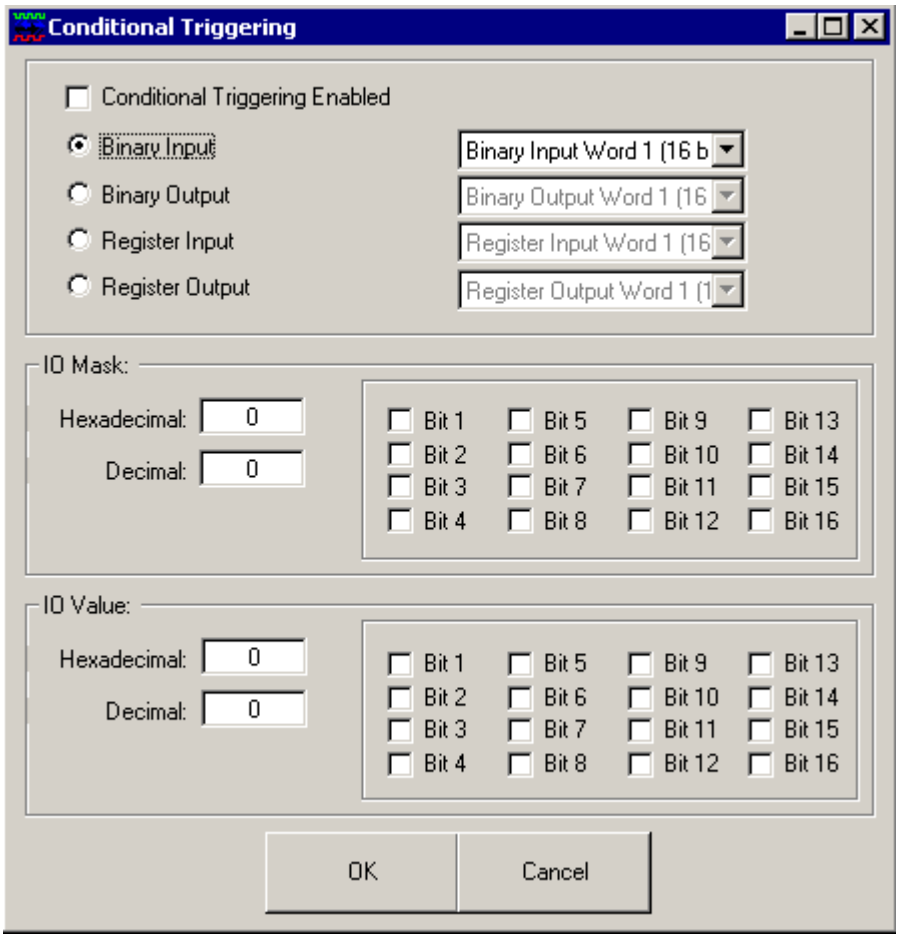


Figure 10-2. AerPlotIO Screen

The IO Mask and IO Value fields need to be entered. The IO Mask field determines which bits in the trigger word are the state of the conditional trigger. If one of the check boxes is not checked, this represents a "don't care" condition. For example, if Binary Input Word 1 is selected, and the IO Mask is equal to 3, the program would only look at the first two bits of Binary Word 1 and the other fourteen bits would be don't cares. If a register input/output is being used, and the collection should trigger when the register reaches a certain value, the IO Mask should be –1.

In the case of binary inputs/outputs, the IO Value specifies the state of the bits that trigger the data acquisition. For example, if Binary Word 1 is selected and the IO Value is equal to 1, bit 1 must be a one and all other bits that are in the IO Mask must be zero to trigger data collection. For register inputs/outputs, the IO Value is the value of the register that will trigger data collection.

∇ ∇ ∇

CHAPTER 11: FILTER

In This Section:

11.1. Introduction

The Filter utility will calculate the coefficients of the controller's second order digital filter in the format required for the A1, A2, B0, B1, B2 axis parameters (for more information, see Axis Parameters in Appendix C). The sample frequency should be set to your servo loop update rate 4000 (default) or 1000 Hz. Do not try to use a filter for any frequency range greater than half of the servo loop update rate.

You must disable the axis before entering the filter constants.

The "Remove Filter" button will reset the digital filter to no filtering. You must click the "Download" and "Write to File" buttons also.

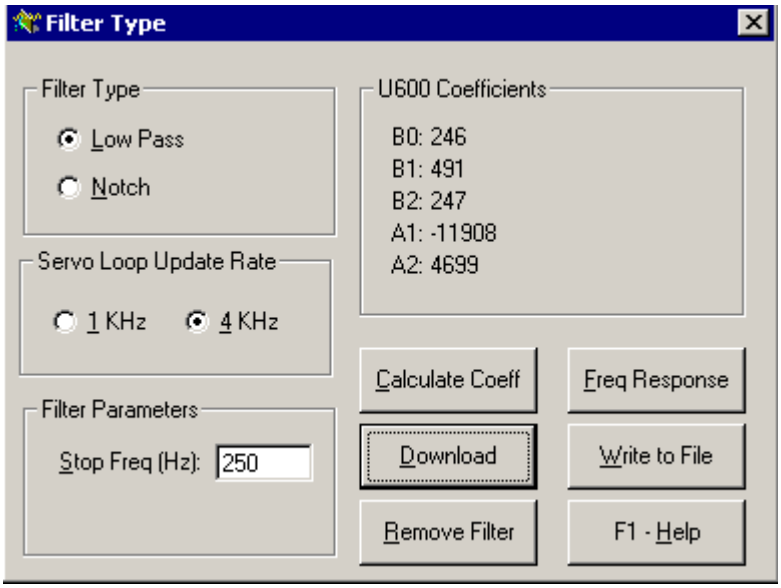


Figure 11-1. The Filter Screen

∇ ∇ ∇

CHAPTER 12: SETUP WIZ

In This Section:

12.1. Introduction

The SetupWiz.exe program will prompt you to configure the common (Global, Task, Machine, and Axis) parameters required for typical applications. It will also configure the axes by use of the axis configuration wizard, which is integral to the MMI600. It will save all of the parameters to the default .Ini files as defined by the MMI600 Setup Page. See Figure 12-1.

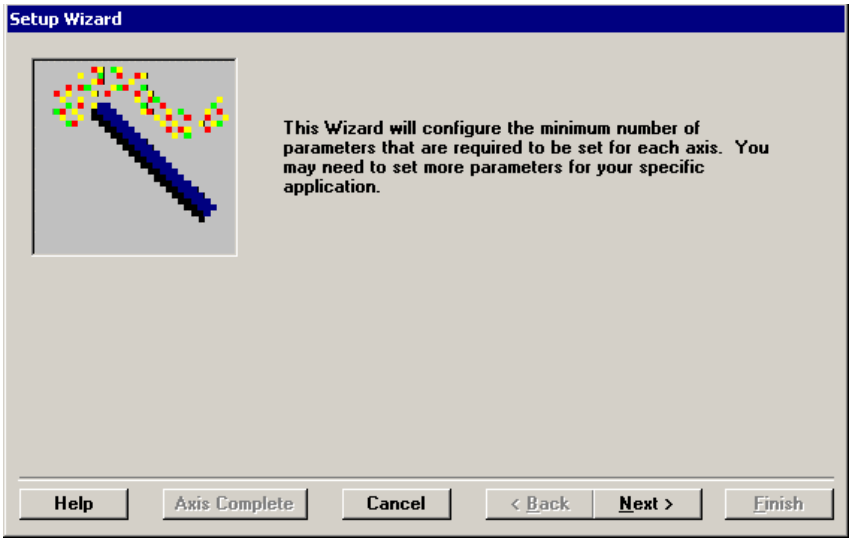


Figure 12-1. The Setup Wizard Start Screen

To access a description of the required information for each page of the setup wizard, press the Help button (on each page).

12.2. Axis Names and Number

Enter the number of axes present in your system and press the <ENTER> key or tab out of the field. This will tell the Wizard how many axes it must prompt you to configure.

Select the desired units (English inches or Metric millimeters) that you would like to use when entering values into the Wizard.

Enter the desired axis name (see Section 12.2.1.) for each axis.

Selecting 'Next' will advance you to the next Wizard configuration screen, 'Back' will take you to the previous Wizard screen, 'Cancel' will exit the Wizard without saving any changes to the axis configuration.

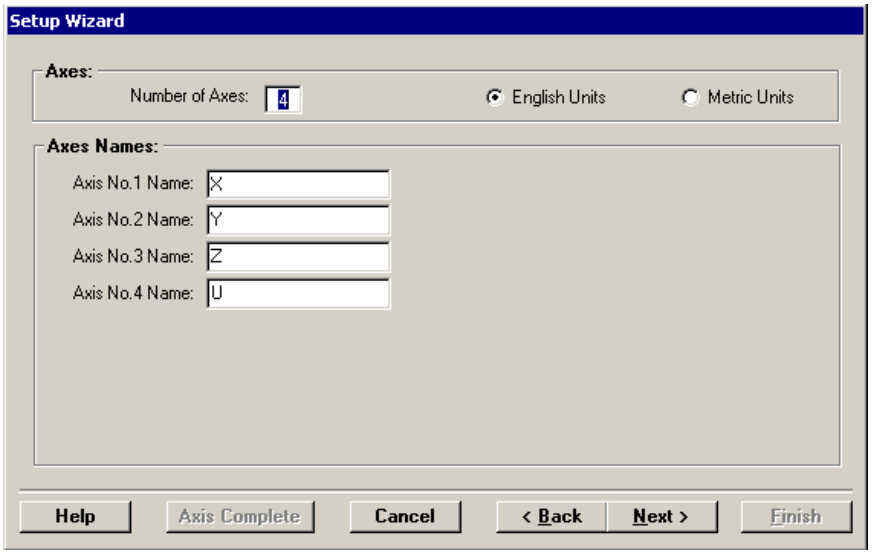


Figure 12-2. The Axis Name/Number Configuration Screen

12.2.1. Axis Names

This allows the default axis names:

X, Y, Z, U, A, B, C, D, x, y, z, u, a, b, c, d

to be changed by the user to an axis name of their choice, with a few exceptions. All reserved key words and key letters may not be used as axis names. Key letters are those used for F, G, and M codes, etc. Key words are those used for commands, such as BIND, FOR, IF, etc. Since there is an axis named X and x, the case of axis name is significant. Axis names are limited to 32 characters, maximum.

If you reassign an axis name, you must exit the MMI600 and restart it for the change to take effect.

12.3. Configuring Axis Type

Select the type of motion [*Machine/Type*] produced by the axis (a Linear, Rotary, or Spindle axis).

Italicized information enclosed in brackets [*italicized*] indicates the type and name of a parameter that can be located in Appendix C (i.e., [*Machine/Type*] – Type is a Machine parameter, more information can be found on Type in Appendix C).

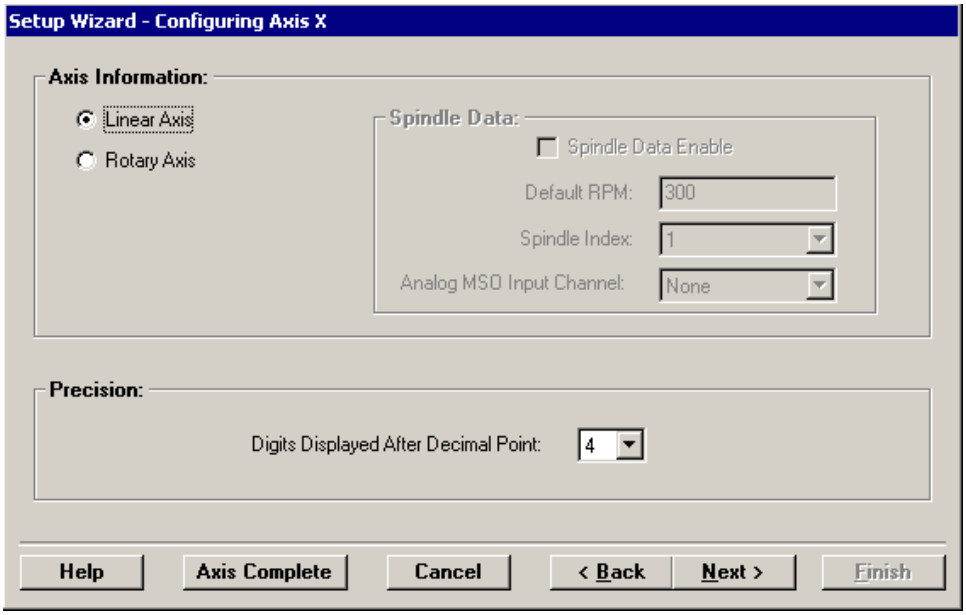


Figure 12-3. The Axis "Type" Configuration Screen

If it is a Spindle Axis:

What is the default RPM [*Task/S1_RPM*] that the spindle is to run at?

Which of the four spindles [*Task/S1_Index*] assigned to this task is it?

Is there a MSO/MFO [*Task/MSO* or *MFO*] control?

How many digits are displayed to the right of the decimal point (dependent on the mode that you have selected) in:

English mode [*Machine/NumDecimalsEnglish*]?

Metric mode [*Machine/NumDecimals/Metric*]?

12.4. Axis Configuration

If the axis configuration shown is correct, select the 'Next' button, otherwise, select the 'Reconfigure Axis' button (see Section 12.4.1.). Selecting 'Reconfigure Axis' will open the Axis Configuration Wizard.

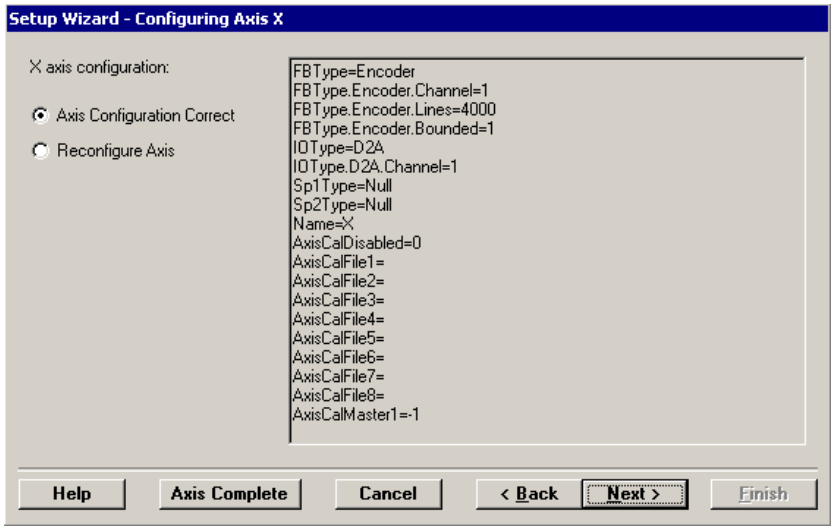


Figure 12-4. The Axis Configuration "Correct or Reconfigure" Screen

12.4.1. Axis Configuration Wizard

The Axis Configuration Wizard will guide you through the process of configuring the axis. UNIDEX 600 Series Controllers support any motor accepting a velocity or torque command and providing feedback from a supported feedback device (such as an encoder, resolver, or inductosyn).

Figure 12-5. The Axis Configuration Wizard Welcome Screen

'Next' will advance you to the next Wizard configuration screen. 'Cancel' will exit the Wizard without saving any changes to the Axis configuration.

12.4.2. Axis & Parameter Names and Task Number Configuration

The second Wizard screen (Figure 12-6) allows axis names to be assigned, the task axis to be assigned, and the axis to be bound to task 1 through 4. Most users will bind all axes to task 1. Also, parameter names may be associated with each axis that is not bound to a task.

If you reassign an axis name, you must exit the MMI600 and restart it for the change to take effect.

Figure 12-6. The Axis Configuration Wizard – Setup Name Screen

Selecting 'Next' will advance you to the next Wizard configuration screen, 'Back' will take you to the previous Wizard screen, 'Cancel' will exit the Wizard without saving any changes to the axis configuration. 'Finish' will save the axis configuration and exit the Wizard.

12.4.3. Configuring Axis Type

The upper list box shows the predefined axis configuration templates available to configure the axis that you have selected. By default, the current axis is selected. You may select a previously configured axis or one of the pre-configured axis types from the list box to use as a template to configure the current axis. Also, you may leave the current axis selected and create a new axis configuration by making a selection from the 3 lower drop-down boxes (Primary/Secondary Feedback, Command Output – see Figure 12-7).

There are predefined axis types for configuring your axes quickly from a template. Selecting a template from the list box will use that axis configuration for the current axis and allow you to modify the configuration as required. All channel assignments (D/A, encoder, resolver, etc.) will default to the number of the physical axis, not the channel assigned to the axis selected from the template. The current axis configuration is displayed below the template list box. Selecting a template will update the axis configuration.

The axis must be configured for a supported feedback type present on the axis. This involves assigning a primary feedback device, which always provides the position feedback. The Command Output may be either Null or D2A (D/A converter).

Optionally, an axis may have a secondary feedback device.

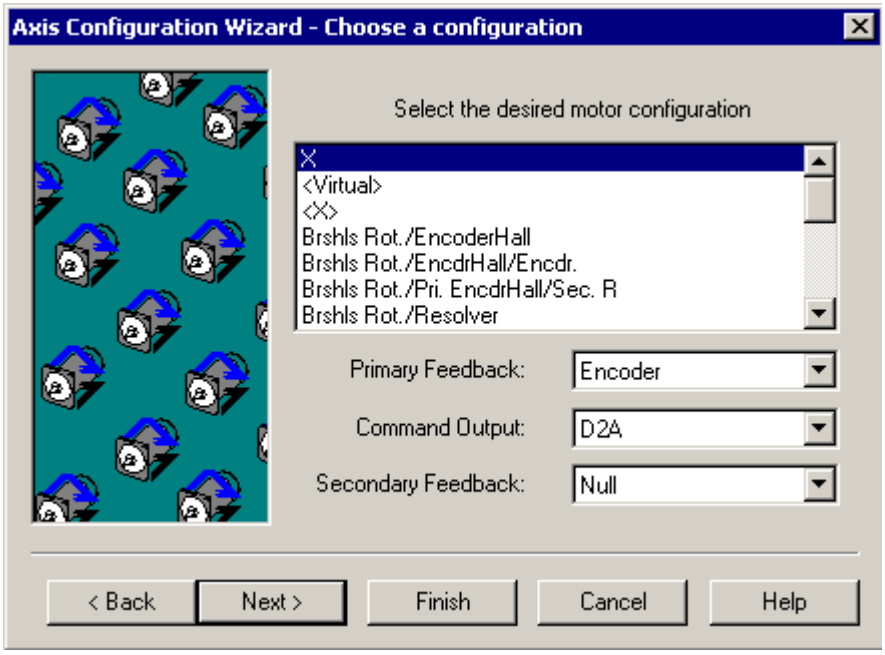


Figure 12-7. The Axis Configuration Wizard – Choose a Configuration Screen

The axis may be configured as a Null of Virtual axis for debugging purposes.

Selecting 'Next' will advance you to the next Wizard configuration screen, 'Back' will take you to the previous Wizard screen, 'Cancel' will exit the Wizard without saving any changes to the axis configuration. 'Finish' will save the axis configuration and exit the Wizard.

12.4.3.1. Predefined Axis Types

The predefined axis types (templates), defined in the \U600\Ini\AxisCfg.Wiz file, for use by the Axis Configuration Wizard are:

- Virtual (Null) Axis (for test purposes)
- BM Series Brushless Rotary Motor with EncoderHall Feedback
- BM Series Brushless Rotary Motor with EncoderHall Feedback and another Encoder as Secondary Feedback
- BM Series Brushless Rotary Motor with EncoderHall Feedback and a Resolver as Secondary Feedback
- BM Series Brushless Rotary Motor with Resolver Feedback
- BLMx Series Brushless Linear Motor with EncoderHall Feedback
- BLMx Series Brushless Linear Motor with InductosynHall Feedback
- DC Brush Rotary Motor with Encoder Feedback
- DC Brush Rotary Motor with Resolver Feedback
- DC Brush Rotary Motor with Encoder Feedback and another Encoder as Secondary Feedback
- DC Brush Rotary Motor with Encoder Feedback and a Resolver as Secondary Feedback
- Spindle Axis (Velocity Command, Open Loop, with no feedback)
- Stepper Motors (Open Loop)
- Stepper Motors (Closed Loop)

12.4.4. Configuring the Primary Feedback Device

The third Axis Wizard Configuration screen allows the primary feedback device to be configured. This screen will vary depending on the primary feedback device selected (so it is not shown here). Pressing the 'Help' button on the bottom of the screen will clarify the required entries. The primary feedback device is used for position and velocity feedback, unless there is a secondary feedback device. When there is a secondary feedback device, the primary feedback device is always used for position feedback only.

Encoder Configuration – Section 12.4.6.1. on page 12-11.

EncoderHall Configuration – Section 12.4.6.2. on page 12-11.

EncoderHall (Pole Pairs) – Section 12.4.6.3. on page 12-13.

Resolver Configuration (or Inductosyn) – Section 12.4.6.4. on page 12-15.

ResolverHall Configuration (or Inductosyn) – Section 12.4.6.5. on page 12-16.

Stepper Motor Configuration – Section 12.4.6.6. on page 12-17.

Null (Virtual) Configuration – Section 12.4.6.7. on page 12-18.

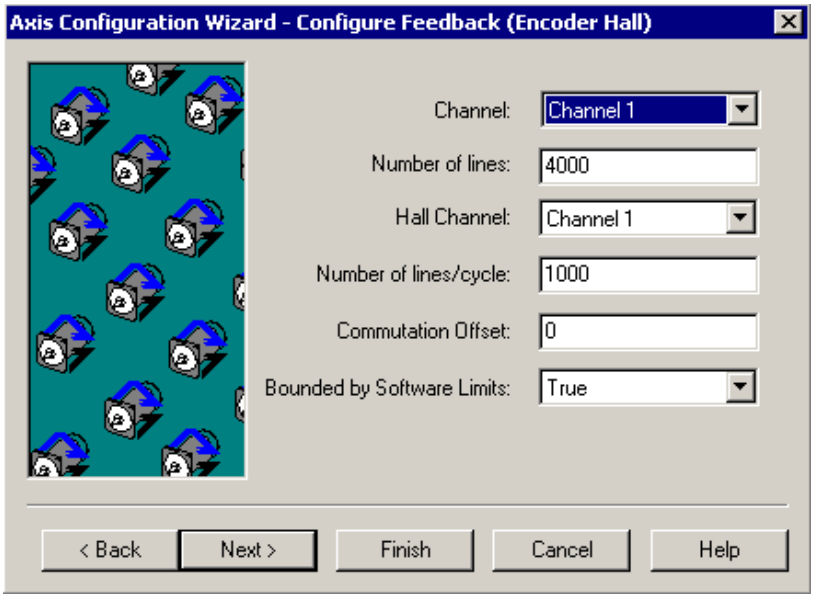


Figure 12-8. The Axis Configuration Wizard – Primary Feedback Screen

Selecting 'Next' will advance you to the next Wizard configuration screen, 'Back' will take you to the previous Wizard screen, 'Cancel' will exit the Wizard without saving any changes to the axis configuration. 'Finish' will save the axis configuration and exit the Wizard.

12.4.5. Configuring a DAC Channel

The D/A (D2A, DAC, or Digital to Analog Converter) channel number must be specified for all axes (except virtual), to provide the command (velocity or torque) to the driver module.

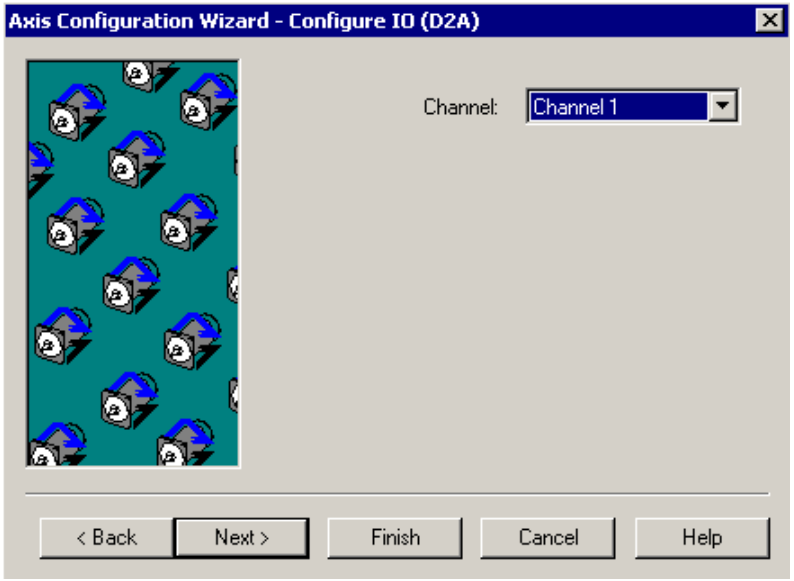


Figure 12-9. The Axis Configuration Wizard – D2A Screen

The DAC channel number defines the channel for the axis specific I/O (CW/CCW/Home Limits, encoder fault, drive fault, Auxiliary (Mode) output and the drive enable) for axes with resolver or resolverhall feedback. Axes with encoder or encoderhall feedback have their I/O associated with the encoder channel number specified.

Selecting 'Next' will advance you to the next Wizard configuration screen, 'Back' will take you to the previous Wizard screen, 'Cancel' will exit the Wizard without saving any changes to the axis configuration. 'Finish' will save the axis configuration and exit the Wizard.

12.4.6. Configuring the Secondary Feedback Device

The Axis Wizard Configuration screen allows the secondary feedback device to be configured. This screen will vary depending on the secondary feedback device selected, so it is not shown here (pressing the ' Help' button on the bottom of the screen will clarify the required entries). The secondary feedback device is always used for velocity feedback. When there is no secondary feedback device, the primary feedback device is always used for position feedback. If you have no secondary feedback device, this screen will not be displayed (you will be on the Axis Calibration Configuration screen).

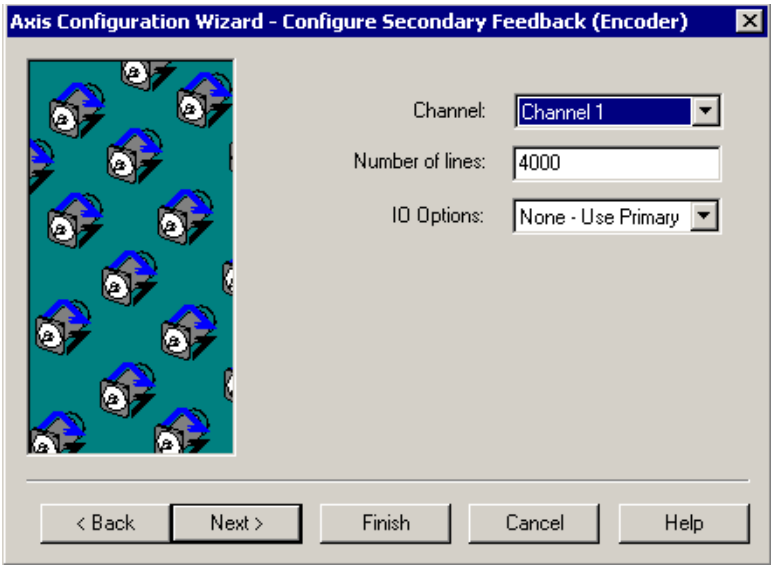


Figure 12-10. The Axis Configuration Wizard – Secondary Feedback Screen

Selecting 'Next' will advance you to the next Wizard configuration screen, 'Back' will take you to the previous Wizard screen, 'Cancel' will exit the Wizard without saving any changes to the axis configuration. 'Finish' will save the axis configuration and exit the Wizard.

12.4.6.1. Encoder Configuration

Channel Number

The channel number specifies the channel number that the encoder feedback device will be read from for this axis, as well as the specific I/O (CW/CCW/Home Limits, encoder fault, drive fault, Auxiliary (Mode) output, and the drive enable).

Encoder Channel Assignments
Channels 1 through 4 are on the UNIDEX 600 card, channels 5 through 8 are on the 4EN-PC card configured as Board 1, channels 9 through 12 are on the 4EN-PC card configured as Board 2, channels 13 through 16 are on the 4EN-PC card configured as Board 3.

Number of Lines

The number of lines for the encoder must be specified. This may also be used to rescale the PGAIN axis parameter.

Rotary Encoder
For rotary encoders, enter the number of lines per revolution of the encoder, after the times 4 multiplication is done by the controller (i.e., for a 1000 line encoder, enter 4000).

Linear Encoder
For brush motors with linear encoders, enter the number of counts seen by the controller per revolution of the motor after the times 4 multiplication is done by the controller (i.e., a ball-screw with a pitch of .1inch, having a linear encoder with 1,270,000 counts per inch (after x4 multiplication), would have 127,000 entered for the number of lines (1,270,000 * .1 = 127,000).

For brushless motors with linear encoders, enter the number of lines per user unit (inch or millimeter) after the x4 multiplication is done by the controller (i.e., 1000 lines per inch would be entered as 4000).

Bounded by Software Limits

The Bounded by Software Limits field within the Axis Configuration Wizard is used to specify if software limits are to be activated for the axis (true or false may be selected). The software limits are defined by the CWEOT and CCWEOT axis parameters.

12.4.6.2. EncoderHall Configuration

Channel Number

The channel number specifies the channel number that the encoder feedback device will be read from for this axis, as well as the specific I/O (CW/CCW/Home Limits, encoder fault, drive fault, Auxiliary (Mode) output, and the drive enable).

Encoder Channel Assignments
Channels 1 through 4 are on the UNIDEX 600 card, channels 5 through 8 are on the 4EN-PC card configured as Board 1, channels 9 through 12 are on the 4EN-PC card configured as Board 2, channels 13 through 16 are on the 4EN-PC card configured as Board 3.

Number of Lines

The number of lines for the encoder must be specified. This may also be used to rescale the PGAIN axis parameter.

Rotary Encoder
For rotary encoders, enter the number of lines per revolution of the encoder, after the times 4 multiplication is done by the controller (i.e., for a 1000 line encoder, enter 4000).

Linear Encoder
For brush motors with linear encoders, enter the number of counts seen by the controller per revolution of the motor after the times 4 multiplication is done by the controller (i.e., a ball-screw with a pitch of .1inch, having a linear encoder with 1,270,000 counts per inch (after x4 multiplication), would have 127,000 entered for the number of lines (1,270,000 * .1 = 127,000).

For brushless motors with linear encoders, enter the number of lines per user unit (inch or millimeter) after the x4 multiplication is done by the controller (i.e., 1000 lines per inch would be entered as 4000).

Commutation Channel (Hall Channel)

EncoderHall
The commutation channel number specifies the channel number used to commutate the motor. The Hall effect sensors determine the absolute rotor position, and then the encoder commutates the motor.

The channel number specified indicates the Hall effect channel and the encoder channel, which will be used to commutate the motor, after switching out of the six step commutation mode.

EncoderHall Channel Assignments
Channels 1 through 4 are on the U600 card, channels 5 - 8 are on the 4EN-PC card configured as Board 1, channels 9 - 12 are on the 4EN-PC card configured as Board 2, channels 13 - 16 are on the 4EN-PC card configured as Board 3.

Brushless Motors w/o Hall Effect Feedback Signals
Brushless motors may be commutated by the controller even if Hall effect feedback signals are not present. To do so, configure the axis as though Hall effect signals are present, then:

Use the MSET command to align the absolute rotor position (vector)
Dwell 1 second
ENABLE the axis
This may be done automatically through a Canned Function.

ResolverHall
A commutation channel number must be specified for the Hall effect sensors to be read. This provides the absolute rotor position for initializing and commutating the motor. After rotor initialization it will then commutate the motor by the primary feedback device, unless a secondary feedback device is present.

Resolver Channel Assignments
Channels 1 - 4 are on the resolver card (RDP-PC) configured as Board 1. Channels 5 - 8 are on the RDP-PC card configured as Board 2. Channels 9 - 12 are on the RDP-PC card configured as Board 3, and channels 13 - 16 are on the RDP-PC card configured as Board 4.

Number of Lines/Electrical Cycle

The number of lines/cycle field specifies the number of lines the encoder (after x4 multiplication by the controller) that are equal to one electrical cycle of the motor. This value must be an integer, although no cumulative commutation error will occur. The maximum number of lines/cycle permitted is 2,147,483,647. The motor may also be commutated by Six Step commutation. For motors with an odd number of electrical cycles, configure the motor as EncoderHall, instead.

The number of lines specified is relative to the encoder channel number specified by the Commutation Channel, typically the velocity feedback device.

Rotary Motors
This is set equal to the number of encoder counts per revolution of the motor (after x4 multiplication), divided by the number of electrical cycles per revolution of the motor (number of poles (pairs)).

Linear Motors
This is set equal to the number of encoder counts (after x4 multiplication) per electrical cycle of the forcer (motor).

Commutation Offset

The commutation offset indicates the number of electrical degrees to align the absolute rotor reference (provided by the commutation channel) to the rotor of the motor. The offset is entered as counts ratioed to 1024 and may be positive or negative. A 360° offset is equal to 1024.

A 10° offset may be calculated as:

10 / 360 * 1024 = 28

Bounded by Software Limits

The Bounded by Software Limits field within the Axis Configuration Wizard is used to specify if software limits are to be activated for the axis (true or false may be selected). The software limits are defined by the CWEOT and CCWEOT axis parameters.

12.4.6.3. EncoderHall (Pole Pairs) Configuration

Channel Number

The channel number specifies the channel number that the encoder feedback device will be read from for this axis, as well as the specific I/O (CW/CCW/Home Limits, encoder fault, drive fault, Auxiliary (Mode) output, and the drive enable).

Encoder Channel Assignments
Channels 1 through 4 are on the UNIDEX 600 card, channels 5 through 8 are on the 4EN-PC card configured as Board 1, channels 9 through 12 are on the 4EN-PC card configured as Board 2, channels 13 through 16 are on the 4EN-PC card configured as Board 3.

Number of Lines

The number of lines for the encoder must be specified. This may also be used to rescale the PGAIN axis parameter.

Rotary Encoder
For rotary encoders, enter the number of lines per revolution of the encoder, after the times 4 multiplication is done by the controller (i.e., for a 1000 line encoder, enter 4000).

Linear Encoder
For brush motors with linear encoders, enter the number of counts seen by the controller per revolution of the motor after the times 4 multiplication is done by the controller (i.e., a ball-screw with a pitch of .1inch, having a linear encoder with 1,270,000 counts per inch (after x4 multiplication), would have 127,000 entered for the number of lines (1,270,000 * .1 = 127,000).

For brushless motors with linear encoders, enter the number of lines per user unit (inch or millimeter) after the x4 multiplication is done by the controller (i.e., 1000 lines per inch would be entered as 4000).

Commutation Channel (Hall Channel)

EncoderHall
The commutation channel number specifies the channel number used to commutate the motor. The Hall effect sensors determine the absolute rotor position, and then the encoder commutates the motor.

The channel number specified indicates the Hall effect channel and the encoder channel, which will be used to commutate the motor, after switching out of the six step commutation mode.

EncoderHall Channel Assignments
Channels 1 through 4 are on the U600 card, channels 5 - 8 are on the 4EN-PC card configured as Board 1, channels 9 - 12 are on the 4EN-PC card configured as Board 2, channels 13 - 16 are on the 4EN-PC card configured as Board 3.

Brushless Motors w/o Hall Effect Feedback Signals
Brushless motors may be commutated by the controller even if Hall effect feedback signals are not present. To do so, configure the axis as though Hall effect signals are present, then:

Use the MSET command to align the absolute rotor position (vector)
Dwell 1 second
ENABLE the axis
This may be done automatically through a Canned Function.

ResolverHall
A commutation channel number must be specified for the Hall effect sensors to be read. This provides the absolute rotor position for initializing and commutating the motor. After rotor initialization it will then commutate the motor by the primary feedback device, unless a secondary feedback device is present.

Resolver Channel Assignments
Channels 1 - 4 are on the resolver card (RDP-PC) configured as Board 1. Channels 5 - 8 are on the RDP-PC card configured as Board 2. Channels 9 - 12 are on the RDP-PC card configured as Board 3, and channels 13 - 16 are on the RDP-PC card configured as Board 4.

Lines/Revolution (Lines/Electrical Cycle)

The number of lines for the encoder must be specified as an integer. This field is NOT used for re-scaling the PGAIN axis parameter (use the Number of Lines field for that purpose).

Rotary Encoders
For rotary encoders enter the number of lines per revolution of the encoder (after x4 multiplication). For example, for a 1000 line encoder, enter 4000.

Linear Encoders
For brushless motors with linear encoders, enter the number of lines equeal to the specified number of electrical cycles (number of Pole Pairs) after x4 multiplication is done by the controller. For example, 1000 lines per Number of Pole Pairs would be entered as 4000. Assume there were 123.33333 counts per electrical cycle on a linear motor. Set the lines/revolution to 370 and Pole Pairs to 3 (370 =123.33333 * 3).

Number of Pole Pairs

The number of Pole Pairs is the number of permanent magnetic poles, expressed as pole pairs, that an Aerotech motor has.

Commutation Offset

The commutation offset indicates the number of electrical degrees to align the absolute rotor reference (provided by the commutation channel) to the rotor of the motor. The offset is entered as counts ratioed to 1024 and may be positive or negative. A 360° offset is equal to 1024.

A 10° offset may be calculated as:

10 / 360 * 1024 = 28

Bounded by Software Limits

The Bounded by Software Limits field within the Axis Configuration Wizard is used to specify if software limits are to be activated for the axis (true or false may be selected). The software limits are defined by the CWEOT and CCWEOT axis parameters.

12.4.6.4. Resolver Configuration (or Inductosyn)

Channel Number

The channel number specifies the channel number that the encoder feedback device will be read from for this axis, as well as the specific I/O (CW/CCW/Home Limits, encoder fault, drive fault, Auxiliary (Mode) output, and the drive enable).

Encoder Channel Assignments
Channels 1 through 4 are on the UNIDEX 600 card, channels 5 through 8 are on the 4EN-PC card configured as Board 1, channels 9 through 12 are on the 4EN-PC card configured as Board 2, channels 13 through 16 are on the 4EN-PC card configured as Board 3.

Resolution

The resolution field specifies the machine steps per electrical cycle of the resolver (or inductosyn). This is entered as 10, 12, 14, or 16 (bits), which represent 1024, 4096, 16,384, and 65,536 machine counts per electrical cycle of the feedback device, respectively.

Poles

The poles field of the resolver screen within the Axis Configuration Wizard specifies the number of total poles (NOT pole pairs) the motor contains. A non-commutated or DC brush motor has zero poles.

Commutation Offset

The commutation offset indicates the number of electrical degrees to align the absolute rotor reference (provided by the commutation channel) to the rotor of the motor. The offset is entered as counts ratioed to 1024 and may be positive or negative. A 360° offset is equal to 1024.

A 10° offset may be calculated as:

10 / 360 * 1024 = 28

Bounded by Software Limits

The Bounded by Software Limits field within the Axis Configuration Wizard is used to specify if software limits are to be activated for the axis (true or false may be selected). The software limits are defined by the CWEOT and CCWEOT axis parameters.

12.4.6.5. ResolverHall Configuration

Channel Number

The channel number specifies the channel number that the encoder feedback device will be read from for this axis, as well as the specific I/O (CW/CCW/Home Limits, encoder fault, drive fault, Auxiliary (Mode) output, and the drive enable).

Encoder Channel Assignments
Channels 1 through 4 are on the UNIDEX 600 card, channels 5 through 8 are on the 4EN-PC card configured as Board 1, channels 9 through 12 are on the 4EN-PC card configured as Board 2, channels 13 through 16 are on the 4EN-PC card configured as Board 3.

Resolution

The resolution field specifies the machine steps per electrical cycle of the resolver (or inductosyn). This is entered as 10, 12, 14, or 16 (bits), which represent 1024, 4096, 16,384, and 65,536 machine counts per electrical cycle of the feedback device, respectively.

Number of Lines/Cycle

The number of lines/cycle field specifies the number of resolver counts that are equal to one electrical cycle of the motor. This parameter is always entered as though the feedback device is in 16 bit (65,536) mode. The electrical cycle of the motor is ratioed to one cycle of the feedback device, as though the feedback device is 65,536 counts per electrical cycle of the feedback device.

Rotary Encoder
This is set equal to the number of resolver counts per revolution of the motor, divided by the number of electrical cycles per revolution of the motor (number of poles (pairs)).

Linear Encoder
This is set equal to the number of resolver counts per electrical cycle of the forcer (motor).

Poles

The poles field of the resolver screen within the Axis Configuration Wizard specifies the number of total poles (NOT pole pairs) the motor contains. A non-commutated or DC brush motor has zero poles.

Commutation Offset

The commutation offset indicates the number of electrical degrees to align the absolute rotor reference (provided by the commutation channel) to the rotor of the motor. The offset is entered as counts ratioed to 1024 and may be positive or negative. A 360° offset is equal to 1024.

A 10° offset may be calculated as:

10 / 360 * 1024 = 28

Bounded by Software Limits

The Bounded by Software Limits field within the Axis Configuration Wizard is used to specify if software limits are to be activated for the axis (true or false may be selected). The software limits are defined by the CWEOT and CCWEOT axis parameters.

12.4.6.6. Stepper Motor Configuration

Stepper axes may be open loop or closed loop. Closed loop axes must have their servo loop gains adjusted.

For closed loop axes, Ki and Kp must be set to zero, and the PGAIN and VGAIN axis parameters will control the servo loop. Increasing or decreasing the number of encoder lines will scale the PGAIN axis parameters.

Channel Number

The channel entry field is used to specify the channel number channel number containing the CW/CCW and Home Limits, as well as the number the encoder feedback will be read from (if it is a closed loop axis).

Encoder Channel Assignments
Channels 1 through 4 are on the UNIDEX 600 card, channels 5 - 8 are on the 4EN-PC card configured as Board 1, channels 9 - 12 are on the 4EN-PC card configured as Board 2, channels 13 - 16 are on the 4EN-PC card configured as Board 3.

Number of Lines

The lines field is set to zero for open loop stepper axes, or the encoder liner per revolution. The number of lines for the encoder must be specified. This may also be used to re-scale the PGAIN axis parameter.

Rotary Encoder
For rotary encoders, enter the number of lines per revolution of the encoder after the x4 multiplication is done by the controller (i.e., for a 1000 line encoder, enter 4000).

Linear Encoder
For motors with linear encoders, enter the number of counts seen by the controller per revolution of the motor after the x4 multiplication is done by the controller (i.e., a ball-screw with a pitch of .1inch, having a linear encoder with 1,270,000 counts per inch (after x4 multiplication), would have 127,000 entered for the number of lines (1,270,000 * .1 = 127,000).

Stepper Channel

This field specifies the clock and direction channel number for the axis. The 4 channels on the U600 board are channels 1 through 4. Then encoder expansion cards (4EN-PC) do not support stepper axes.

Bounded by Software Limits

The Bounded by Software Limits field within the Axis Configuration Wizard is used to specify if software limits are to be activated for the axis (true or false may be selected). The software limits are defined by the CWEOT and CCWEOT axis parameters.

If a drive enable signal is required for the stepper axis, specify a D2A (DAC) for the Command Output so that a DAC output may be specified. The drive enable associated with this DAC channel may then be used as the drive enable for the stepper axis.

12.4.6.7. Null (Virtual) Configuration

A Null (or Virtual) secondary feedback device requires no parameters to be entered. This is the default configuration.

An axis may be configured as Virtual or NULL by setting the Primary, Command Output and Secondary feedback devices to Null. In virtual axes, the servo loop is bypassed, and the Position and Velocity feedback are instead set equal to the Position and Velocity Commands, respectively. The tracking displays for a virtual axis will display position and velocity as though it were a real axis, except that the Position Error and Velocity Error will always be zero. This will facilitate debugging of a CNC motion program when no drives are physically present. However, even if drives and feedback devices are present, you may configure the axis as virtual. In this case, no torque will be commanded to the drive, and no feedback will be read from the feedback device. But when drives and feedback are properly connected, normally the SIMULATION parameter is a more convenient way to debug programs, since you won't need to restore the actual axis configuration after debugging.

When an axis is configured virtual, the IOLEVEL axis parameter will be automatically set to 63, to avoid axis faults from being generated.

If a virtual axis is homed, it will immediately set the current position to the value of the HomeOffsetInch (or HomeOffsetDeg) task parameter, rather than simulating any motion. Axis calibration, hardware and software limits have no effect on virtual axes.

12.4.6.8. Configuring Dual Loop Axes

The Hall effect, limits, encoder fault and drive fault inputs are always read from the same channel. The axis status words are derived from a direct read of this data as part of the background task. The servo loop reads the hall inputs directly because they need to have minimal latency for motor commutation. This implies that the servo loop can (and does in this instance) look at a different source for the hall inputs than does the axis status word. The use must be aware of this when configuring axes.

Dual loop encoder axes MUST be configured as follows:

1. The Hall effect inputs, D/A, limits, and the secondary feedback channel must be assigned to the same channel number (which must be channels 1-8 and the option to use velocity limits must be selected). For example, if the velocity encoder uses channel 2, then the D/A should also use channel 2. You must specify the Number of Lines \ Electrical Cycle relative to the secondary feedback device, even though the value is entered on the screen for the primary feedback device.

2. The primary feedback can be connected to any available encoder channel and this connection contains only the encoder and marker information.

In this configuration all of the I/O is coming back on the same physical channel for both the drive and encoder interface and as a result the status word will display the hall effect feedback correctly and you will not have spurious drive fault errors.

12.4.7. Configuring Axis Calibration Data

This screen, within the Axis Configuration Wizard, allows ASCII axis calibration files to be specified for the axis. 2D calibration files are specified on the main setup page of the MMI600. Axis calibration files contain a look-up table of absolute positions and correction values at those absolute positions. Multiple calibration files (up to 8) may be simultaneously active per axis.

Figure 12-11. The Axis Configuration Wizard – Set Axis Cal. File Screen

Selecting the 'Add' button will allow an axis calibration file to be specified for the axis. This will cause a dialog box to be displayed allowing a file to be specified. The path and file name may be manually entered, or you may select the '…' (Browse) button to find the desired file.

The Master Axis specifies the axis whose absolute positions are used as the index into the look-up table to produce the correction values for this axis. The master axis is normally the current axis and will be the default master axis number. Another axis may be used as the master axis, such as for orthogonality correction.

Selecting 'OK' will add the calibration file to the axis configuration

Selecting 'Cancel' will return you to the main axis calibration screen.

Selecting 'Update' allows the highlighted axis calibration file entry to be modified. The master axis may be changed or a new file may be specified.

Selecting 'Delete' will remove the highlighted axis calibration file entry.

The Disable Axis Calibration flag may be used to disable the use of all axis calibration files specified on this screen (Figure 12-11).

12.4.8. Saving an Axis Configuration

The final Axis Configuration screen will display a summary of the configuration of the axis. If 'Finish' is selected, the axis will then be configured on the controller and the configuration will be written to the file defined on the main Setup page of the MMI600 (C:\u600\Ini\AxisCfg.ini by default).

Figure 12-12. The Axis Configuration Wizard – Save/Finish Screen

Selecting 'Back' will take you the previous Wizard screen. 'Next' will advance you to the next Wizard Configuration screen. 'Cancel' will exit the Wizard without saving any changes to the configuration, and 'Finish' will save the axis configuration and exit the Wizard.

12.5. Scaling and Feedrates

Enter the machine counts per user unit (after the controller does x4 multiplication) for the axis.

Linear Axes [*Machine/CentsPerInch*]

Rotary Axes [*Machine/CentsPerDeg*]

Spindle Axes [*Machine/CentsPerDeg*]

Enter the maximum feedrate that may be commanded for the axis.

Linear Axes [*Machine/MaxFeedrateIPM*]

Rotary Axes [*Machine/MaxFeedrateRPM*]

Spindle Axes [*Machine/MaxFeedrateRPM*]

Enter the G0 rapid feedrate that may be commanded for the axis.

Linear Axes [*Machine/RapidFeedrateIPM*]

Rotary Axes [*Machine/RapidFeedrateRPM*]

Spindle Axes [*Machine/RapidFeedrateRPM*]

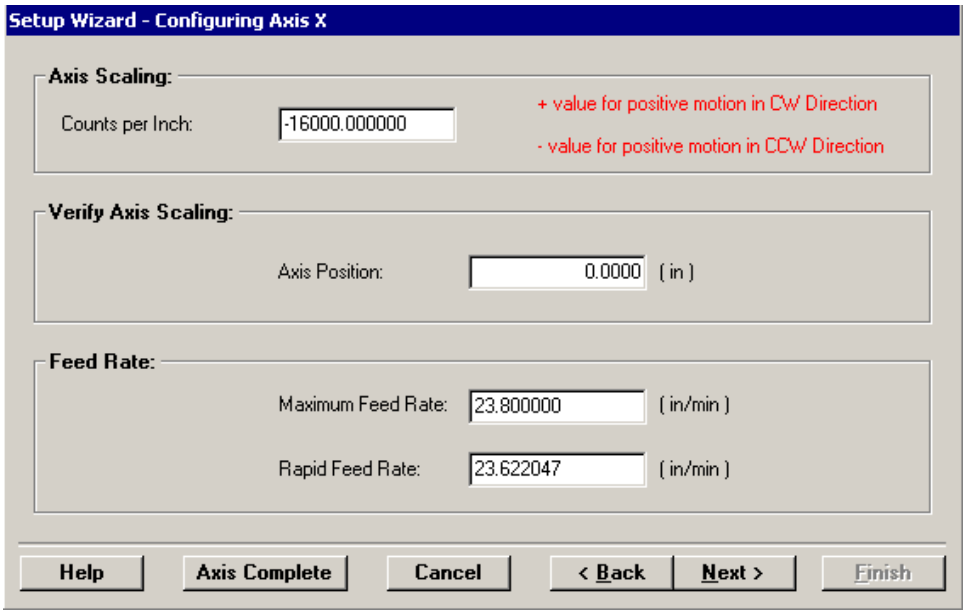


Figure 12-13. The Scaling and Feedrate Screen

12.6. Home Cycle Configuration

An axis must have an absolute reference (0,0) point, when the system is powered up. An axes reference position is located by homing the axis, via the Home command, Jog Page or the Home Tab of the Run Page. As each axis reaches its reference position, its position registers are set equal to the value in that axes' HomeOffsetInch (HomeOffsetDeg for rotary axes) machine parameter. Operators may be required to home axes and/or a homing sequence may be defined via the Home Setup on the MMI Options Page of the Setup page of the MMI600.

The following parameters must be defined for the Home Cylce.

1. Select the Type of Home Cycle [*Machine/HomeType*]
2. Select the Home Direction [*Machine/HomeDirection*]
3. Enter the Home Feed Rate for:

   Linear Axes [*Machine/HomeFeedrateIPM*]

   Rotary Axes [*Machine/HomeFeedrateRPM*]

   Spindle Axes [*Machine/HomeFeedrateRPM*]
4. Enter a Home Offset for (optional):

   Linear Axes [*Machine/HomeOffsetInch*]

   Rotary Axes [*Machine/HomeOffsetDeg*]

   Spindle Axes [*Machine/HomeOffsetDeg*]

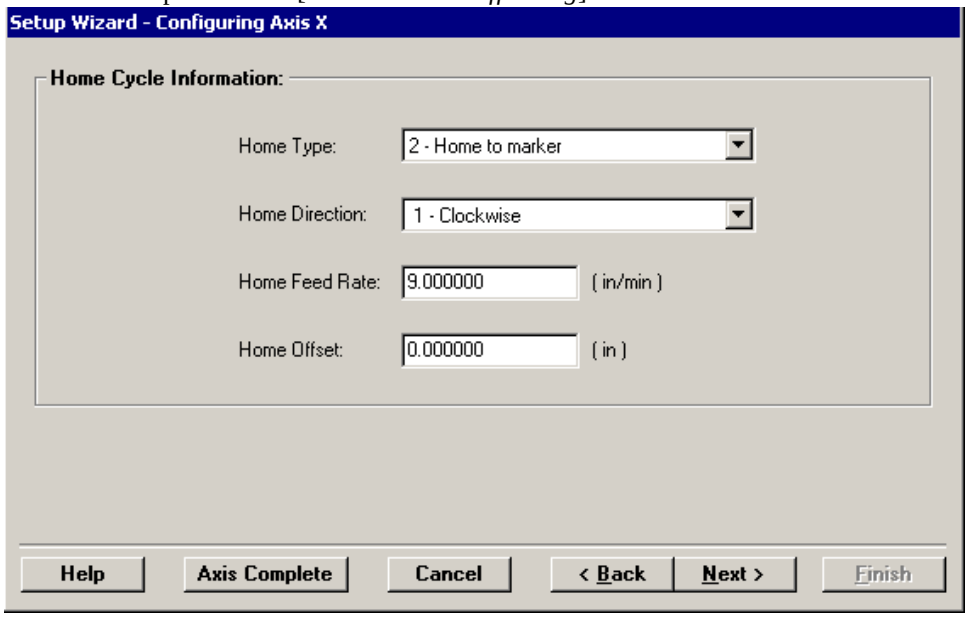


Figure 12-14. The Home Cycle Screen

If the HomeOffsetInch machine parameter is set to a non-zero value with axis calibration active, the axis may not stop on the marker at the completion of the home cycle. If it desired to do so, for a positioning test, you must temporarily disable axis calibration, within the axis configuration wizard.

If a virtual axis is homed, it will immediately set the position to the home position, rather than simulating any motion.

Homing will cancel all fixture offsets and presets.

When using the CNC (MMI600), the HomeOffset task parameter is used. However, for Library invoked homing, the HOMEOFFSET axis parameter is used. If using both CNC and Library interfaces simultaneously, the user must use both of these parameters.

If an axis is in the Simulation, Dry Run, or Machine Lock modes when it is homed, the home command will never complete, because the axis does not move in these modes.

Homing will disable normalcy, cutter offset and cutter radius compensation modes.

12.7. Asynchronous and G0 Accel/Decel Parameters

1. Select the desired Acceleration Mode [*Axis/ACCELMODE*]
2. Enter the desired Acceleration Rate [*Axis/ACCELRATE*]
3. Enter the desired Acceleration Time [*Axis/ACCEL*]
4. Select the desired Deceleration Mode [*Axis/DECELMODE*]
5. Enter the Deceleration Rate [*Axis/DECELRATE*]
6. Enter the desired Deceleration Time [*Axis/DECEL*]

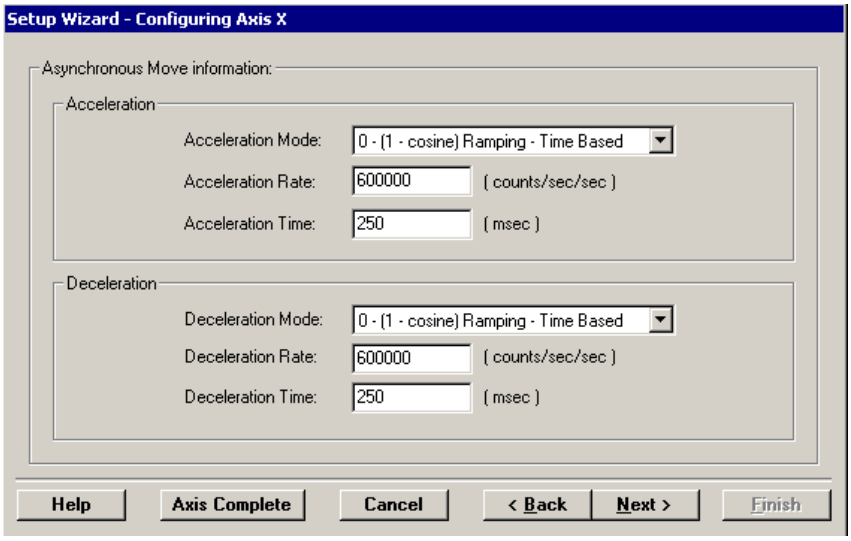


Figure 12-15. The Asynchronous Move Screen

12.8. Position Limits and Velocity Trap

Enter the desired Position Error Limit (in user units) [Axis/POSERRLIMIT]
Enter the In Position Error Limit (in user units) [Axis/INPOSLIMIT]
Enter the Velocity Trap (in user units)

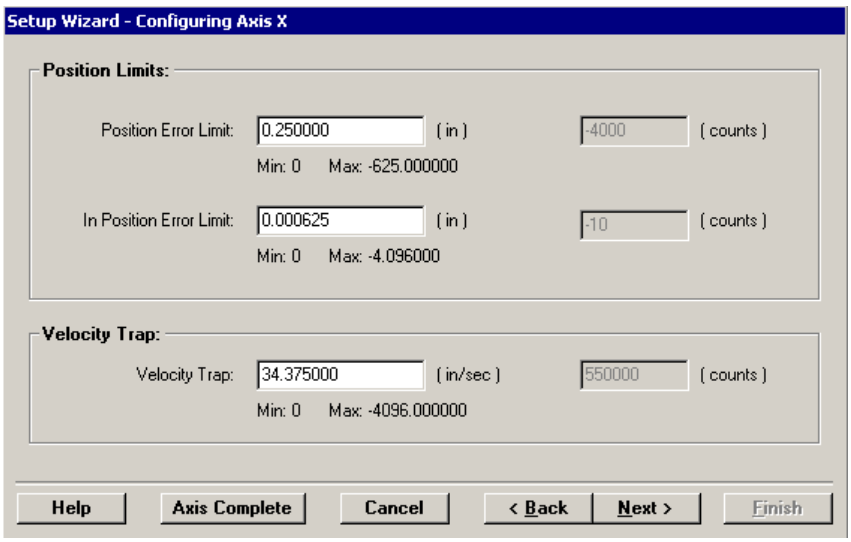


Figure 12-16. Position Limits and Velocity Trap Screen

12.9. Configure the Drive Interface States

Specify the active state for the axis and drive interface signals. The user may configure the active state of the following signals.

0.	Drive Shutdown	(output to drive)	0x1
1.	AUX (Mode) Output	(output to drive)	0x2
2.	CW limit Switch	(input from drive)	0x4
3.	CCW limit Switch	(input from drive)	0x8
4.	Home limit Switch	(input from drive)	0x10
5.	Drive Fault	(input from drive)	0x20

The value specified is a bit-mask where only the specified bits are valid. Setting a bit to one implies the input or output is active high. Refer to Section 2.5. Drive Signals in the Users Guide for more information.

The easiest way to configure these signals via the IOLEVEL axis parameter is to view the state of the signals via the AerStat.exe utility. Knowing the state of the signal and viewing the state via AerStat will allow you to toggle the appropriate bits in the IOLEVEL axis parameter to correct the state.

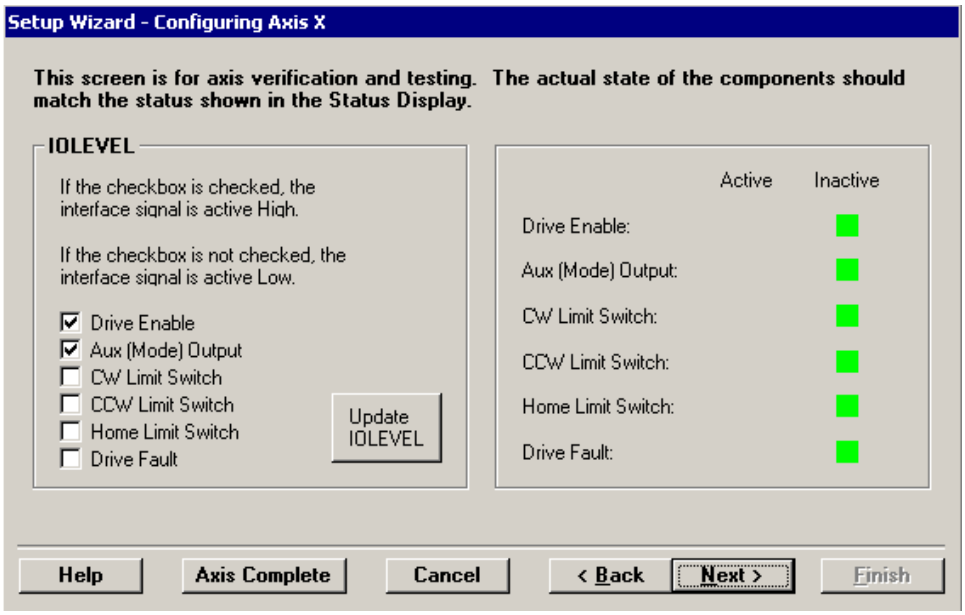


Figure 12-17. The Drive Interface Configuration Screen (via the IOLEVEL)

Be sure to set the CW, CCW, and Drive Fault bits in the FAULTMASK axis parameter to enable the detection of these faults, then set the bits in the appropriate mask parameters (DISABLEMASK, HALTMASK, AUXMASK, ABORTMASK, INTMASK, and BRAKEMASK) for actions to occur on these faults.

12.10. Configure the FAULTMASK

This axis parameter determines which faults the system will detect. The parameter is a bit mask where each bit corresponds to a specific fault. Setting a bit to a one enables monitoring of the fault assigned to that bit. Conversely, clearing a bit causes the system to ignore that fault if it occurs. If a fault is detected, it's bit value is "anded" into the FAULT axis parameter value.

Each bit set in this parameter should have a bit set in at least one of the other mask parameters (DISABLEMASK, HALTMASK, AUXMASK, ABORTMASK, INTMASK and BRAKEMASK) to define the action to occur for that fault. If you set a bit in the FAULTMASK, but fail to set any corresponding bits in one of the other masks listed above, then the FAULT parameter will be set, but no axis action will occur. These actions will occur immediately after detection of the fault, usually within a millisecond.

You can also trigger program-related actions to take place when an axis fault occurs, with the TaskFault task parameter. CNC programs may be stopped when by axis faults, via the HaltTaskOnAxisFault task parameter.

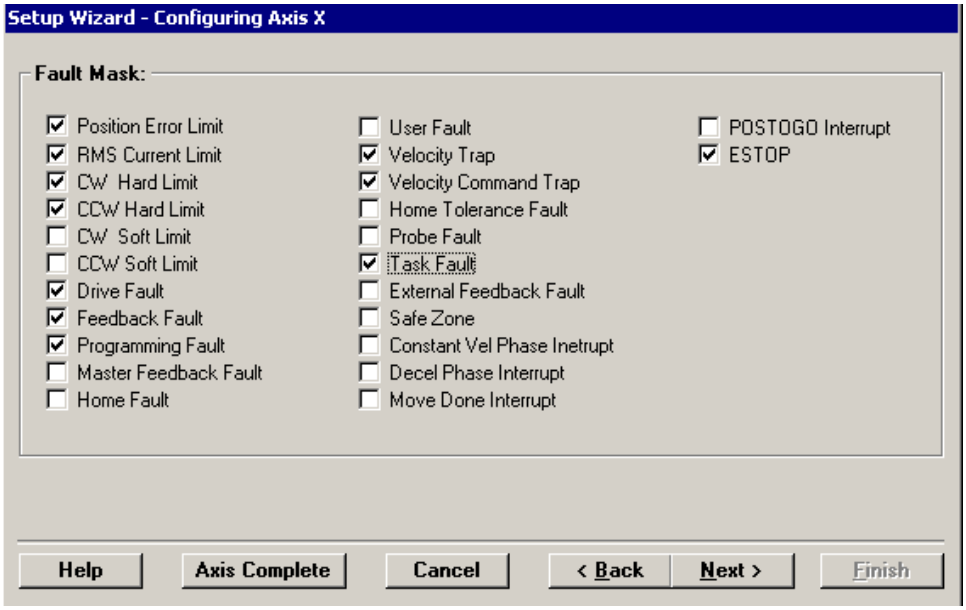


Figure 12-18. The FAULTMASK Configuration Screen

12.11. Configure the DISABLEMASK

This axis parameter determines which faults will cause an axis to be disabled. This parameter is a bit mask where each bit corresponds to a specific fault. The DISABLEMASK takes priority over the HALTMASK and ABORTMASK (i.e., if the DISABLEMASK is set to occur, the HALTMASK or ABORTMASK will have no effect, because the DISABLEMASK would disable the axis before it could halt or abort).

Each bit set in this parameter should also be set in the FAULTMASK axis parameter, to enable detection of that fault condition.

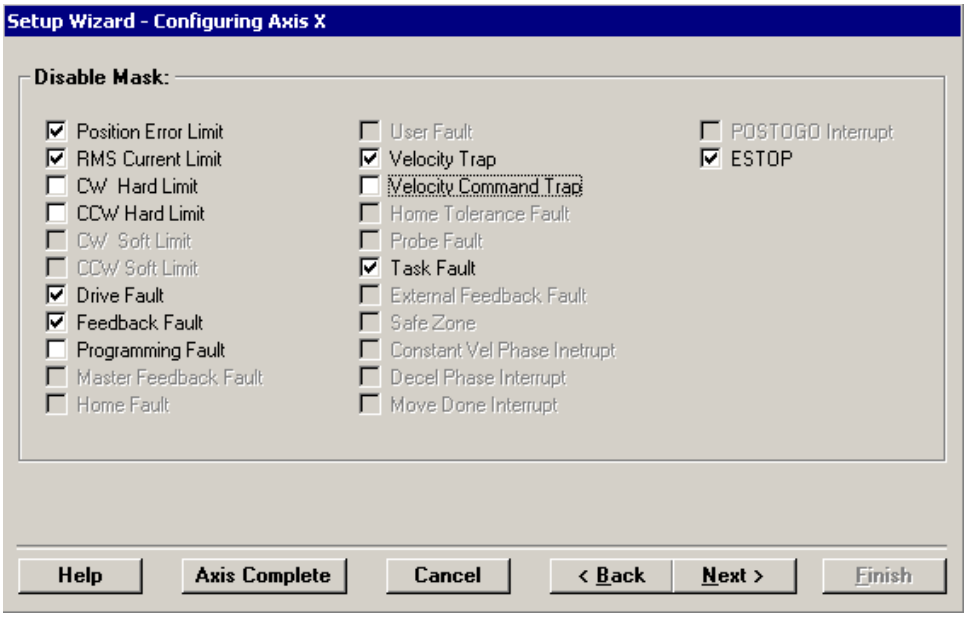


Figure 12-19. The DISABLEMASK Configuration Screen

In the screen above, and all subsequent screens in the same format, a grayed checkbox (such as that for the CW Soft Limit) indicates that the fault condition can not be checked here because it is not checked (set active) on the FAULTMASK page (Section 12.10.). If a box is grayed and checked, it will also be inactive on this screen. It will be ignored unless its corresponding checkbox is also set on the FAULTMASK page.

12.12. Configure the HALTMASK

This axis parameter defines the fault conditions that cause the axis to halt. The value specified for this parameter is a bit mask where each bit corresponds to a specific fault. The axis will halt if the bits for FAULTMASK and HALTMASK are set to one for any given bit position. In halting motion, the axis will decelerate to zero velocity based on the time/rate specified in its deceleration axis parameters. Setting a bit to a one halts the axis when that particular fault occurs (assuming the corresponding bit in the FAULTMASK parameter is set).

This parameter has no effect on the position error tracking. If an axis is triggered by a fault condition to abort and halt simultaneously, the abort takes priority. Each bit set in this parameter should also be set in the FAULTMASK axis parameter, to enable detection of that fault condition. The DISABLEMASK takes priority over the HALTMASK and ABORTMASK, i.e; if the DISABLEMASK is set to occur, the HALTMASK or ABORTMASK will have no effect, because the DISABLEMASK would disable the axis before it could halt or abort.

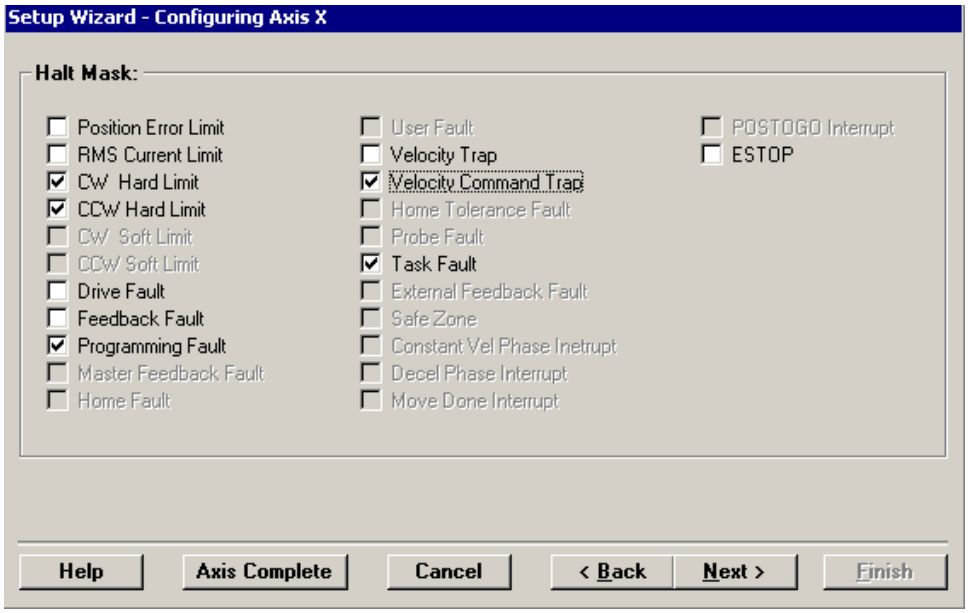


Figure 12-20. The HALTMASK Configuration Screen

12.13. Configure the AUXMASK

This axis parameter allows the user to designate which fault conditions will enable the auxiliary (mode) output associated with the axis. This parameter is a bit mask where each bit corresponds to a specific fault. Each bit set in this parameter should also be set in the FAULTMASK axis parameter, to enable detection of that fault condition.

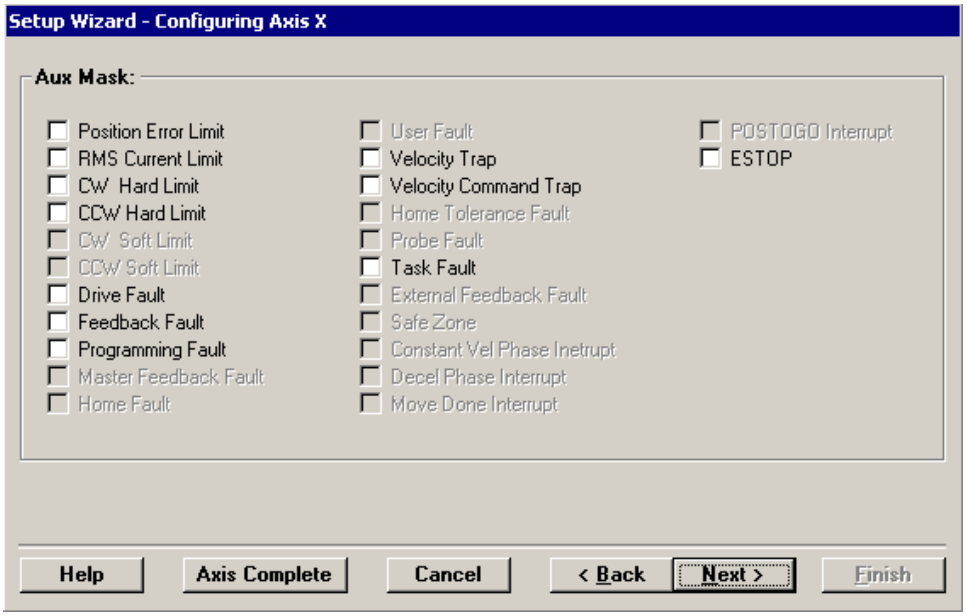


Figure 12-21. The AUXMASK Configuration Screen

12.14. Configure the ABORTMASK

This axis parameter is a mask that determines which fault conditions will cause an axis to abort motion. This parameter is a bit mask where each bit corresponds to a specific fault. If the system aborts motion, it disregards the DECEL axis parameter value and stops the axis abruptly. This also sets the current position error to zero. If an axis is triggered by a fault condition to abort and halt simultaneously, the abort takes priority. Each bit set in this parameter should also be set in the FAULTMASK axis parameter, to enable detection of that fault condition. The DISABLEMASK takes priority over the HALTMASK and ABORTMASK (i.e., if the DISABLEMASK is set to occur, the HALTMASK or ABORTMASK will have no effect, because the DISABLEMASK would disable the axis before it could halt or abort).

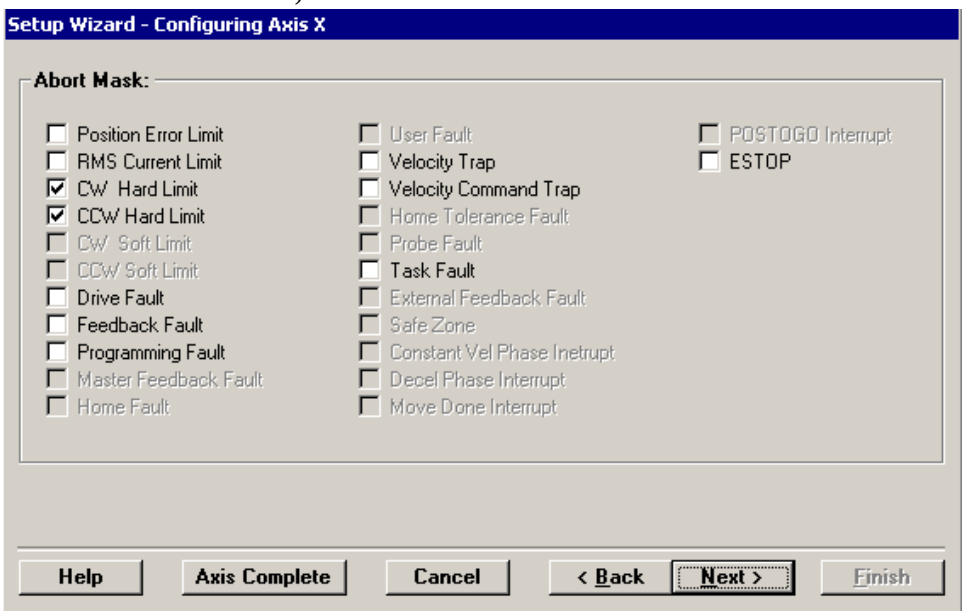


Figure 12-22. The ABORTMASK Configuration Screen

12.15. Configure the INTMASK

This axis parameter allows the user to determine which fault conditions will cause an interrupt to be generated back to the host. This parameter is a bit mask where each bit corresponds to a specific fault. An interrupt is generated if the bits for INTMASK and FAULTMASK are set to one for a given bit position, when that fault occurs. Therefore, each bit set in this parameter should also be set in the FAULTMASK axis parameter, to enable detection of that fault condition.

Interrupts will only be generated for new axis faults, that is, the controller will only generate an interrupt once for each occurrence of a particular axis fault.

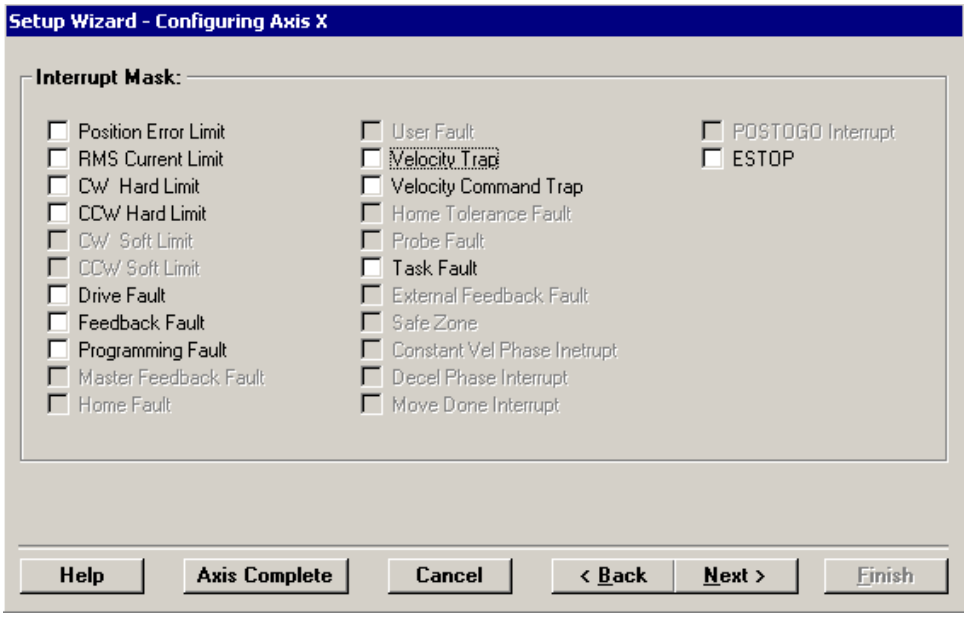


Figure 12-23. The INTMASK Configuration Screen

12.16. Configure the BRAKEMASK

The BRAKEMASK axis parameter is a bitmask used to determine the conditions by which the brake output should be activated, typically for vertically mounted axes. This parameter is a bit mask where each bit corresponds to a specific fault. However, a non-zero BRAKEMASK parameter will cause the brake to be enabled whenever the drive is disabled.

On the UNIDEX 600/650, which has only 1 brake output, any disabled axis with a non-zero BRAKEMASK parameter will engage the brake. The brake will be disengaged when all axes with their BRAKEMASK set are enabled. Also, any of these axes with the BRAKEMASK set will cause the brake output to be activated if the axis faults. If multiple axes are controlled by this brake output, it will require coordination between their DISABLEMASK parameters, such that, if one of the axes were to generate a fault and be disabled, all other axes controlled by the brake output would also be disabled, since the brake, would then be activated by the first axis generating the fault.

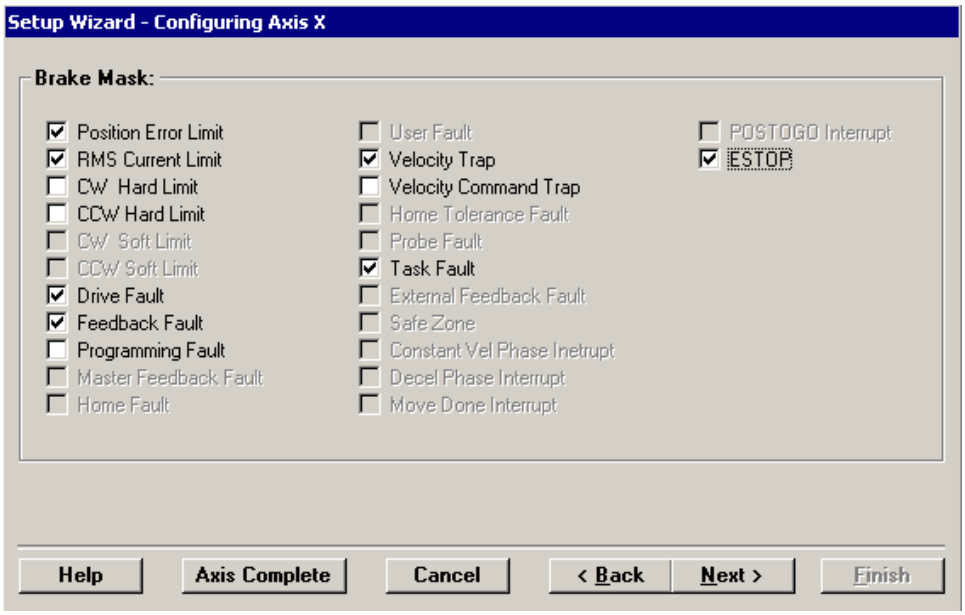


Figure 12-24. The BRAKEMASK Configuration Screen

12.17. Configure the Current Limits

Enter the peak current that your amplifier is capable of producing (see Aerotech Servo Amplifier information in the online help file).

Enter the peak current rating of your motor (see Aerotech motor information in the online help file).

Enter the continuous current rating of your motor (see Aerotech Motor Information in the online help file).

Enter the desired period in milliseconds that the continuous (RMS) current should be calculated over.

Optionally, you may override the calculated values shown in the gray fields at the bottom of the screen. This is done by selecting the 'Override Calculated Limits' checkbox. This will un-gray the I Max and I Avg Limit fields, allowing you to enter values overriding those that were calculated for these fields.

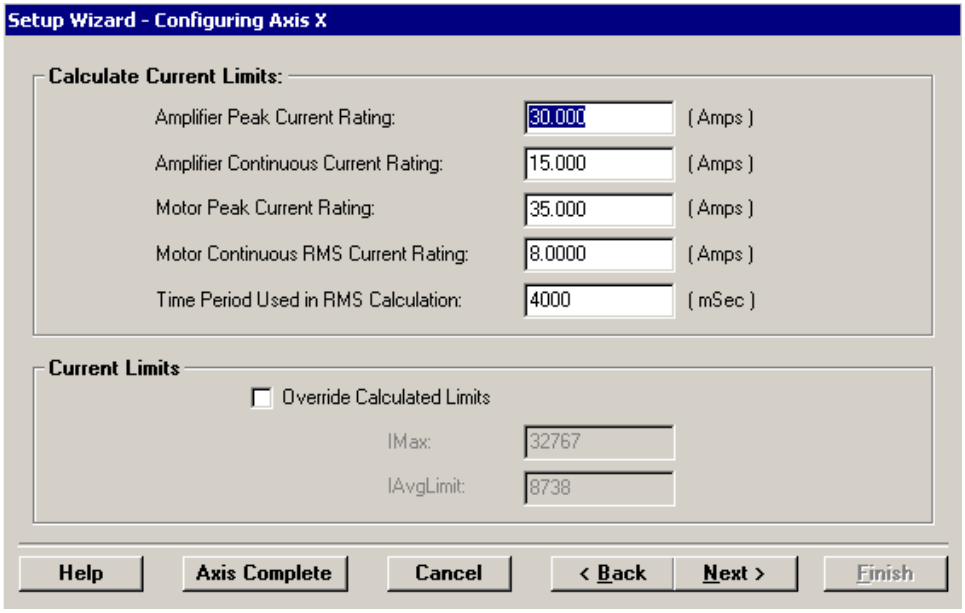


Figure 12-25. The Current Limit Configuration Screen

12.18. Axis Configuration Complete

When all of the axes have been configured, continue with the next step to configure the other (task and global) parameters. Clicking 'Next' will automatically bring you back to the beginning of the next axis configuration (if there are more axes to be configured). Once all axes are configured, you will begin configuring the Task and Global parameters.

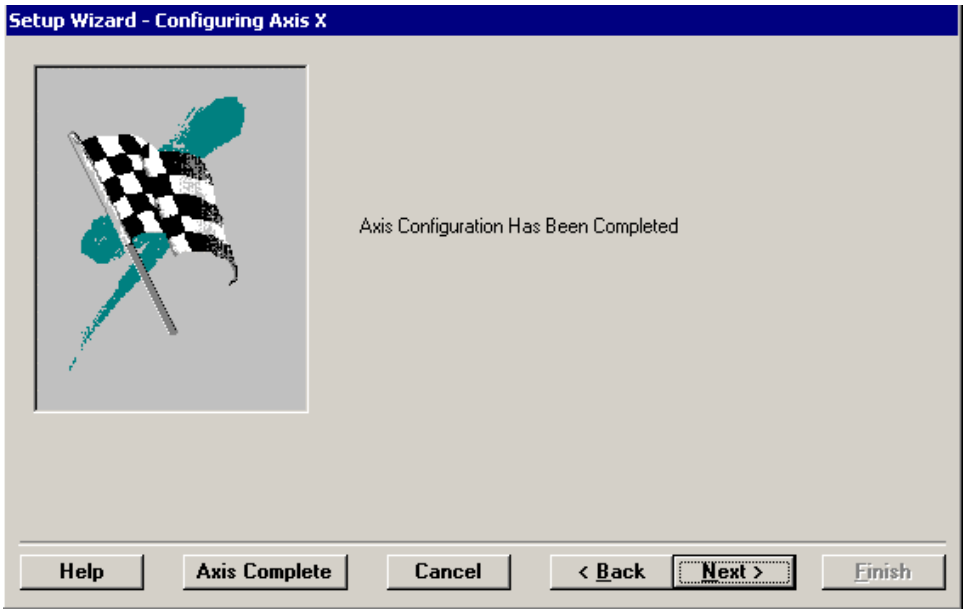


Figure 12-26. The Axis Configuration Complete Screen

12.19. Accel/Decel and Task Initialization

Select G63 (Sinusoidal) or G64 (Linear) as the default Acceleration/Deceleration mode.

Select either G67 (Time) or G68 (Rate) as the default Acceleration/Deceleration type.

Select either the G90 (Absolute) or G91 (Incremental) as the default programming mode.

Select G17, G18 or G19 as the default circular contouring plane.

Select an axis for the Coordinate System 1 I Axis.

Select an axis for the Coordinate System 1 J Axis.

Select an axis for the Coordinate System 1 K Axis.

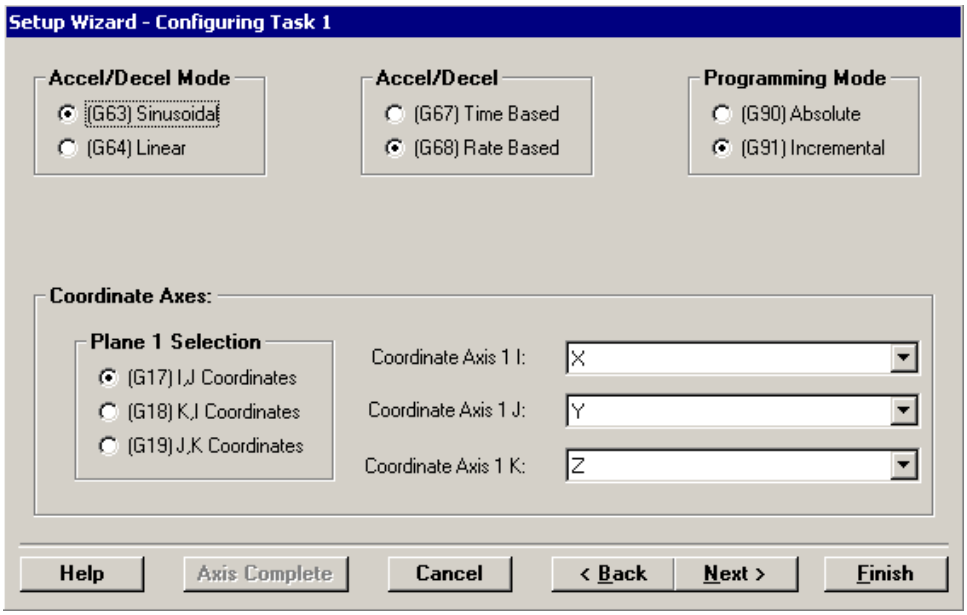


Figure 12-27. The Accel/Decel and Task Initialization Screen

12.20. Configure the ESTOP, Feedhold, and MFO

Would you like the Global Emergency Stop [*Task/ESTOP*] input active?

Would you like an external Feedhold [*Task/Feedhold*] input?

Would you like an external Manual Feedrate Override [*Task/MFO*]?

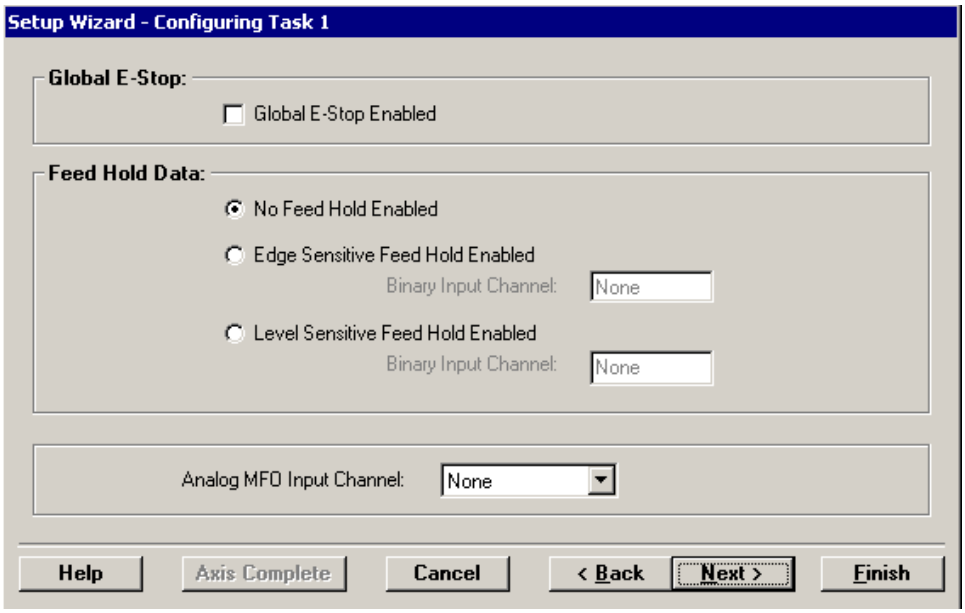


Figure 12-28. The ESTOP, FeedHold, and MFO Configuration Screen

12.21. Configure Synchronous Accel/Decel

Enter the Acceleration Rate [*Task/AccelRateIPS2*] for synchronous motion when Linear Axes are dominant (G99).

Enter the Acceleration Rate [*Task/AccelRateDPS2*] for synchronous motion when Rotary Axes are dominant (G98).

Enter the Acceleration Time [*Task/AccelTimeSec*] for synchronous motion.

Enter the Deceleration Rate [*Task/DecelRateIPS2*] for synchronous motion when Linear Axes are dominant.

Enter the Deceleration Rate [*Task/DecelRateDPS2*] for synchronous motion when Rotary Axes are dominant.

Enter the Deceleration Time [*Task/DecelTimeSec*] for synchronous motion.

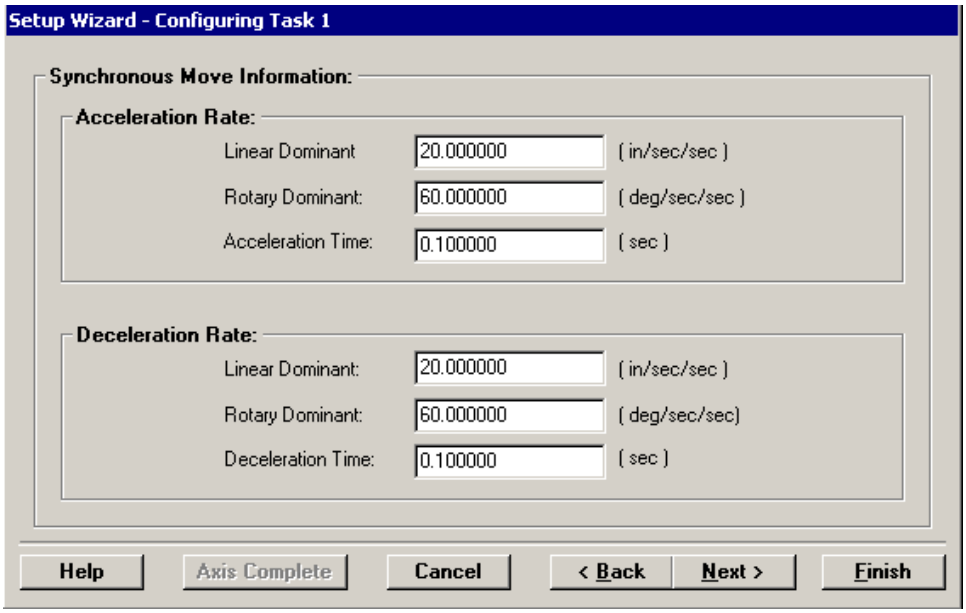


Figure 12-29. The Synchronous Move Information Screen

12.22. Setup Wizard – Configuration Complete

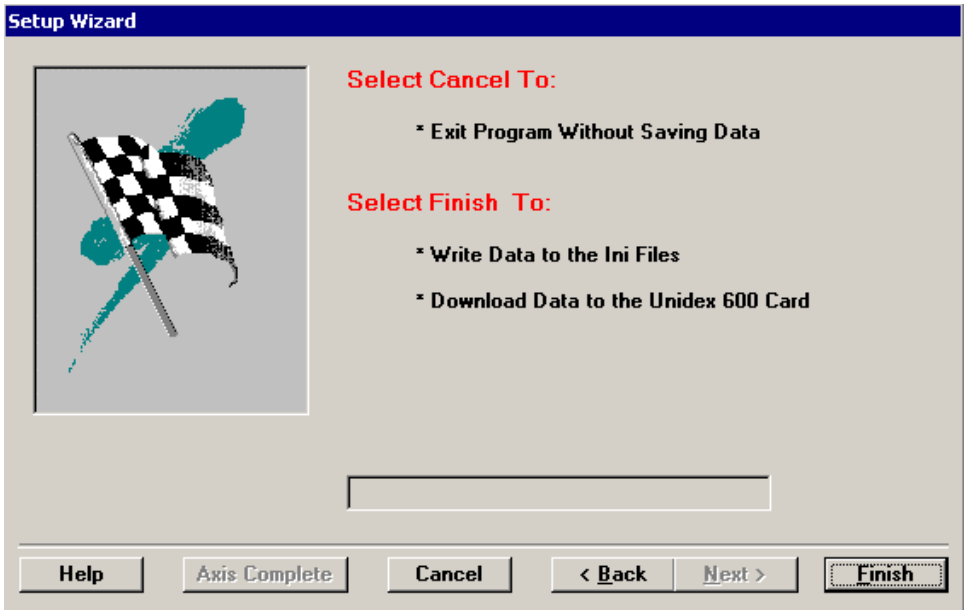


Figure 12-30. The Finish Screen of the Setup Wizard

∇ ∇ ∇

CHAPTER 13: PRMSETUP

In This Section:

13.1. Introduction

The PrmSetup utility is provided when a complete system (motors, drives, tables, etc) is purchased from Aerotech as a system (a single customer order). It will install the pre-configured .Ini files onto your PC.

It may be distributed on the CD-ROM containing the UNIDEX 600 software or on its own.

∇ ∇ ∇

APPENDIX A: GLOSSARY OF TERMS

In This Section:

A.1. Introduction

This appendix contains definition of terms used throughout this manual.

Active - A *task* is *active* once a *program* on its call stack has begun execution. A *program* is active if it is the top *program* on a *task* call stack. A *task* or *program* must be *associated* before becoming *active.*

Associated - A t*ask* is *associated* if it has at least one *program* on its call stack. A *program* is *associated* if it is on at least one of the *task* call stacks.

Asynchronous Motion - Non-coordinated motion that is independent of CNC execution.

Axis Index - Zero based index used to identify an axis.

Axis Parameters - Parameters that affect the specified *physical axis.*

Bind - Declaring permanent ownership of a *task axis* by a *task.* A task binds a task axis.

Callback - A means of communications between the axis processor and frontend. Implies the involvement of an interrupt.

Capture - Declaring ownership of a *task axis* by a *task*. A task captures a task axis.

Contour Motion - Implies CNC Motion commands G1, G2, G3……

Download (Send) - Implies communications to the axis processor card. Data is always *downloaded* or *sent* to the U600.

Executing - A *task* is *executing* if processing the actions of a single *program block*. A *program* is *executing* if it is *active* and a *block* is being processed by a *task*.

Free - Releasing of ownership of a *task axis* by a task.. A task frees a task axis.

Global Parameters - Parameters that affect the over system.

Global Variables - A variable that can be accessed or shared by any *task* or *program.*

Machine Parameters - Parameters that affect the specified *physical axis.*

Map - A way to relate *task axes* to *physical axes.* Map a task axis to a physical axis.

Physical Axis - Implies direct correlation to hardware. *Physical Axis 1* is channel 1.

Point-to-Point Motion - Implies CNC Motion Commands G0.

Program - *Programs* are loaded independently of any *task*. The *program* contains code, variable, and label information. A *program* can be associated with any number of *tasks*. *Program* code and label information are common to all *tasks*. *Program* variables are specific to the process.

Program Handle - An identifier that is assigned to a *program* that the axis processor uses to identify that program.

Program Variables - A variable that is local-in-scope to a given *program*. These are local to the currently *active program*.

Read (Open) - Implies file access. A file is always *read* or *opened*.

Task (Process) - An independently running *process* containing its own set of parameters, variables, and call stack.

Task Axis - Used by a *task* and *mapped* to a *physical axis*. Designated by following letters - X Y Z U V W A B x y z u v w a b.

Task Index - Zero based index used to identify a task.

Task Parameters - Parameters that affect a given *task*.

Task Variables - A variable that is local-in-scope to a given *task*. These can be accessed or shared by all *programs* that are/or become *active* on the given *task.*

Upload - Implies communications to the axis processor card. Data is always *uploaded* to the U600.

Write (Save) - Implies file access. A file is always *written* or *saved*.

∇ ∇ ∇

APPENDIX B: TROUBLESHOOTING

In This Section:
- Introduction

This appendix contains a troubleshooting guide to the axis level.

Table B-1. Troubleshooting to the Axis Level

Fault Message (bold) - Description	Possible Problem(s)	Possible Solution(s)
PosErrLimit - Axis Position error has exceeded *PosErrLimit* axis parameter	1. Mechanical problem or excess friction/load. 2. Velocity/acceleration commanded exceeds system limitations. 3. Axis load fuse blown. 4. *PosErrLimit* axis parameter set too low.	1. Verify mechanics and load. 2. Verify system limitations. 3. Replace axis load fuse. 4. Increase *PosErrLimit* axis parameter.
IAvgLimit - RMS current exceeded *IAvgLimit* axis parameter.	1. Axis duty cycle too high. 2. Mechanical problem or excess friction/load.	1. Add dwell time. 2. Verify mechanics and load.
CW Hardware EOT Limit - CW hardware end-of-travel limit encountered.	1. Possible user programming error. 2. User unit scaling problem.	1. Correct program. 2. *CntsPerInch* parameter.
CW Software EOT - Axis attempted to move beyond limit of *CWEOT* axis parameter	1. *CWEOT* axis parameter is incorrectly set or the user's program is in error.	1. Correct *CWEOT* axis parameter or user program.
Drive fault - Drive fault has occurred.	1. Drive module generated an over current fault or has failed.	1. Cycle power to drive module or replace it.
Feedback fault- Feedback device fault.	1. Feedback cable loose/disconnected. 2. Feedback device failed. 3. Feedback device without power. 4. Drive power out.	1. Verify cable. 2. Test/replace device. 3. Verify supply levels. 4. Apply power.
Programming fault - An error has occurred in a user CNC program, i.e., a syntax error or an invalid condition.	1. An error has occurred during a program, for example, a syntax error or attempting to set a parameter to an invalid value, or an axis that was commanded to move while disabled.	1. Correct the user program.
Master Feedback fault - Feedback device fault on axis configured as master.	1. Feedback cable loose/disconnected. 2. Feedback device failed. 3. Feedback device without power.	1. Verify cable. 2. Test/replace device. 3. Verify supply levels.
Homing fault - Error during a homing cycle.	1. The type of homing cycle specified could not complete.	1. An invalid homing cycle was attempted, the limits are bad, open, reversed, or there is a hardware problem, possibly the reference pulse.
User fault - User error has occured.	1. The frontend application caused this fault to be generated, it should have handled it.	1. Contact the developer to correct the frontend application.

Table B-1. Troubleshooting to the Axis Level

Fault Message (bold) - Description	Possible Problem(s)	Possible Solution(s)
Velocity Trap - Actual motor velocity has exceeded *VelTrap* axis parameter.	1. Axis servo loop not tuned well enough. 2. Axis load fuse blown. 3. The axis load or friction has increased.	1. Adjust servo loop gains. 2. Replace axis load fuse. 3. Increase *VelTrap* axis parameter.
Velocity Command Trap - Commanded motor velocity has attempted to exceed *VelCmdTrap* axis parameter.	1. The maximum axis commanded velocity has been exceeded.	1. Correct program or verify axis *VelCmdTrap* axis parameter value.
Home Switch Tolerance - The *HomeSwitchTol* axis parameter has been exceeded during a home cycle.	1. The reference pulse (marker or resolver null) is too close to the home limit.	1. Adjust the feedback device so that the reference pulse is greater than the *HomeSwitchTol* axis parameter. 2. Increase the *HomeSwitchTol* axis parameter.
Probe - The touch probe input has been activated.	1. Activation of the probe input was due to a programming error or an obstruction.	1. Modify the program or clear the obstruction. 2. Adjust servo loop gains or adjust mechanics.
Task fault - A task fault occurred.	1. A task fault occurred and may any of the active faults in the faultmask.	1. Correct the fault causing the task fault.
External feedback - The actual axis velocity varied from the commanded velocity by an amount greater than the *FBWindow* axis parameter.	1. Mechanical problem or excess friction/load. 2. Velocity/acceleration commanded exceeds system limitations. 3. Axis load fuse blown. 4. *FBWindow* axis parameter set too low.	1. Verify mechanics and load. 2. Verify system limitations. 3. Replace axis load fuse. 4. Increase *FBWindow* axis parameter.
SafeZone fault - The axis violated a SafeZone defined by the *SafeZone* axis parameter.	1. The user CNC program violated the safezone defined by the *SafeZoneMode*, *SafeZoneCW*, and the *SafeZoneCCW* axis parameters.	1. Correct the CNC program or modify the *SafeZoneMode*, *SafeZoneCW*, and *SafeZoneCCW* axis parameters.

∇ ∇ ∇

APPENDIX C: PARAMETERS

In This Section:

Task, Machine, and Global parameters only apply to CNC directed motion (**G1**, **G2**, **G3**). Axis parameters apply to asynchronous motion and **G0** commands.

C.1. Description

This section contains the parameter reference table and the parameter descriptions. The reference table contains a one-line summary of each parameter; the description contains more detailed descriptions of each parameter.

The following tables provide the names of every *UNIDEX* 600 Series controller parameter, and its maximum, minimum and default values. These tables are generated by the *PrmManu*l example program that queries the axis processor directly. Therefore, the tables below are correct for the firmware that was executing at the time of the creation of this manual. See the AerParm library functions in the *U600 Library Reference Manual P/N EDU156* and the *PrmManul* program for examples explaining how to retrieve the current valid parameters for the firmware.

The table does not give the units of the parameters; the units are implicit in the name. For Machine, Task, and Global parameters, the units are clearly implied by the suffix of the name. For example: "*HomeOffsetDeg*" is in units of degrees. Some other abbreviations used in these suffixes are:

"Sec" -	Seconds
"IPS" -	Inches Per Second
"DPS" -	Degrees Per Second
"IPS2" -	Inches Per Second^2
"DPS2" -	Inches Per Second^2
"RPM" -	Revolutions per minute
"Index" -	An *AXISINDEX_xxxx* constant
"X" or "Y" or "Axis" -	An *AXISINDEX_xxxx* constant
"Mask" or "Status"-	A bitwise mask

Axis parameters are a special case: the units are not in the name. Axis parameter units must be inferred from the meaning of the parameter:

times	Milliseconds
distances	Machine counts
velocities or speeds	Machine counts per millisecond
accelerations	Machine counts per millisecond^2

Exceptions to the above will be noted in the description text of that parameter.

C.1.1. Name

The parameter name. Axis parameters are always shown in all UPPERCASE letters. Global, Task, and Machine parameters are shown in mixed case for greater legibility.

C.1.2. Type (of Parameter)

What the parameter applies to. Axis and Machine parameters apply only to the specified axis. Task parameters apply to the specified task. Global parameters apply to the axis processor in general.

C.1.3. Access

A series of letters indicating access. R is read, W is write, U means the axis processor may/will update/write this value. If the code contains a W and U, then both the user and the controller may write to it. For example, the axis parameter *CLOCK* has access RWU. In these cases, what the user writes may be overwritten.

C.1.4. Minimum, Maximum

The parameter can be any value between and including these two values. Many parameters have as a limit: "n/a". This indicates the parameter has the full range of an integer 32-bit value (-2,147,483,648 to +2,147,483,648).

C.1.5. Default

This is the value of the parameter immediately after the controller is reset.

C.2. Axis Parameters

Axis parameters are on a per axis basis, meaning that each axis has its own independent set of parameters. All axis parameters are specified in capital letters and integer values. Machine parameters are specified on a per axis basis as well, but are decimal or floating point values.

Table C-1. Axis Parameters

Name	Parameter #	Access	Minimum	Maximum	Default
A1	129	RW	-2,147,483,648	2,147,483,647	0
A2	130	RW	-2,147,483,648	2,147,483,647	0
ABORTMASK	72	RW	0	4,294,967,295	0
ACCEL	48	RW	0	100,000	0
ACCELMODE	50	RW	0	3	0
ACCELRATE	53	RW	1	2,147,483,647	100,000
AFFGAIN	24	RW	0	1,000,000	0
ALPHA	119	RW	0	65,536	65,536
ALT_STATUS	113	RU	0	4,294,967,295	0
AUX	21	RW	0	1	0
AUXDELAY	132	RW	0	4,294,967,295	0
AUXMASK	68	RW	0	4,294,967,295	0
AUXOFFSET	71	RW	0	4,294,967,295	0
AUXVELCMD	137	RW	0	48	0
AVGVEL	12	RU	-2,147,483,648	2,147,483,647	0
AVGVELTIME	13	RW	10	1,000	1,000
B0	126	RW	-2,147,483,648	2,147,483,647	8,192
B1	127	RW	-2,147,483,648	2,147,483,647	0
B2	128	RW	-2,147,483,648	2,147,483,647	0
BASE_SPEED	110	RW	0	10,000,000	0
BRAKEMASK	124	RW	0	4,294,967,295	0
CAMADVANCE	117	RW	-100,000,000	100,000,000	0
CAMOFFSET	84	RW	-2,147,483,648	2,147,483,647	0
CAMPOINT	79	RU	0	4,294,967,295	0
CAMPOSITION	78	RU	0	4,294,967,295	0
CCWEOT	39	RW	-2,147,483,648	2,147,483,647	-2,147,483,647
CLOCK	4	RWU	0	4,294,967,295	0
CWEOT	38	RW	-2,147,483,648	2,147,483,647	2,147,483,647
DACOFFSET	125	RW	-32,767	32,767	0
DECEL	49	RW	0	100,000	0
DECELMODE	51	RW	0	3	0
DECELRATE	54	RW	1	2,147,483,647	100,000
DISABLEMASK	66	RW	0	4,294,967,295	12,483
DRIVE	20	RW	0	1	0
ECHO	3	RW	0	4,294,967,295	0
EXTR2DSCL	109	RW	1	65,536	1
FAULT	64	RWU	0	4,294,967,295	0

Table C-1. Axis Parameters (continued)

Name	Parameter #	Access	Minimum	Maximum	Default
FAULTMASK	65	RW	0	4,294,967,295	69,839
FBWINDOW	43	RW	0	1,000,000	0
FEEDRATEMODE	52	RW	0	1	0
GANTRYMODE	140	RW	0	32	0
GANTRYOFFSET	141	RW	-2,147,483,648	2,147,483,647	0
GEARMASTER	142	RW	-2,147,483,648	2,147,483,647	0
GEARMODE	136	RW	0	1	0
GEARSLAVE	134	RW	-2,147,483,648	2,147,483,647	0
HALTMASK	69	RW	0	4,294,967,295	60
HOMEOFFSET	122	RWU	0	4,294,967,295	0
HOMESWITCHPOS	56	RU	-2,147,483,648	2,147,483,647	0
HOMESWITCHTOL	57	RW	0	16384	0
HOMEVELMULT	116	RW	1	100	100
IAVG	31	RU	-2,147,483,648	2,147,483,647	0
IAVGLIMIT	33	RW	0	32,767	32,767
IAVGTIME	34	RW	10	8,000	1,000
ICMD	35	RU	-2,147,483,648	2,147,483,647	0
ICMDPOLARITY	123	RW	4,294,967,295	1	0
IMAX	32	RW	0	32,767	32,767
INPOSLIMIT	37	RW	0	65,536	65
INTMASK	67	RW	0	4,294,967,295	0
IOLEVEL	70	RW	0	63	63
IVEL	11	RU	-2,147,483,648	2,147,483,647	0
KI	16	RW	0	5,000,000	2,000
KP	17	RW	0	10,000,000	10,000
MASTERLEN	82	RW	0	2,147,483,647	0
MASTERPOS	80	RWU	-2,147,483,648	2,147,483,647	0
MASTERRES	81	RU	0	4,294,967,295	0
MAXCAMACCEL	121	RW	0	65,536	0
MAX_PHASE	111	RW	0	360	0
MOTIONSTATUS	97	RU	0	4,294,967,295	0
MOVEQDEPTH	89	RU	0	4,294,967,295	0
MOVEQSIZE	90	RU	0	4,294,967,295	16
PGAIN	18	RW	0	100,000	10
PHASE_SPEED	112	RW	1	10,000,000	1
PHASEAOFFSET	143	RW	-32,767	32,767	0
PHASEBOFFSET	144	RW	-32,767	32,767	0
POS	2	RWU	-2,147,483,648	2,147,483,647	0
POSCMD	7	RWU	-2,147,483,648	2,147,483,647	0
POSERR	8	RU	-2,147,483,648	2,147,483,647	0
POSERRLIMIT	36	RW	0	10,000,000	65,536
POSTARGET	10	RU	-2,147,483,648	2,147,483,647	0
POSTOGO	9	RU	-2,147,483,648	2,147,483,647	0
POSTOGOIRQ	131	RW	0	2,147,483,648	0

Table C-1. Axis Parameters (continued)

Name	Parameter #	Access	Minimum	Maximum	Default
POSTOLERANCE	138	RW	0	2,147,483,647	0
POSTOLTIME	139	RW	0	8,388,607	0
PROFILETIME	86	RW	1	1,000	250
PROFQDEPTH	87	RU	0	4,294,967,295	0
PROFQSIZE	88	RU	0	4,294,967,295	32
RAWPOS	6	RU	-2,147,483,648	2,147,483,647	0
RESOLVER	5	RU	0	10,000,000	0
REVERSALMODE	29	RW	0	1,000	0
REVERSALVALUE	30	RU	0	1,000	0
SAFEZONECCW	45	RW	-2,147,483,648	2,147,483,647	0
SAFEZONECW	44	RW	-2,147,483,648	2,147,483,647	0
SAFEZONEMODE	46	RW	0	2	0
SCALEPGAIN	133	RW	0	2	0
SERVOSTATUS	98	RU	0	4,294,967,295	64,896
SIMULATION	47	RW	0	2	0
SOFTLIMITMODE	115	RW	0	1	0
STATUS	1	RU	0	4,294,967,295	64,896
SYNCSPEED	85	RW	1	2,147,483,647	1,000
SYSTEMCLOCK	58	R	0	2,147,483,647	0
VELCMDTRAP	41	RWU	0	65,536,000	2,116,667
VELPOSITION	42	R	0	4,294,967,295	0
VELTIMECONST	120	RW	0	1,000	0
VFF	19	RW	0	1	0
VGAIN	118	RW	0	8,388,607	0

C.2.1. Modifying an Axis' Parameter within a CNC Program

Any axis' parameter may be modified within a CNC program (or MDI command line) by specifying the axis parameter name followed by a decimal point and the axis name. The case of these axis parameters is significant (all are upper case letters), as defined in Table C-1. Axis Parameters.

The axis name is that assigned when the axis is configured and bound to the task within the axis configuration wizard. If the default axis name is used, the task axis names would apply.

C.2.2. ABORTMASK

This axis parameter is a mask that determines which fault conditions will cause an axis to abort motion. This parameter is a bit mask where each bit corresponds to a specific fault. If the system aborts motion, it disregards the **DECEL** axis parameter value and stops the axis abruptly. This also sets the current position error to zero. If an axis is triggered by a fault condition to abort and halt simultaneously, the abort takes priority. Each bit set in this parameter should also be set in the **FAULTMASK** axis parameter, to enable detection of that fault condition. The **DISABLEMASK** takes priority over the **HALTMASK** and **ABORTMASK**, i.e.; if the **DISABLEMASK** is set to occur, the

HALTMASK or **ABORTMASK** will have no effect, because the **DISABLEMASK** would disable the axis before it could halt or abort.

C.2.3. ACCEL

The time (in milliseconds) for the axis to ramp up to a new velocity. This axis parameter is used only if the **ACCELMODE** axis parameter is set to 0 or 1 (Sinusoidal/Linear) for time based ramping of **G0** (point to point) moves and asynchronous moves.

This parameter is not used for CNC contoured motion (**G1**, **G2, G3**) (refer to *AccelTimeSec* task parameter for ramping contoured moves).

C.2.4. ACCELMODE

This axis parameter allows the user to select the type of ramping used during the execution of **G0** (point to point) moves and asynchronous moves. This ramping may be time-based (using the **ACCEL** parameter) or rate-based (using the **ACCELRATE** parameter). Also, the user can configure the ramping to be either linear or sinusoidal (1-cosine). Figure C-1 indicates how to set this parameter. The default for this parameter is zero for a time-based (1-Cosine) ramp.

This parameter is not used for CNC contoured motion (**G1**, **G2, G3**) (refer to *AccelTimeSec* task parameter for ramping contoured moves).

0 - (1-Cosine) Ramping - Time Based
1 - Linear Ramping - Time Based
2 - (1-Cosine) Ramping - Rate Based
3 - Linear Ramping - Rate Based

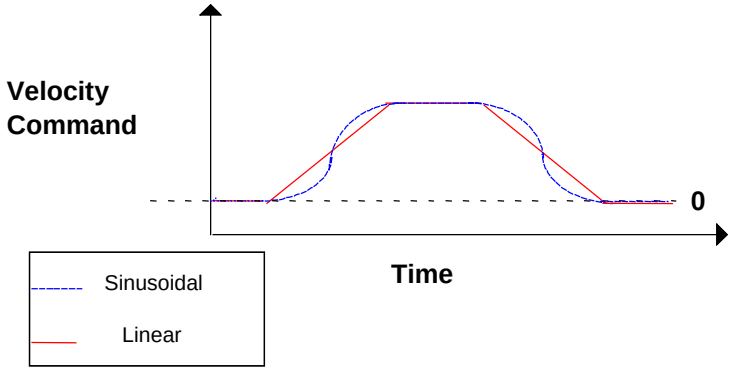


Figure C-1. ACCELMODE Ramp Setting

C.2.5. ACCELRATE

This parameter sets the rate of acceleration for G0 (point to point) moves and asynchronous moves, while the **ACCELMODE** parameter specifies rate based ramping. The units are machine counts per second^2. This parameter is used only when the **ACCELMODE** parameter is set to 2 and 3.

This parameter is not used for CNC contoured motion (**G1**, **G2, G3**) (refer to *AccelTimeSec* task parameter for ramping contoured moves).

C.2.6. AFFGAIN

This axis parameter sets the Acceleration Feed Forward Gain within the servo loop. **AFFGAIN** may be used to compensate for inertia loads, on torque mode axes. The value of the **AFFGAIN** should be proportional to the inertia, i.e. the larger the inertia of the system, the larger the **AFFGAIN**. The **ALPHA** axis parameter will filter (smooth out) the rapid current command changes produced by the **AFFGAIN** axis parameter.

Refer to the *UNIDEX* 600 Hardware manual under "Servo loop" for a description of how **AFFGAIN** is used in the servo loop, or Tuning Servo Loops, under the AerTune chapter of the User**'**s Guide.

You can use the AutoTune feature within the AerTune.exe utility (on the Tools menu) to automatically determine servo loop gains for a torque mode axis.

C.2.7. ALPHA

The **ALPHA** axis parameter filters the commanded acceleration, which is scaled by the **AFFGAIN** axis parameter. This minimizes disturbances in the servo loop caused by sudden acceleration changes that might be generated, such as a handwheel input with a large scaling factor. The scaling of this parameter is inverse, meaning that 65,536 produces no filtering of the **AFFGAIN** axis parameter and a value of 1 would produce the maximum filtering. This filtering only takes place if the **AFFGAIN** parameter is non-zero.

C.2.8. ALT_STATUS

This is a bitwise value that provides additional status for the axis. To test for a particular condition, simply bitwise-and (BAND operator) the **SERVOSTATUS** with the desired bit value. For example;

```
If (ALT_STATUS.X BAND 0h00040000)
```

could be used to test for the X axis being used in simulation mode.

Table C-2. ALT_STATUS Bit Definitions

Name	Hex Value	Description
LAST_MVE	0h00000001	final move into home
ALT_HME	0h00000002	alternate home
PWR_HME	0h00000004	home start in limit
FND_HME	0h00000008	home limit detected
TRQ_SET	0h00000010	valid torque parameters
TRQ_EN	0h00000020	torque enable
HALL_COMM	0h00000040	hall effect commutation
HALL_EDGE	0h00000080	first edge received
DUAL_LOOP	0h00000100	dual loop feedback
VME_JOG	0h00000200	vme jog mode
LST_DIR	0h00000400	ccw motion in hall commutation
JOG_DIR	0h00000800	positive jog direction
EXT_RES	0h00001000	external resolver feedback
REV_ERROR_ON	0h00002000	reversal error enabled
LAST_REV_POS	0h00004000	last reversal in positive direction
LAST_REV_NEG	0h00008000	last reversal in negative direction
SAFE_ENABLE	0h00010000	safe zone enabled
SAFE_OUTSIDE	0h00020000	safe zone is outside region
SIMULATION	0h00040000	feedrate override
HOME_VELOCITY	0h00080000	home off the velocity transducer
ACCEL_RATE	0h00100000	acceleration rate
DECEL_RATE	0h00200000	deceleration rate
PHASE_MODE	0h00400000	phase advance mode active
HOME_COMPLETE	0h00800000	home cycle complete
LIMIT_OVERRIDE	0h01000000	always check soft limits if reset
SLAVE_ENCODER	0h02000000	slave encoder
QUE_CAM	0h04000000	cam table queued
HOME_NOLIM	0h08000000	home without home limit
SPEED_OUTPUT	0h10000000	velocity command mode
STEPR	0h20000000	stepper motor
Invert Polarity	0h40000000	ICMD Polarity = 1
Encoder Master Feedback	0h80000000	axis slaved to master

C.2.9. AUX (Mode) Output

This axis parameter reflects the state of the auxiliary output (also referred to as the mode output) of a selected axis. An auxiliary value of one indicates that the auxiliary output is enabled. This output may be used to activate a motor brake for a vertically mounted axis. The user may configure the **FAULTMASK** and **AUXMASK** parameters to cause this output to enable on an axis fault. Therefore, each time a fault condition occurs, the system would apply a brake to the motor. The enabled state of the **AUX** output can be specified as either a high or low voltage state. Refer to the **IOLEVEL** axis parameter for an explanation of the process. By default the **AUX** is active low, meaning, when the **AUX** is enabled, there is a zero voltage (sinking current) on the auxiliary line.

C.2.10. AUXDELAY

This parameter suspends fault checking for all faults in the **AUXMASK** parameter after a "fault acknowledge" for the specified period of time. This parameter is specified in units of 10 milliseconds (i.e., 1 = 10ms, 5 = 50ms, etc…).

C.2.11. AUXMASK

This parameter allows the user to designate which fault conditions will enable the auxiliary (mode) output associated with the axis. This parameter is a bit mask where each bit corresponds to a specific fault. Each bit set in this parameter should also be set in the FAULTMASK axis parameter to enable detection of that fault condition.

C.2.12. AUXOFFSET

This parameter is added to the master position before doing the auxiliary table lookup. For example, if the table covers master positions from 0 to 360 degrees and the actual master position is 2 degrees and the *CAMOFFSET* parameter is 3 degrees, then the CNC will use the value of 5 degrees as the master position to search the table. To understand how this parameter functions, the reader must be familiar with the operation of the synchronized auxiliary output tables on the *UNIDEX* 600 Series motion controller. In brief, each synchronized auxiliary output table entry specifies a master position and a corresponding state for the auxiliary output. When the observed master position becomes greater than or equal to that specified in the table entry, the output gets set to the appropriate state. The only requirement is that the master positions must constantly increase and never repeat. This parameter refers to an offset applied to the master position of the auxiliary output table associated with an axis. The point at which the table begins and ends is advanced or retarded. The user must be aware of the table's setup before setting the value of this parameter.

C.2.13. AUXVELCMD

The **AUXVELCMD** specifies the source of an external velocity command, which is added to the current axis' velocity command. The **AUXVELCMD**, while active, does not change the position command or target of the specified axis. Only the actual position of the motor will change in response to **AUXVELCMD**. When **AUXVELCMD** is disabled (set to zero) the position command and target are updated to reflect any change in position, which may have occurred while **AUXVELCMD** was active. The valid range is 0 to 48 where:

- 0: No auxiliary velocity command is added to the current axis velocity command.
- 1-16: Axis 1 to 16's velocity command is added to the current axis velocity command.
- 17-32: Axis 1 to 16's actual velocity is added to the current axis velocity command. Where 17 is axis 1, 18 is axis 2…32 is axis 16.
- 33-48: Axis 1 to 16's torque command is added to the current axis velocity command. Where 33 is axis 1, 34 is axis 2…48 is axis 16

The gantry mode utilizes the AuxVelCmd axis parameter for commanding motion on the slave axis, so it should not be used while the gantry mode is active.

This parameter does not change the velocity command; it integrates the auxiliary velocity command into the position command.

The AUXVELCMD can be useful, for example, when tracing an X-Y pattern on a part that is moving on a conveyor belt, when the X-Y stages are not moving with the conveyor belt.

C.2.14. AVGVEL

The controller maintains a read-only parameter called **AVGVEL** that reports the average velocity for a given axis. This average has no effect on the operation of the controller, but is maintained for the benefit of the application program. The **AVGVELTIME** parameter specifies the time period to average the velocity over. The units for this parameter are counts per millisecond.

C.2.15. AVGVELTIME

The **AVGVELTIME** axis parameter specifies the time period over which to average the velocity when calculating the value of the AVGVEL axis parameter and the AvgVelUnits machine parameter. It is in units of milliseconds. However, this parameter is only used to the next lowest multiple of 10 milliseconds. For example, a value of 12 or 19 is equivalent to a value of 10. See the **AVGVEL** axis parameter for more details.

C.2.16. B0/B1/B2/A1/A2

These axis parameters may be used to eliminate instabilities within the servo loop due to electrical or mechanical oscillations. These instabilities must first be identified. These axis parameters implement a generic second order digital filter to the torque command in the following form:

$$Y(Z) / X(Z) = ((B0 + B1 * Z^{-1} + B2 * Z^{-2}) * 8192) / ((A0 + A1 * Z^{-1} + A2 * Z^{-2}) * 8192)$$

Where Z^{-1} is the delay operator, **A0** is always 1, and the sampling frequency is the current servo loop update rate (typically 4 kHz). Since **A0** is always assumed to be 1, there is no corresponding axis parameter. These parameters are only used in torque loop configured axes. Setting B0=B1=4,096 will provide the minimal amount of filtering.

You must disable the axis before entering the filter constants.

If this filter is used in conjunction with the **GANTRYMODE**, it must be applied to the master and slave axes.

Aerotech provides a utility, Filter.exe to compute coefficients (**B0**/**B1**/**B2**/**A2**/**A3**) for the torque command filter based upon a desired roll-off frequency. You may use digital filter design software utilities other than Filter.exe to calculate the coefficients, but there are some considerations. Digital filter design software utilities usually generate floating point numbers for the values of these filter coefficients. However, the axis parameters must be specified as integers. The conversion from floating point to integer is accomplished by scaling all the filter coefficients by 8,192 and rounding/truncating the result. For example, assume a digital filter design utility yields the following coefficients.

B0 = 0.9760	B1 = -1.8797	B2 = 0.9760
A0 = 1	A1 = -1.8797	A2 = -1.8797

Multiplying by 8,192 and truncating yields produces the following:

B0 = 7,996	B1 = -15,398	B2 = 7,996
A0 = 8,192	A1 = -15,398	A2 = 7,800

Performing the conversion from floating point values to integer values can affect the DC gain of the filter due to truncation or rounding of the fractional results. To verify the DC gain of the filter, perform the following operation with both floating point values and their scaled integer equivalents.

$$(B0 + B1 + B2) / (A1 + A2 + A3) = DC\_GAIN$$

The user must be familiar with digital signal theory in order to use these parameters properly, otherwise the Aerotech filter utility program must be used.

If the DC Gain of the floating point and integer representations are not equal, then adjust the **B0**/**B1**/**B2**/**A2**/**A3** coefficients. This can usually be accomplished by increasing or decreasing **B0**/**B2** and/or **B1** by ±1 count to compensate for the rounding effect.

C.2.16.1. A Typical Low-Pass Filter

These filter values are most often used with brushless motors providing, a 250 Hertz low pass filter:

A1 = -11908
A2 = 4699
B0 = 245
B1 = 491
B2 = 245

C.2.17. BASE_SPEED

This parameter, as well as the **MAX_PHASE** and **PHASE_SPEED** parameters, allow the speed torque characteristics of an AC brushless motor to be customized. Normally, these parameters are only used with motors having a large back-EMF (K_b) constant. Additional speed may be generated from a given motor by adjusting each slope of a dual slope curve that determines the torque angle at various motor speeds. The **BASE_SPEED** determines the speed the motor will reach at a 20° phase advance. The **PHASE_SPEED** parameter determines the speed the maximum torque angle will be reached that is specified by the **MAX_PHASE** parameter.

This function sacrifices torque output for speed.

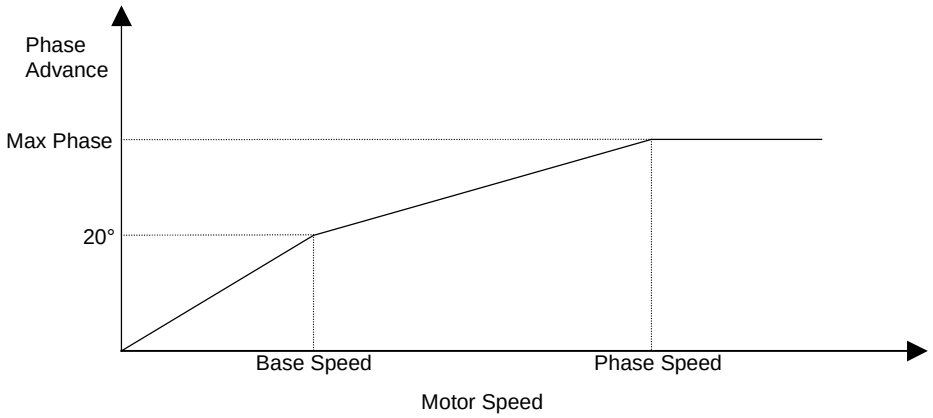


Figure C-2. Phase Advance, Torque Angle vs. Speed Relationship

As the motor velocity reaches the base speed, the phase advance reaches 20 electrical degrees. The **PHASE_SPEED** parameter specifies the motor velocity at which the phase advance reaches the **MAX_PHASE** degrees offset. The units are machine steps per second.

C.2.18. BRAKEMASK

The **BRAKEMASK** parameter is a bitmask used to determine the conditions the brake output should be activated, typically for vertically mounted axes. This parameter is a bit mask where each bit corresponds to a specific fault. However, a non-zero **BRAKEMASK** parameter will cause the brake to be enabled whenever the drive is disabled.

On the *UNIDEX* 600/650, which has only 1 brake output; any disabled axis with a non-zero **BRAKEMASK** parameter will engage the brake. The brake will be disengaged when all axes with their **BRAKEMASK** set are enabled. Also, any of these axes with the **BRAKEMASK** set will cause the brake output to be activated if the axis faults. If multiple axes are controlled by this brake output, it will require coordination between their **DISABLEMASK** parameters such that if one of the axes were to generate a fault and be disabled, that all other axes controlled by the brake output would also be disabled, since the brake would then be activated by the first axis generating the fault.

C.2.19. CAMADVANCE

This axis parameter is used only for electronic gearing, File Driven Camming and **HANDWHEEL** camming. It serves the same function as the **CAMOFFSET** axis parameter, except that instead of shifting the master position by a constant value, it shifts it proportional to the velocity of the master axis. Its intended use is to compensate for following error on the slave axis. The value shifted by the **CAMADVANCE** axis parameter is in addition to the shift value applied by the **CAMOFFSET** axis parameter, and is:

Shift value = Master velocity (cnts / msec) * CAMADVANCE / 8192

This parameter is without units, and its value may be negative. It is set for the slave axis. This parameter should always be set before **SYNC**ing the axis in a non-zero mode.

This parameter value is added to the master position before doing the cam table lookup. It is used to shift, or offset the relationship between the master and slave positions, as defined in a cam table (Figure C-3, Camming Illustration). This allows you to shift the curve left or right, without writing a new cam table.

C.2.20. CAMOFFSET

This axis parameter is used only for electronic gearing, Camming from a File and HANDWHEEL camming. The value of this axis parameter is added to the master position before doing the cam table lookup. It is used to shift, or offset the relationship between the master and slave positions, as defined in a cam table (see Figure C-3, Camming Illustration). This allows you to shift the curve left or right, without writing a new cam table.

This parameter is in counts, and may be negative. It is set for the slave axis, in respect to the units of the master axis (see Figure C-3 below). This parameter must be set before **SYNC**ing the axis to a non-zero mode.

For example, suppose the master axis is X, and the slave axis is Y. Furthermore, suppose that the X-axis has 10,000 counts/inch, and the cam table specifies that at a master position of 3 inches, the slave must be at position "s". If instead, you want a master position of 1 inch to correspond to slave position "s", then set the CAMOFFSET value to (10,000 counts/inch)/(3-1 inches) = 5,000 counts:

CAMOFFSET.Y = 5000

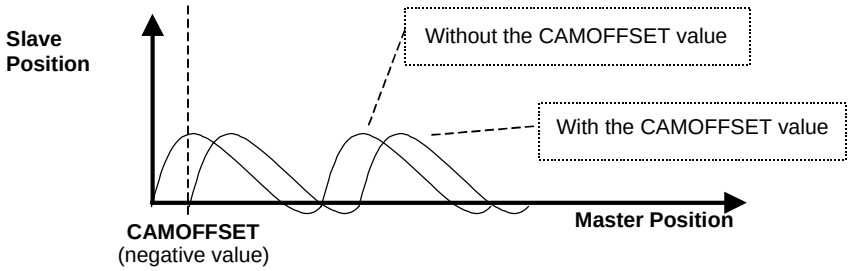


Figure C-3. Camming Illustration

C.2.21. CAMPOINT

This axis parameter is only used for File Driven Camming and HANDWHEEL camming. It is a read only parameter that indicates the point in the cam table producing the slaved axis position. The CAMPOINT axis parameter represents the index of the point in the Cam table that is immediately larger than the current CAMPOSITION value. This parameter should always be examined on the slave axis, and the units are in points in the cam table. It is zero based (1st point in the table is CAMPOINT=0).

C.2.22. CAMPOSITION

This axis parameter is for File Driven Camming and HANDWHEEL camming**.** It is a read only parameter that indicates the master axis position used to index into the cam table to produce he slaved axis position. The CAMPOSITION parameter value is the master position value that is currently being used to perform the cam table lookup. This value is after any modulo position is done on the master position. For example, if the table extends from 1,000 counts to 10,000 counts, and the current position of the master axis is 12,000 counts, this parameter will read 2,000 counts. This parameter should always be examined on the slave axis, and the units are in counts.

C.2.23. CCWEOT

This axis parameter determines the software counterclockwise end-of-travel limit. The controller will not move to a position that exceeds this value. Instead, a CCW_FAULT occurs each time the user attempts to command a position beyond this value. The user must enter the CCW end-of-travel position in machine steps from the home position in the range (not exceeding) 2.1 Billion (±2 E31). The **CCWEOT** should always be more negative than the **CWEOT** parameter.

Be sure to set the CCW Software Limit bit in the **FAULTMASK** axis parameter to enable the detection of this fault, then set the bit in the appropriate mask parameter (**DISABLEMASK**, **HALTMASK**, **AUXMASK**, **ABORTMASK**, **INTMASK** and **BRAKEMASK**) for the action to occur on this fault.

The bounded by software limits flag must be set TRUE within the axis configuration wizard for that axis, in order for soft limits to be active.

Software limits will be ignored until after the axis has been homed, if the **SOFTLIMITMODE** parameter is set to one.

Software limits may not be activated for virtual axes.

C.2.24. CLOCK

This axis parameter is a counter that may be used as a clock with one millisecond resolution. It is constantly incremented by the controller for each axis. It is reset to 0 when the controller's firmware is loaded and can also be reset by the user to any value. Therefore, it can be used as a clock for relative timing purposes. This parameter can be monitored in the AerDebug utility via the ParmMonitor command to verify controller operation. If an error is returned by the command, then the controller is not running its firmware, probably due to a jumper or AerReg (operating system registry) configuration problem. If the parameter is displayed, but is not incrementing, verify that Jumper JP3 is set to position 1-2 (not 2-3) on the *UNIDEX* 600 controller card.

EXAMPLES:

```
$GLOBAL0 = CLOCK.X                          ; Program start time
N001
;
; Run your program here
;
N1000
$GLOBAL0 = (CLOCK.X - $GLOBAL0 ) / 1000     ; global0 now holds elapsed time in
                                            ;  seconds, between line N001 and
                                            ;  N1000
 MSGLAMP1 BLUE $GLOBAL0                     ; This line will display global0
                                            ;  value on the MMI panel
```

C.2.25. CWEOT

This axis parameter determines the software clockwise end-of-travel limit. The controller will not move to a position that exceeds this value. Instead, a CW_FAULT occurs each time the user attempts to command a position beyond this value. The user must enter the CW end-of-travel position in machine steps from the home position in the range (not exceeding) -2.1 Billion (-((±2 E31)-1)). The CWEOT should always be more positive than the **CCWEOT** parameter.

Be sure to set the CW Software Limit bit in the FAULTMASK axis parameter to enable the detection of this fault, then set the bit in the appropriate mask parameter (**DISABLEMASK**, **HALTMASK**, **AUXMASK**, **ABORTMASK**, **INTMASK** and **BRAKEMASK**) for the action to occur on this fault.

The bounded by software limits flag must be set TRUE within the axis configuration wizard for that axis, in order for soft limits to be active.

Software limits will be ignored until after the axis has been homed, if the **SOFTLIMITMODE** parameter is set to one.

Software limits may not be activated for virtual axes.

C.2.26. DACOFFSET

The **DACOFFSET** axis parameter is used to null any offset in the command to the drive, typically used only for axes in the velocity mode (tachometer based systems). Brushless motors utilize the **PHASEAOFFSET** and **PHASEBOFFSET** axis parameters.

The value is entered as a signed number representing the number of D/A counts (+/-) required to bring the DAC output command to zero volts. This value is added to all motor command values before being output to the DAC.

For example, if an axis has a 60 mV (.060 volt) offset, -392 would be entered on a UNIDEX 600/650 for this parameter. This is calculated by dividing the offset by the voltage value of each step of the DAC. UNIDEX 600/650 has a 16-bit DAC (+/- 32767 counts). The voltage per bit is equal to the maximum voltage divided by the maximum counts 20/65,536 = .0003 for a UNIDEX 600/650.

$$.060/(20/2^{16}) = 197 \text{ counts for the UNIDEX } 600/650$$

C.2.27. DECEL

This parameter controls the time that it takes to decelerate the current velocity to a lesser velocity during **G0** (point to point) moves and asynchronous moves while the **DECELMODE** parameter specifies time-based ramping. Deceleration refers to any decrease in velocity. The user may also specify deceleration mode parameters from within a parts program.

This parameter is NOT used for CNC contoured motion (**G1**, **G2, G3**) (refer to *DecelTimeSec* task parameter for ramping contoured moves).

C.2.28. DECELMODE

This parameter allows the user to select the type of ramping used during the deceleration of the axis during **G0** (point to point) moves and asynchronous moves. This ramping may be time-based (using the **DECEL** parameter) or rate-based (using the **DECELRATE** parameter). Also, the user may configure the ramping to be either linear or sinusoidal (1-cosine). Figure C-4 serves as an aid in setting this parameter. The default for this parameter is for a time-based linear ramp.

This parameter is not used for CNC contoured motion (**G1**,**G2**,**G3**) (refer to *DecelTimeSec* task parameter for ramping contoured moves).

0 - Linear Ramping - Time Based
1 - (1-Cosine) Ramping - Time Based
2 - Linear Ramping - Rate Based
3 - (1-Cosine) Ramping - Rate Based

The user may also specify deceleration mode parameters from within a parts program.

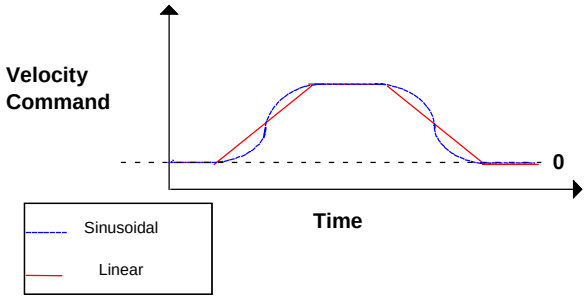


Figure C-4. DECELMODE Ramp Setting

C.2.29. DECELRATE

This axis parameter sets the rate of deceleration during **G0** (point to point) moves and asynchronous moves, while the **DECELMODE** parameter specifies rate-based ramping. You may also specify deceleration parameters from within a parts program. The units are in machine counts per second^2.

This parameter is not used for CNC contoured motion (**G1**, **G2, G3**) (refer to *DecelTimeSec* task parameter for ramping contoured moves).

C.2.30. DISABLEMASK

This axis parameter determines which faults will cause an axis to be disabled. This parameter is a bit mask where each bit corresponds to a specific fault. The **DISABLEMASK** takes priority over the **HALTMASK** and **ABORTMASK**, i.e.; if the **DISABLEMASK** is set to occur, the **HALTMASK** or **ABORTMASK** will have no effect, because the **DISABLEMASK** would disable the axis before it could halt or abort.

Each bit set in this parameter should also be set in the **FAULTMASK** axis parameter, to enable detection of that fault condition.

C.2.31. DRIVE

This axis parameter enables and disables the motor's torque associated with an axis. A zero disables the drive, while a one enables it. This parameter may be used to enable and disable a drive from within a CNC program as shown in the example below.

C.2.31.1. Enabling Drives

Each of the drives may be enabled/disabled by clicking on the axis name at the left side of the position display within the MDI, JOG or Run pages; or via the ENABLE command. The state of the axes drive enable signals is indicated by bit 0 of their STATUS axis parameter.

Redefining the ENABLE command as a Canned Function allows a subroutine to be called whenever the drive is enabled. This is useful for initializing brushless motors without hall-effect feedback sensors present via the MSET command.

EXAMPLE:

```
ENABLE X Y                    ; ENABLE the X and Y drives
```

If your DRIVE won't enable, it could be due to one of several problems.

1. A fault is present (Drive Fault, etc.)
2. You have assigned its DAC (D2A) channel to two or more axes within the Axis Configuration Wizard.

C.2.32. ECHO

This is a "dummy" parameter. It has no effect on the operation of the controller, but may be used by the user for storage of data. It is essentially equivalent to an axis variable.

C.2.33. EXTR2DSCL

The **EXTR2DSCL** axis parameter is used to extend the range of the servo loop parameters for dual loop systems using a resolver for velocity feedback.

For example, if the range of the **PGAIN** parameter is inadequate (i.e., 1 is too little and 2 is too much), increase the EXTR2DSCL parameter to an arbitrary value such as 128. Then increase the encoder lines per revolution parameter within the axis configuration proportionally by 128, (i.e.; enter a value 128 times greater). This will allow a more reasonable range for the setting of the axis servo parameters (**KP**, **KI,** and **PGAIN**).

C.2.34. FAULT

This axis parameter indicates axis faults that have occurred on an axis, since the last time the axis faults were cleared, by reading this parameter. You can also clear axis faults, by writing to this parameter.

When an axis fault occurs, the UNIDEX 600 MMI will flash an error message in the position tracking display of the Run or Manual pages.

This parameter is a bit mask, where each bit corresponds to a specific axis fault that may occur. A zero value indicates that no axis faults have occurred, and since the parameter is bit-mapped, non-zero values for this parameter reflect the occurrence of multiple axis faults. When a fault occurs, the appropriate bit (refer to Table C-3 for Axis Fault Bit Definitions) is 'anded' into the FAULT parameter value, and will remain set until cleared.

The user may clear faults by writing to the FAULT axis parameter. When writing to the FAULT parameter, the UNIDEX 600 attempts to clears all faults whose bits are set in the value written to the parameter. Fault bits set to zero are not cleared. For example, If the FAULT parameter value is "10", the user could write "10" to the FAULT parameter, to clear all active faults (set FAULT back to zero). If the user wrote "8" to the FAULT parameter, it would then read "2." The user should keep in mind that if the condition causing the axis fault is still present, the axis fault will immediately reoccur, so that it will appear as though the fault did not clear.

A better way to view faults, is by using the \U600\Bin\AerStat.exe utility program. This will display all active faults, for all axes, in a convenient graphical format.

The FAULT axis parameter is used to view and clear axis faults. The FAULTMASK and related axis parameters are used to determine the machine behavior for fault conditions.

Table C-3. Axis Faults

Bit	Hex Value	Fault Name	Description
0	0x1	Position Error Limit	Difference between instantaneous commanded position and actual position exceeds the amount specified in the *POSERRLIMIT* parameter.
1	0x2	RMS Current Limit	Average current exceeds the amount specified in the *IAVGLIMIT* parameter averaged over *IAVGTIME* parameter.
2	0x4	CW Hard Limit	The system encountered the CW (clockwise) limit switch. (see *IOLEVEL* parameter)
3	0x8	CCW Hard Limit	The system encountered a CCW (counter clockwise) limit switch. (see *IOLEVEL* parameter)
4	0x10	CW Soft Limit	The user commanded an axis to move beyond the position specified in the *CWEOT* (clockwise end-of-travel) axis parameter.
5	0x20	CCW Soft Limit	The user commanded the axis to move beyond the position specified in the *CCWEOT* (counter-clockwise end-of-travel) axis parameter.
6	0x40	Drive Fault	Drive fault input. (see *IOLEVEL* parameter) However, after clearing the drive fault input, this bit continues to reflect the fact that the fault occurred.
7	0x80	Feedback Fault	Feedback failure input from the feedback associated with the axis. This typically occurs when the feedback device is not functioning properly, or the feedback cable is disconnected.
8	0x100	Programming Fault	Axis processor received an invalid command from the PC host. These only occur when processing programming commands from programs running on the PC (U600MMI, AerDebug). Refer to the *UNIDEX 600 Series Library Reference*, *P /N EDU156* under "Programming errors".
9	0x200	Master Feedback Fault	Feedback failure input from the feedback channel associated with the axis configured as a master. This usually occurs when the feedback device on the master axis is defective, or the cabling is bad.
10	0x400	Home Fault	System encountered a homing fault. This typically occurs for either of two reasons: while executing a home cycle the home limit switch input was not detected; or when the system encounters an end-of-travel limit switch before the first resolver null or marker pulse.
11	0x800	User Fault	Application has requested a fault be generated with the *AerProgSetUserFault( )* function. It provides a way for a programmer to generate an axis fault from within a C/C++ or VB application program.
12	0x1000	Velocity Trap	Actual velocity exceeded the value specified in the *VELTRAP* axis parameter.
13	0x2000	Velocity Command Trap	Instantaneous commanded velocity exceeded the value specified in the *VELCMDTRAP* axis parameter.
14	0x4000	Home Tolerance Fault	Distance traveled from when the system detected the marker pulse (or the Resolver null), until the system encountered the home limit switch is less than the value specified in the *HOMESWITCHTOL* parameter. This occurs during a homing sequence.

Table C-3. Axis Faults (continued)

Bit	Hex Value	Fault Name	Description
15	0x8000	Probe Fault	Occurs each time the probe trigger causes the position to latch. This is useful for notifying the application program that position information is available.
16	0x10000	TaskFault	TaskFault occurred while executing a CNC command running a task. (please see the *TaskFault* Task parameter)
17	0x20000	External Feedback Fault	Difference between the integration of the velocity command and velocity feedback is greater than the *FBWINDOW* axis parameter.
18	0x40000	Safe Zone	*SAFEZONE* axis parameters are active and the axis has violated the defined safe zone.
19	0x80000	Constant Velocity Phase Interrupt	Axis interrupt was generated when move reached constant non-zero velocity (see *INTMASK* Axis parameter).
20	0x100000	Decel Phase Interrupt	Axis interrupt was generated when move reached the decel phase.
21	0x200000	Move Done Interrupt	Axis interrupt was generated when move was done.
22	0x400000	*POSTOGO* interrupt	Axis interrupt was generated when *POSTOGO* passed under the *POSTOGOIRQ* value (see the *POSTOGOIRQ* Axis parameter).
23	0x800000	ESTOP	Emergency stop has occurred (see Section 2.8.)
24	0x1000000	WatchDog	Fail Safe timer
25	0x2000000	Position Tolerance	Axis did not move the distance specified by POSTOLERANCE, within the POSTOLTIME period at the start of the move
26-31			Unused

C.2.35. FAULTMASK

This axis parameter determines which faults the system will detect. The parameter is a bit mask where each bit corresponds to a specific fault. Setting a bit to a one enables monitoring of the fault assigned to that bit. Conversely, clearing a bit causes the system to ignore that fault if it occurs. If a fault is detected, its bit value is "anded" into the FAULT axis parameter value.

Each bit set in this parameter should have a bit set in at least one of the other mask parameters (**DISABLEMASK**, **HALTMASK**, **AUXMASK**, **ABORTMASK**, **INTMASK** and **BRAKEMASK**) to define the action to occur for that fault. If you set a bit in the **FAULTMASK**, but fail to set any corresponding bits in one of the other masks listed above, then the **FAULT** parameter will be set, but no axis action will occur. These actions will be occur immediately after detection of the fault, usually within a millisecond.

You can also trigger program-related actions to take place when an axis fault occurs, with the **TaskFault** task parameter. CNC programs may be stopped when by axis faults, via the **HaltTaskOnAxisFault** task parameter.

C.2.36. FBWINDOW

When running an axis in dual (secondary feedback present) loop mode, the servo loop integrates the velocity feedback from the secondary feedback device, to calculate the **VELPOSITION** (position at the motor). The **FBWINDOW** parameter allows the user to specify the maximum amount by which the motor position, and the load position (**POS** axis parameter, as derived from the primary feedback device), may differ. An EXTERNAL FEEDBACK fault occurs if the absolute value of that difference exceeds the amount specified by this parameter. A value of zero disables this monitoring. For an axis fault to be generated by this condition, the 'External Feedback Fault' bit must be set within the **FAULTMASK** axis parameter also. The units of this parameter are counts.

C.2.37. FEEDRATEMODE

This axis parameter determines if the axis is subject to feedrate override control while executing motion from the *AerMotn* library function calls, not CNC motion. Setting this parameter to zero disables the feedrate override control, while a value other than zero makes the axis subject to feedrate override. The default value is zero (0). If the *FEEDRATEMODE* is not zero, then the feedrate override control is expected over the analog input channel 1 on UNIDEX 600/650 systems.

This feedrate override control effects only motion generated from the *AerMotnxxx* library function calls and not CNC generated motion.

C.2.38. GANTRYMODE

This axis parameter is set for the master axis and represents the number of the slave axis. The gantry mode utilizes the **AuxVelCmd** axis parameter for commanding motion on the slave axis, so it should not be used while the gantry mode is active.

0	Gantry Mode is off (default)
1-16	Gantry Mode 1
17-32	Gantry Mode 2

C.2.38.1. Configuring GANTRYMODE

1. Align the Gantry so that it can be run.
2. Set GANTRYMODE to mode 2
3. Home the master axis. The slave will follow.
4. Set GANTRYOFFSET correctly (to remove skew and to set the distance away from the marker)
5. Set GANTRYMODE to mode 1

C.2.39. GANTRYOFFSET

This axis parameter is set for the master axis and represents the distance the slave axis should be away from the marker in machine steps.

C.2.40. GEARMASTER

This axis parameter along with the **GEARSLAVE** axis parameter is used for gearing. It defines the ratio of movement between the master and slave axes. This parameter is set for the salve axis, not the master. This parameter is automatically set by the TRACK command (tracking mode), or it may be set manually in conjunction with the gearing mode (by setting the **GEARMODE** axis parameter to one). It may also be set when the tracking mode is active to change the gear ratio on the fly.

C.2.41. GEARSLAVE

This axis parameter is used along with the **GEARMASTER** axis parameter for gearing. The ratio of which defines the movement between the master and slave axes. This parameter is set for the slave axis, not the master. This parameter is automatically set by the TRACK command (the tracking mode), or it may be set manually in conjunction with the gearing mode (by setting the **GEARMODE** parameter to one). It may also be set when the tracking mode is active to change the gear ratio on the fly.

C.2.42. GEARMODE

The GEARMODE axis parameter will enable and disable electronic gearing. Electronic gearing is a form of camming motion where a slave axis moves at a ratio of a master axis movement. Setting this parameter to 1 (on the slave axis) will immediately enable electronic gearing. Setting it to zero (on the slave axis) will disable electronic gearing. Gearing motion requires that you have previously defined a master and a slave axis using the **CFGMASTER** command. See Figure C-3 for additional information.

Once gearing is enabled, the slave axis will always travel at the gear ratio of the master axis speed, even as the master axis speed changes. The gear ratio is as follows:

Slave speed (counts/sec) = Master speed (counts/sec) * (GEARSLAVE / GEARMASTER)

Where, the **GEARSLAVE** and **GEARMASTER** axis parameter values in the formula above are those of the slave axis. In gearing, the actual values of these two parameters are not relevant, only their ratio is. The programmer should be aware that the speeds above are in machine-counts/sec, not user-units. If the CntsPerInch / CntsPerDeg parameters vary for the two axes, then the speed ratio will be different when viewed in user-units: (the formula below assumes both axes are linear Type axes).

Slave speed (user-units/sec) = Master speed (user-units/sec) *
(GEARSLAVE * CntsPerInch-of-Master/GEARMASTER *CntsPerInch-of-Slave)

The user must be warned that when gearing is enabled, the slave axis will immediately try to follow the master speed. So the master axis should not be moving at the time electronic gearing is enabled, or the slave axis will instantaneously try to move at the specified speed (see above formula) thereby jerking the slave axis. If you must establish gearing while the master axis is moving, see the TRACK command, which allows you to establish gearing while simultaneously blending in an acceleration for the slave axis. See the CFGMASTER command for an example program.

If you repeatedly enable and disable the GEARMODE, you may have to set the MASTERPOS axis parameter to prevent 32-bit over runs and resultant jerking motion of the slave axis.

C.2.43. HALTMASK

This axis parameter defines the fault conditions that cause the axis to halt. The value specified for this parameter is a bit mask where each bit corresponds to a specific fault. The axis will halt if the bits for **FAULTMASK** and **HALTMASK** are set to one for any given bit position. In halting motion, the axis will decelerate to zero velocity based on the time/rate specified in its deceleration axis parameters. Setting a bit to a one halts the axis when that particular fault occurs (assuming the corresponding bit in the **FAULTMASK** parameter is set). This parameter has no effect on the position error tracking. If an axis is triggered by a fault condition to abort and halt simultaneously, the abort takes priority. Each bit set in this parameter should also be set in the **FAULTMASK** axis parameter, to enable detection of that fault condition. The **DISABLEMASK** takes priority over the **HALTMASK** and **ABORTMASK**, i.e.; if the **DISABLEMASK** is set to occur, the **HALTMASK** or **ABORTMASK** will have no effect, because the **DISABLEMASK** would disable the axis before it could halt or abort.

C.2.44. HOMEOFFSET

This axis parameter is the home offset in machine steps (counts). In other words it is the value that will be loaded into the machine position registers at the completion of the home cycle. It will not cause movement from the marker pulse (or resolver null position). The home position will be set to this value. To be at zero following the home cycle will then require an absolute move to zero.

When using the CNC (UNIDEX 600 MMI), the HomeOffset task parameter is used. However, for Library invoked homing, the HOMEOFFSET axis parameter is used. If using both CNC and Library interfaces simultaneously, the user must use the HomeOffset task parameters.

However, it is overwritten by the HomeOffsetInch (or HomeOffsetDeg for rotary axis) machine parameters, after conversion to machine steps. This parameter is the value that will be loaded into the machine position registers at the completion of the home cycle, so this parameter should not be set by the user, as it will be overwritten.

C.2.45. HOMESWITCHPOS

This axis parameter indicates the resolver value when the home limit switch is encountered.

This parameter is valid only for axis using resolver based feedback.

C.2.46. HOMESWITCHTOL

To ensure the accuracy of a resolver based homing sequence, there must be a minimum distance between the Home Limit Switch and the resolver null position. Otherwise, the controller might miss the first resolver null and use the second as the home position (before the Home Offset). The required distance depends on two factors: feedback resolution and home feedrate. This parameter specifies the minimum distance in machine steps that must exist between the home limit and the resolver null. Failure to maintain this distance causes a *HOME_SWITCH_TOLERANCE* fault to occur.

This parameter is valid only for axis using resolver based feedback.

Be sure to set the Home Switch Tolerance bit in the **FAULTMASK** axis parameter to enable the detection of this fault, then set the bit in the appropriate mask parameter (**DISABLEMASK**, **HALTMASK**, **AUXMASK**, **ABORTMASK**, **INTMASK** and **BRAKEMASK**) for the action to occur on this fault.

C.2.47. HOMEVELMULT

This axis parameter is used to scale the velocity during the home cycle after the home limit has been found, while the marker (or resolver null) search is in progress. This allows a high speed search for the home limit and a slower speed search for the marker, providing a quick and accurate home cycle. For example, a value of 1 represents 1% of the commanded home velocity (HomeFeedRateIPM or HomeFeedRateRPM task parameters) during the marker search portion of the home cycle. The default value of 100 represents 100% of the commanded home velocity during the marker search portion of the home cycle.

C.2.48. IAVG

The IAVG axis parameter is the actual current of the axis averaged over a time determined by the **IAVGTIME** parameter. Units are in D/A (DAC) counts, where ± 32,767 represents ±10 volts out of the D/A, which translates into +/- peak current by the amplifier. This parameter is a read-only parameter.

C.2.49. IAVGLIMIT

This axis parameter detects an over current condition based upon the setting of the **IAVGTIME** parameter. The value specified in the **IAVGTIME** parameter determines what time period to average the instantaneous current (ICMD). An RMS current limit fault occurs if the RMS average exceeds the limit set by this parameter. As with the **IMAX** parameter, the range of this parameter is 0 to 32,767, where 10 volts is represented by the value 32,767. To set this parameter, determine the command voltage that the amplifier requires to produce the desired maximum RMS motor current. IAVGLIMIT = (MaxV /10)* 32767, where MaxV is the voltage into the amplifier producing the peak of the desired maximum RMS current level.

For example, the torque applied to the motor (in torque mode), may be easily calculated for brush or brushless motors, knowing a few parameter values and the K_T (motor torque constant). The UNIDEX 600\650 Controllers have a 16 bit Digital-to-Analog converter used to convert a signed 16 bit number (+32,767 through -32,767) to an analog voltage in

the range of +10 volts through –10 volts. This voltage is applied to the command input of the servo amplifier, knowing the G_m (transconductance) value of the servo amplifier allows the current (Amperes) output to be calculated. Aerotech's amplifiers typically (model dependant) have a peak output current of 20 or 30 amperes. Simply meaning, a +/- 10 volts in to the amplifier will be equal to +/- *x* amperes, where *x* is the peak output current of the servo amplifier. This current produces a torque generated by the motor, equal to the current multiplied by the K_T of the motor. For example, if the peak output command (or instantaneous output command) from the DAC was 16,384 (+5 volts), the amplifier produced 30 amperes for an input of 10 volts and the motor had a K_T of 16 oz.in. (Ounce-inches) per ampere, then:

(5 volt command / 10 volt, max. amps, input command) * 30 Amperes (max. I. Output) * 16 oz.in K_T = 240

or

(5/10) * 30 * 16 = 240

Therefore, the applied torque is 240 oz.in.

To calculate this value for an Aerotech brushless motor, from the motor data, use the "continuous stall current max" (A peak) specified for the motor, **or** use the "continuous stall current" (A RMS) specified after converting it to Peak by multiplying it by 1.414

To calculate this value for an Aerotech DC brush motor, from the motor data, use the RMS current specified for the motor, **or** divide the continuous toque (Tc) by the motor torque constant (Kt); i.e.,

Tcont / Kt = RMS value

Be sure to set the RMS Current Limit bit in the **FAULTMASK** axis parameter to enable the detection of this fault, then set the bit in the appropriate mask parameter (**DISABLEMASK**, **HALTMASK**, **AUXMASK**, **ABORTMASK**, **INTMASK** and **BRAKEMASK**) for the action to occur on this fault.

C.2.50. IAVGTIME

This axis parameter defines the time period over which the system will average the instantaneous commanded current. However, this parameter is only used to next lowest 10 milliseconds interval. For example, a value of 12 or 19 is equivalent to a value of 10. The **IAVGLIMIT** parameter is dependent on the setting of this parameter to detect an over-current condition.

C.2.51. ICMD

The ICMD axis parameter returns the instantaneous current command, where ±32767 represents ±10 volts. This voltage is the command applied to the drive module.

C.2.52. ICMDPOLARITY

The ICMDPOLARITY axis parameter is used for two purposes. The first is to vary the polarity of the current command output by the servo loop, which inverts the current command polarity that is output from the DAC to the drive. On a UNIDEX 600/650, a positive current command should generate negative axis motion. This parameter is normally set to 0 for no inversion, it may be set to 1 to correct for reversed servo loop phasing, not to reverse the direction of the motor on a properly phased servo loop. Also, if

using a dual feedback loop, the velocity and position loop must be in phase (with each other). Otherwise, toggling this parameter would correct one feedback element and improperly phase the other.

Additionally, a unipolar command may be output by the servo loop for a unidirectional spindle motor (if required), by setting this parameter to -1. The status of this command is indicated by the " invert polarity" bit of the **ALT_STATUS** axis parameter. A value of 1 indicates that the **ICMDPOLARITY** value is 1.

C.2.53. IMAX

This axis parameter sets the peak commanded output current, when the axis is operating in the current (torque) mode. This is done by limiting the maximum output voltage of the current command signal that is in turn translated into a proportional motor current by the drive module. 10 volts is represented by 32,767. To set this parameter, determine the maximum input command voltage that the amplifier requires to produce the maximum desired motor current. The default value of this parameter is 32,767, producing a 10-volt current command signal that would command the maximum current from the drive module.

For brushless motors this parameter is set to the peak value of the desired sinusoidal output current.

This parameter could also be used to limit the maximum commanded velocity in the velocity command mode, but is not normally used for this purpose due to the **VELTRAP** and **VELCMDTRAP** axis parameters.

See the **IAVGLIMIT** axis parameter for an example on calculating the instantaneous torque produced by the motor.

C.2.53.1. Computing Torque (Closed-Loop Torque Mode)

The torque applied to the motor (in torque mode) may be easily calculated for brush or brushless motors if you know a few parameter values and the K_t (motor torque constant). The UNIDEX 600/650 Controllers have a 16 bit Digital-to-Analog converter used to convert a signed 16 bit number (+32,767 through –32,767) to an analog voltage in the range of +10 volts through –10 volts. This voltage is applied to the command input of the servo amplifier (knowing the G_m (transconductance) value of the servo amplifier allows the current (Amps) output to be calculated). Aerotech's amplifiers typically (model dependant) have a peak output of 20 or 30 amps. Simply meaning, a +/- 10 volts in to the amplifier will be equal to +/- *x* amps, where *x* is the peak output current of the servo amplifier. This current produces a torque generated by the motor, equal to the current multiplied by the K_t of the motor. For example, if the peak output command (or instantaneous output command) from DAC was 16,385 (+5 volts), the amplifier produced 30 amps for an input of 10 volts, and the motor had a K_t of 16 oz-in (ounce-inches) per amp, then:

(5 volt command / 10 volt, max amps input command) * 30 amps (max I output) * 16 oz-in K_t = 240

or

5/10 * 30 *16 = 240

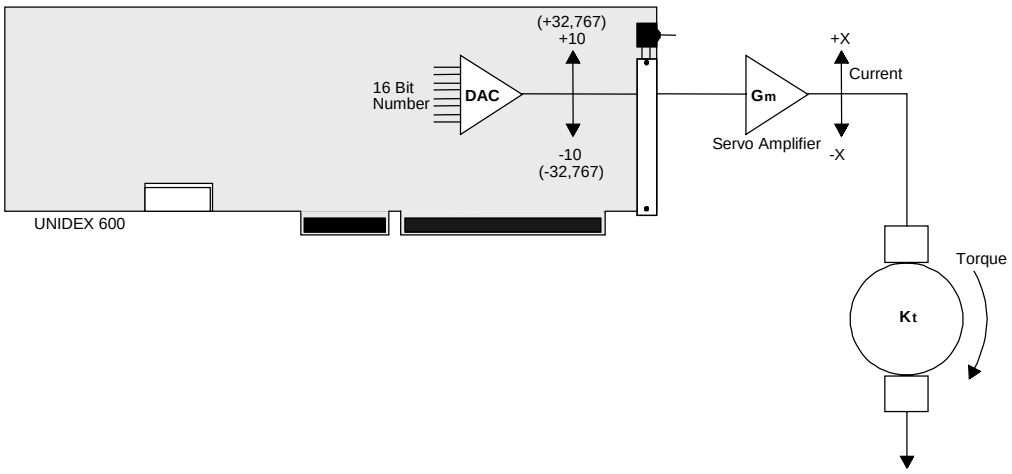


Figure C-5. Closed-Loop Torque Mode

C.2.54. INPOSLIMIT

This axis parameter allows the user to define the in-position band. If the axis has completed its move and the observed position error is within the range determined by the in-position (plus or minus) set by this parameter, the axis in-position bit within the **STATUS** axis parameter become active. The units of this parameter are machine steps.

In the **G361** mode, the following types of motion commands will wait until the velocity command is zero, AND until the axis is within the in-position band, before proceeding onto the next command.

G0

G1 / **G2** / **G3**, with a **G9** on the CNC program line

G1 / **G2** / **G3**, when **G109** mode is active

G1 / **G2** / **G3**, when CNC Block Look-Ahead enforces a **G9** at the end of a move

ENDM

Furthermore, the **PSOx** commands will wait until the in-position band is true, before starting that command.

C.2.55. INTMASK

This axis parameter allows the user to determine which fault conditions will cause an interrupt to be generated back to the host. This parameter is a bit mask where each bit corresponds to a specific fault. An interrupt is generated if the bits for INTMASK and **FAULTMASK** are set to one for a given bit position, when that fault occurs. Therefore, each bit set in this parameter should also be set in the **FAULTMASK** axis parameter, to enable detection of that fault condition.

Interrupts will only be generated for new axis faults, that is, the controller will only generate an interrupt once for each occurrence of a particular axis fault.

C.2.56. IOLEVEL

This axis parameter allows the user to specify the active state for the axis and drive interface signals. The user may configure the active state of the following signals.

0.	Drive Shutdown	(output to drive)	0x1
1.	AUX (Mode) Output	(output to drive)	0x2
2.	CW limit Switch	(input from drive)	0x4
3.	CCW limit Switch	(input from drive)	0x8
4.	Home limit Switch	(input from drive)	0x10
5.	Drive Fault	(input from drive)	0x20

The value specified is a bit-mask where only the specified bits are valid. Setting a bit to one implies the input or output is active high. Refer to Section 2.5. Drive Signals in the Users Guide for more information.

The easiest way to configure these signals via the **IOLEVEL** axis parameter is to view the state of the signals via the AerStat.exe utility. Knowing the state of the signal and viewing the state via AerStat will allow you to toggle the appropriate bits in the IOLEVEL axis parameter to correct the state.

Be sure to set the CW, CCW Limit and Drive Fault bits in the **FAULTMASK** axis parameter to enable the detection of these faults, then set the bits in the appropriate mask parameters (**DISABLEMASK**, **HALTMASK**, **AUXMASK**, **ABORTMASK**, **INTMASK** and **BRAKEMASK**) for the actions to occur on these faults.

C.2.57. IVEL

The **IVEL** parameter returns the instantaneous commanded velocity, in counts per second.

C.2.58. KI

This axis parameter sets the integral gain of the velocity loop for the selected axis. Refer to the UNIDEX 600 Hardware Manual for a description of how this parameter functions in the servo loop. Note, that you may use the AutoTune feature, within the AerTune.exe utility (on the Tools menu) to automatically determine servo loop gains for a torque mode axis. For axis with a tachometer (velocity command mode), set this parameter to 0.

C.2.59. KP

This axis parameter sets the proportional gain of the velocity loop for the selected axis. Refer to the UNIDEX 600 Hardware Manual for a description of how this parameter functions in the servo loop. Note, that you may use the AutoTune feature, within the AerTune.exe utility (on the Tools menu) to automatically determine servo loop gains for a torque mode axis. For axis with a tachometer (velocity command mode), set this parameter to 0.

C.2.60. MASTERLEN

This axis parameter is used only in the context of Camming (master/slave motion). It is used to modulo the **MASTERPOS** axis parameter to avoid overruns of the 32-bit **MASTERPOS** value. It is set in counts, for the slave axis. It is used only when the master is a rotary axis, and the master axis will be cycled through multiple revolutions while synchronized.

When cycling a rotary axis through multiple revolutions, the controller modulo's the position to zero when it reaches 360 degrees. The **MASTERLEN** axis parameter serves the same purpose for the MASTERPOS axis parameter. The **MASTERLEN** axis parameter must be set to the number of counts per 360 degrees, to avoid overruns of the 32-bit counter. For example, if axis A is a rotary master axis with 1000 counts/deg mastering the X axis, then set MASTERLEN.X=360000.

This value must be zero, or an integer multiple of the range of master positions spanned by the cam table. Otherwise, when it modulo's the position, it will incorrectly index into the cam table and violently jerk the slave axes.

C.2.61. MASTERPOS

This axis parameter is only used for Camming (master/slave motion). This axis parameter is set for the slave axis. It is the position of the master axis in counts, as seen by the slave. It is the actual value the slave axis uses, in conjunction with the cam table, to find its slave position. Usually, it is equal to position of the master, and can be ignored. However, there are two situations where they may vary:

1. If the master axis has a higher task axis index than the slave (see Section C.2.61.1., below)
2. Prior to Synchronization of the axis (see description below).

If a slave axis is in a non-zero **SYNC** mode, then the **MASTERPOS** value of the slave axis is "read-only", because it is being continuously updated by the controller, which is integrating the master axis velocity. However, if a slave axis is in sync mode 0, then the **MASTERPOS** value of the slave is read/write, and does not track the master position. Therefore, when a slave is synced up, changes in position of the master axis are tracked accurately, but the initial master position value is undetermined. For example, suppose after power-up, a camming action occurs which brings the master to 1000 counts. Then suppose camming is disabled, and the master axis is moved back to 0 counts. During this latter move **MASTERPOS** is not updated, so it still reads 1000 counts, even though the actual position of the master axis is 0 counts. As explained above, **MASTERPOS** is not set on syncing up, so **MASTERPOS** now has a 1000 count offset. If this is done repeatedly, it can eventually lead to overruns of the 32-bit **MASTERPOS** value, which leads to violent jerks of the slave axis. Furthermore, it can lead to erroneous motion when syncing in mode 2.

Therefore the following line is always recommended prior to synchronization.

```
MASTERPOS.X = POS.Y          ; where X is the slave, Y is the master
```

Or, if you are tracking Position command on the master:

```
MASTERPOS.X = POSCMD.Y      ; where X is the slave, Y is the master
```

This should only be set while the master axis and slave axes are not synchronized, or you may jerk the slave axis.

In a related point, if the master is a rotational axis you should also set the **MASTERLEN** axis parameter to the number of counts per rotation, to avoid **MASTERPOS** overruns.

C.2.61.1. Master Axis Selection

At least two axes are involved in any master/slave motion, the axis whose motion is dependent upon the other, is the slave axis and a user specified master axis is used to command the slave. The master axis may be an existing axis, a handwheel, or a virtual axis. If you want to base axes motion on another axes motion, clearly the master axes needs to be a real axes, or all slave motion must be pre-computed against time, with a virtual axis being the master, commanded at a constant velocity, synchronizing all slave axes. However, if all you want to do is provide a velocity or distance profile to a single axis, then your master should be configured as virtual.

It is recommended, where possible, that you select your axes such that the master axis has a smaller axis index then the slave. For example, X as the master and Y as the slave axis is preferred, than vise versa. This is because when the master axis has a higher axis index than the slave, and the master axis is not moving at a constant speed, a slight mis-tracking will exist. The **MASTERPOS** read by the slave is not equal to the position of the master. This "following error" is equal to the acceleration of the master (in cnts/msec squared) and is usually only a few counts.

Finally, if the master is a rotary Type axis, you must consider setting the **MASTERLEN** axis parameter to avoid 32-bit overruns of the **MASTERPOS**.

C.2.62. MASTERRES

The master resolver axis parameter is the current resolver position (0-65535) for the axes master axis, if the axis has been configured to track a HANDWHEEL or as a slave to a master with resolver feedback. It is "0" otherwise.

C.2.63. MAX_PHASE

Refer to the **BASE_SPEED** axis parameter for a description of this parameter.

C.2.64. MAXCAMACCEL

This axis parameter is used only in the context of File Driven Camming, only when **SYNC** mode 3 is used. This parameter is most useful when synching on the fly (switching cam tables without de**SYNC**ing in between). If this axis parameter is non-zero, it will limit the acceleration of the slave axis to its value, which is in user units/sec/sec. To deactivate this feature, set the parameter value to 0.

C.2.65. MOTIONSTATUS

This is a bitwise axis parameter, providing the current status of the motion on this axis. Also, be aware that some motion conditions may also be reported in the **STATUS**, **Status1**, **Status2**, **Status3,** and **SERVOSTATUS** parameters. There are pre-defined definitions of the bits in this axis parameter within AerStat.Pgm. These definitions take the form of "MOTIONSTATUS_xxx" where xxx is the bit name as listed below. For example, the following could be used to test for the X axis Moving bit being TRUE;

If (MOTIONSTATUS.X BAND MOTIONSTATUS_Moving)
Or
If (MOTIONSTATUS.X BAND 0h00000002)

Table C-4. MOTIONSTATUS Bit Definitions

Name	Hex Value	Description
MOVE_DIR	0h00000001	move direction
MOVING	0h00000002	G0, Home or Async. Motion Active
ACCEL	0h00000004	axis in accel. Phase of G0, Home or Async. Motion
DECEL	0h00000008	axis in deceleration phase of G0, Home or Async. Motion
HOMING	0h00000010	axis in home cycle
FEED_OVER	0h00000020	feedrate override
PROFILING (NOTE)	0h00000040	axis in profiling mode (G1, G2, G3 or other contoured motion)
SYNC	0h00000080	axis in sync mode (camming motion)
CAM_TABLE	0h00000100	cam table enabled
HOME_DIR	0h00000200	home direction
CONT_MOVE	0h00000400	continuous move (HOME or STRM, ...
QUEUE	0h00000800	motion queue active
HOLD	0h00001000	hold active
AUX_MODE	0h00002000	aux mode
BLOCK	0h00004000	block motion
HOLD_QUEUE	0h00008000	hold queue
JOG	0h00010000	Jog mode active
DISABLE	0h00010000	disable command
HALT	0h00020000	halt command
ABORT	0h00040000	abort command
ACCEL_ON	0h00080000	Acceleration command
DECEL_ON	0h00100000	Deceleration enabled
ACCEL_SIGN	0h00200000	Acceleration sign used for direction change
CONST_ACCEL	0h00400000	linear/1-cosine acceleration flag
CONST_DECEL	0h00800000	linear/1-cosine deceleration flag
BOUNDED	0h01000000	bounded i.e., use software limits
SETUP_PEND	0h02000000	setup command pending
CHCKR_FLAG	0h04000000	set along with setup_pend & cleared when checker runs
QUICK_HOME	0h08000000	quick home active
IRQ_PENDING	0h10000000	interrupt pending
PENDANT_JOG	0h20000000	pendant jog mode active
MRKR_ARMED	0h40000000	marker armed
Jog Mode Enabled	0h80000000	Jog mode is enabled

C.2.65.1. Profiling Bit in the MOTIONSTATUS Axis Parameter

The profiling bit in the **MOTIONSTATUS** axis parameter indicates that the axis is in the profiling mode (**G1**, **G2**, **G3** or other contoured motion). This bit will be TRUE even during a manual feedhold or when the MFO = 0, however, it will be false during a feedhold due to Jog/Interrupt, ONGOSUB, or a canned function.

C.2.65.2. Moving Bit in the MOTIONSTATUS Axis Parameter

The moving bit in the **MOTIONSTATUS** axis parameter indicates that the axis is in the profiling mode (**G0**, **HOME** or other contoured motion). This bit will be TRUE, even during a manual feedhold or when the MFO = 0, however, it will be false during a feedhold due to Jog/Interrupt, ONGOSUB, or a canned function.

C.2.66. MOVEQDEPTH

The move queue depth parameter indicates how many commands are waiting to be executed from that axis motion command queue.

C.2.67. MOVEQSIZE

The move queue size parameter indicates the maximum number of motion commands that are permitted in the queue.

C.2.68. PGAIN

This axis parameter determines the gain of the position loop for the selected axis. Refer to the UNIDEX 600 Hardware Manual for a description of how this parameter functions in the servo loop. Note, that you may use the AutoTune feature, within the AerTune.exe utility (on the Tools menu) to automatically determine servo loop gains for a torque mode axis.

If the **PGAIN** axis parameter can not be increased greater than 1, without producing an instability in the axis, its units may be rescaled.

C.2.69. PHASE_SPEED

Refer to the **BASE_SPEED** axis parameter for a description of this parameter.

C.2.70. PHASEAOFFSET

This axis parameter is used with the **PHASEBOFFSET** axis parameter to null any offset in the DAC's (commands) output to the drive for phase A (and Phase B) of a brushless motor. Brush motors utilize the **DACOFFSET** axis parameter.

The value is entered as a signed number representing the number of D/A counts (+/-) required to bring the DAC output command to zero volts. This value is added to all motor command values before being output to the DAC.

For example, if an axis has a 60 mV (.060 volt) offset, -392 would be entered for this parameter. This is calculated by dividing the offset by the voltage value of each step of the DAC. UNIDEX 600 has a 16-bit DAC (+/- 32767 counts), generating a +/- 10-volt

output from the DAC. The voltage per bit is equal to the maximum voltage divided by the maximum counts; 20/65,536 = .0003

$$.060/(20/2^{16}) = 197 \text{ counts for the UNIDEX 600}$$

C.2.71. PHASEBOFFSET

This axis parameter is used with the **PHASEAOFFSET** axis parameter to null any offset in the DAC's (commands) output to the drive for phase B (and Phase A) of a brushless motor. Brush motors utilize the **DACOFFSET** axis parameter.

The value is entered as a signed number representing the number of D/A counts (+/-) required to bring the DAC output command to zero volts. This value is added to all motor command values before being output to the DAC.

For example, if an axis has a 60 mV (.060 volt) offset, -392 would be entered for this parameter. This is calculated by dividing the offset by the voltage value of each step of the DAC. UNIDEX 600 has a 16-bit DAC (+/- 32767 counts), generating a +/- 10-volt output from the DAC. The voltage per bit is equal to the maximum voltage divided by the maximum counts; 20/65,536 = .0003

$$.060/(20/2^{16}) = 197 \text{ counts for the UNIDEX 600}$$

C.2.72. POS

This axis parameter specifies the observed (as viewed from the primary feedback device) position of the axis in machine counts. This parameter is a signed 32-bit integer having a range of ±2 E31 - 1 (approximately +/- 2.1 billion). Upon initialization, the system sets the current observed position (**POSCMD**) equal to the **POS** parameter value. At the completion of a homing cycle, this parameter is set to the value of the **HOMEOFFSET** axis parameter.

Normally, you would read this parameter, however, you may also set this parameter to a value, which will not result in any axis motion, because the position command will be set to the same value.

Setting this parameter will clear the "homed" condition of the axis (if it was previously homed).

The PositionUnits machine parameter provides the actual axis position in user units.

C.2.73. POSCMD

The position command axis parameter is the current value of the position command, in units of machine counts. In other words, this is the position the axis is being commanded to move to. The actual position **POS** may differ at any given time, based on the mechanics of the system. The **PositionCmdUnits** machine parameter provides the command position in user units.

C.2.74. POSERR

The position error is the instantaneous difference, in counts, between the commanded position (**POSCMD**) and the actual position (**POS**). *POSERR = (POSCMD-POS)*; this axis parameter is continuously updated on each servo loop cycle.

C.2.75. POSERRLIMIT

This axis parameter determines the maximum absolute position error that can occur on an axis before it generates a position error limit axis fault. The units of this axis parameter are machine counts.

Be sure to set the Position Error Limit bit in the **FAULTMASK** axis parameter to enable the detection of this fault, then set the bit in the appropriate mask parameter (**DISABLEMASK**, **HALTMASK**, **AUXMASK**, **ABORTMASK**, **INTMASK** and **BRAKEMASK**) for the action to occur on this fault.

C.2.76. POSTARGET

The position target axis parameter is the position targeted by the servo loop for a particular move. This parameter is updated only when a move begins. If the move is a free run type move with no target (similar to the STRM CNC command), this parameter is not updated, also, this parameter will be zero during contoured motion (**G1**, **G2**, **G3**, etc.) or Cam table motion, since it is not used during contoured motion.

C.2.77. POSTOGO

The position to go axis parameter indicates the number of counts remaining for the current move. For example, if the current position of the axis is zero and an *Absolute* move commanded a position of 10,000, the POSition TO GO would indicate 10,000. Also, if the current position was 1,000 (**POS** parameter = 1,000), the *POSTOGO* parameter would indicate 9,000. This indicates the commanded distance left to go in the current move, without regard to the current position error of the axis. Also, this is always a non-positive (absolute) value. This parameter will be zero during contoured motion (**G1**, **G2**, **G3**, etc.) or Cam table motion, since it is not used during contoured motion, only **G0** and Asynchronous motion.

C.2.78. POSTOGOIRQ

The user can configure the UNIDEX 600 Series controller to generate an interrupt based on the position to go left in a move. The POSTOGOIRQ axis parameter specifies the distance from the end of the commanded move where the interrupt will be generated. This interrupt is based on the commanded position to go and does not take into account the position error of the axis. To enable this function, set the POSTOGOIRQ axis parameter with the desired value and set the POSTOGO bit (VB/C programmers, see AERDEF.H for bit definitions) in the INTMASK axis parameter. The interrupt is identical to those generated for axis faults. Therefore, the fault handling code should verify whether an axis fault condition is present or whether a POSTOGOIRQ interrupt has occurred and react appropriately. The POSTOGOIRQ interrupt is only activated after an axis has been homed.

Be sure to set the PosToGo Interrupt bit in the INTMASK for the interrupt to occur on this fault. This bit has no purpose within all other faultmasks. Additionally, this fault is generated only by non-contoured, non-cam motion, such as, that produced by the G0, and HOME and Asynchronous motion commands.

C.2.79. POSTOLERANCE

The POSTOLERANCE axis parameter is used to trigger the Position Tolerance FAULT. If the axis moves less than twice the POSTOLERANCE value within **POSTOLTIME** milliseconds, the fault will occur.

This is useful in situations where an axis is commanded to move a distance less than the **POSERRLIMIT** and there is a long IAVGTIME specified. This prevents the axis from running away before the IAVGLIMIT fault occurs. The move length limitation prevents spurious faults from occurring on moves that are shorter then the **POSTOLERANCE** value or axes which are moving at slow velocities. This fault can not occur on virtual axes.

C.2.80. POSTOLTIME

The **POSTOLTIME** axis parameter specifies the time in milliseconds before the **POSTOLERANCE** axis parameter generates an axis fault.

C.2.81. PROFILETIME

The PROFILETIME axis parameter is used only in the profile mode of trajectory generation, available only to those users programming the machine via the library functions. It will define the execution time of each profile block, within the profile mode.

See the Library Reference Manual (EDU156) for more information.

C.2.82. PROFQDEPTH

The profile queue depth parameter indicates the number of motion profile commands in the queue that are waiting to be executed by the axis.

C.2.83. PROFQSIZE

The profile queue size parameter indicates the maximum number of motion profile commands permitted in the motion profile queue.

C.2.84. RAWPOS

Same as the **POS** axis parameter, except this parameter indicates the change in axis position made by Axis Calibration or 2D Axis Calibration.

C.2.85. RESOLVER

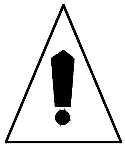
The resolver axis parameter indicates the absolute position of the axis as reported by the resolver, if present, not the position of the master axis resolver.

C.2.86. REVERSALMODE

To provide greater positioning accuracy, this axis parameter allows you to specify the number of machine steps (counts) required to compensate for any backlash present in the system. Backlash, is the "play" or "slop" in the mechanics, and occurs when a drive screw changes direction and turns a fixed amount before the load begins to actually move in the

new direction. This parameter specifies the distance in machine steps before the actual stage movement. Setting this parameter to zero counts disables backlash compensation.

Backlash compensation has no effect on virtual axes and axes that have not been homed. Once the axis is homed, the compensation value will be applied when the axis first moves in the direction opposite of the last move in the home cycle.



When compensation is added, the compensation will be added to the position command in the first millisecond of the move, that is the commanded speed of the motion during compensation will be equal to the reversal value number of counts per millisecond, with an instantaneous commanded acceleration and deceleration. This should not be a problem, since reversal values are normally small, i.e. only a few counts.

The compensation will not show up in the position command, raw position, or position error values. It is simply added on to the position value. The **REVERSALVALUE** axis parameter shows the current amount of compensation added to the position value.

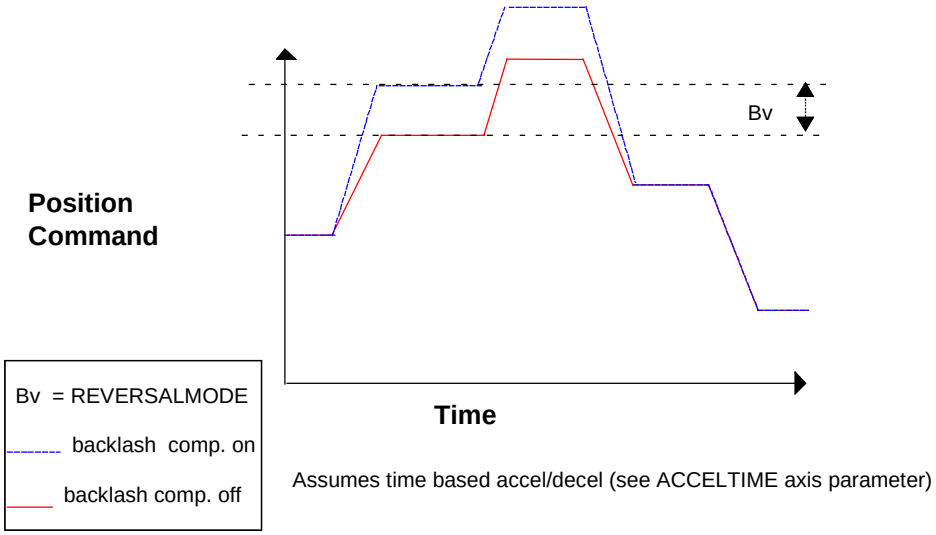


Figure C-6. REVERSALMODE Accuracy Position

C.2.87. REVERSALVALUE

This axis parameter is the current correction value (in machine counts) for the reversal mode (backlash compensation), output in the current direction. This parameter is read only.

C.2.88. SAFEZONECCW

This axis parameter allows the user to specify the counter-clockwise boundary for the safe zone of an axis in machine steps. Safe zones are useful for designating an area in which the axis can travel or one in which the axis can not travel. To enable or disable these zones. The user must set the **SAFEZONEMODE** axis parameter. Safe zones may be enabled/disabled during homing, by the **SOFTLIMITMODE** axis parameter. The distance specified by this parameter begins at the hardware home position.

Be sure to set the Safe Zone bit in the **FAULTMASK** axis parameter to enable the detection of this fault, then set the bit in the appropriate mask parameter (**DISABLEMASK**, **HALTMASK**, **AUXMASK**, **ABORTMASK**, **INTMASK** and **BRAKEMASK**) for the action to occur on this fault.

C.2.89. SAFEZONECW

This axis parameter allows the user to specify the clockwise boundary of the safe zone associated with an axis in machine steps. The user may use a safe zone to designate a boundary in which the axis can travel, or one in which the axis can not travel. To enable or disable these zones. The user must properly set the **SAFEZONEMODE** axis parameter. Safe zones may be enabled/disabled during homing, by the **SOFTLIMITMODE** axis parameter. When setting this parameter it is necessary to know that its distance starts at the hardware home position.
Be sure to set the Safe Zone bit in the **FAULTMASK** axis parameter to enable the detection of this fault, then set the bit in the appropriate mask parameter (**DISABLEMASK**, **HALTMASK**, **AUXMASK**, **ABORTMASK**, **INTMASK** and **BRAKEMASK**) for the action to occur on this fault.

C.2.90. SAFEZONEMODE

This axis parameter determines how the system interprets the **SAFEZONECW** and **SAFEZONECCW** parameters. Setting this parameter to zero disables the safe zone, while a one defines an area in which the axis may not exit, and a two defines an area in which the axis may not enter. A safe zone fault occurs each time the associated axis moves into or out of an area that violates an active safe zone.

Be sure to set the Safe Zone bit in the **FAULTMASK** axis parameter to enable the detection of this fault, then set the bit in the appropriate mask parameter (**DISABLEMASK**, **HALTMASK**, **AUXMASK**, **ABORTMASK**, **INTMASK** and **BRAKEMASK**) for the action to occur on this fault.

C.2.91. SCALEPGAIN

This axis parameter will scale up (increase) the **PGAIN** servo loop axis parameter. There are 2 modes, one based upon position error and the other based upon commanded velocity.

Mode 1 is based upon position error. When the **POSERR** < 100 machine counts, the **PGAIN** axis parameter will be multiplied by the corresponding value in the table producing a stiffer steady state (at rest) servo loop. To enable the **PGAIN** scaling, set

SCALEPGAIN to 1, to disable it, set it to 0. The scaling is based upon the table below. The scaling is based upon the formula shown, with the PosGainTable [**POSERR**] term providing an index into the table shown:

PGAIN = PGAIN * PosGainTable[POSERR]

Examples:

If POSERR equals 1, the PGAIN multiplier would be 10.
If POSERR equals 17, the PGAIN multiplier would be 2.
If POSERR is18 or greater, the PGAIN multiplier is1.

Table C-5. Mode 1

ELEMENT #	0	1	2	3	4	5	6	7	8	9
Element 0	10	10	9	9	8	8	7	7	6	6
Element 10	5	5	4	4	3	3	2	2	1	1
Element 20	1	1	1	1	1	1	1	1	1	1
Element 30	1	1	1	1	1	1	1	1	1	1
Element 40	1	1	1	1	1	1	1	1	1	1
Element 50	1	1	1	1	1	1	1	1	1	1
Element 60	1	1	1	1	1	1	1	1	1	1
Element 70	1	1	1	1	1	1	1	1	1	1
Element 80	1	1	1	1	1	1	1	1	1	1
Element 90	1	1	1	1	1	1	1	1	1	1

Mode 2 is based upon the velocity command. When **IVEL** < 100 machine counts, the PGAIN axis parameter will be multiplied by the corresponding value in the table producing higher gain while crossing through zero velocity. To enable the **PGAIN** scaling, set SCALEPGAIN to 2, to disable it, set it to 0. The scaling is based upon the table below. The scaling is based upon the formula shown, with the PosGainTable [**IVEL**] term providing an index into the table shown:

PGAIN = PGAIN * PosGainTable [IVEL]

Examples:

If IVEL equals 1, the PGAIN multiplier would be 10.
If IVEL equals 17, the PGAIN multiplier would be 2.
If IVEL is 18 or greater, the PGAIN multiplier would be 1.

Table C-6. Mode 2

ELEMENT #	0	1	2	3	4	5	6	7	8	9
Element 0	1	1	10	10	9	8	7	7	6	6
Element 10	5	5	4	4	3	3	2	2	1	1
Element 20	1	1	1	1	1	1	1	1	1	1
Element 30	1	1	1	1	1	1	1	1	1	1
Element 40	1	1	1	1	1	1	1	1	1	1
Element 50	1	1	1	1	1	1	1	1	1	1
Element 60	1	1	1	1	1	1	1	1	1	1
Element 70	1	1	1	1	1	1	1	1	1	1
Element 80	1	1	1	1	1	1	1	1	1	1
Element 90	1	1	1	1	1	1	1	1	1	1

C.2.92. SERVOSTATUS

This is a bitwise axis parameter, providing the current status of the motion on this axis. Also, be aware that some motion conditions may also be reported in the **STATUS**, **Status1**, **Status2**, **Status3,** and **MOTIONSTATUS** parameters. There are pre-defined definitions of the bits in this axis parameter within AerStat.Pgm. These definitions take the form of "SERVOSTATUS_xxx" where xxx is the name as listed below. For example;

If (SERVOSTATUS.X BAND SERVOSTATUS_DriveEnabled)

or

If(SERVOSTATUS.X BAND 0h00000001)

could be used to test for the X axis Drive being enabled.

Table C-7. SERVOSTATUS bit definitions

Name	Hex Value	Description
DRIVE	0h00000001	Drive is enabled
AUX	0h00000002	Auxiliary (Mode) output on
CW_LIMIT	0h00000004	CW hardware limit active
CCW_LIMIT	0h00000008	CCW hardware limit active
HOME	0h00000010	Homing cycle active
DRIVE_FLT	0h00000020	Drive fault input active
ATHOME	0h00000040	Axis at home position
DONE	0h00000080	Motion Done
INPOS	0h00000100	Axis is in position
FAULTED	0h00000200	Axis fault present (See FAULT)
PROBE_INPUT	0h00000400	Probe input active
MARKER	0h00000800	Marker
HALL INPUT B	0h00001000	Hall-effect input B (#1)
HALL INPUT A	0h00002000	Hall-effect input A (#2)
HALL INPUT C	0h00004000	Hall-effect input C (#3)
HALL 4 (Unused)	0h00008000	Hall-effect input 4 (not used)
INEG_LIMIT	0h00010000	KI clamped negative
IPOS_LIMIT	0h00020000	KI clamped positive
VFF	0h00040000	VFF Enabled
BRAKE ACTIVE	0h00080000	Brake Output Active
ALIVE	0h00100000	Axis has no D/A
VVF_0ATC	0h00200000	VFF or position loop zero
FEEDBACK_IN	0h00400000	Feedback fault present
MFEEDBACK_IN	0h00800000	Master Feedback fault active
HP_VME_LASER	0h01000000	HP VME Laser
SCALEPGAIN	0h02000000	SCALEPGAIN active
AC	0h04000000	AC motor selected
MSET	0h08000000	Axis in MSET mode
HOMED	0h10000000	Axis has been homed since reset
ENCODER	0h20000000	Axis has encoder feedback
ERROR_MAP	0h40000000	Error mapping enabled
PLOOP_ONLY	0h80000000	Position loop only

C.2.93. SIMULATION

To facilitate easy debugging of parts programs, this axis parameter allows the user to place an axis into a simulation mode. While in this mode, the motor's torque remains steady (holding the axis in position), but no motion occurs. This axis parameter is used to implement Simulation mode (parameter value=1). While this parameter's value is non-zero, the motor's torque remains steady (holding the axis in position), but no axis motion occurs. While executing a CNC program on a simulated axis, the controller performs all calculations normally, but the torque command never reaches the motor. Instead, the torque serves as the feedback for this axis, effectively creating a system free from velocity error. While moving an axis in simulation mode, the velocity and position commands behave normally, but the position remains constant, and the velocity is zero. All other features, such as data acquisition, continue to function normally.
Setting this parameter to zero disables the simulation mode, while a one enables it. The default is to disable the simulation mode. You should not modify the simulation axis parameter while the axis is in motion.

Setting this parameter to two activates machine lock, which updates the machines internal positions for the new current position.

Alternatively, an axis may be configured as a virtual axis for debugging purposes. In this mode, **SIMULATION** has no effect.

C.2.94. SOFTLIMITMODE

This axis parameter sets the active mode for the software limits (defined by the **CWEOT** and **CCWEOT** parameters), as well as the safe zones (defined by the **SAFEZONECW**, **SAFEZONECCW**, and **SAFEZONEMODE** parameters). In many systems, the current absolute position of an axis is unknown until after the axis reaches its home position. Therefore, the user should not activate a software limit or a safe zone until the system successfully completes the homing process. However, the mechanics of some systems do not permit execution of a normal homing sequence. Therefore, the user must use an alternate method to determine the absolute position. In this case, it may be logical to permit software limits and safe zones to be active at all times.

Setting this parameter to a one (1) causes software limits and safe zones to become active only after successfully homing the axis. The default value is zero (0), which causes software limits and safe zones to be active before and after homing the axis. Software limits and safe zones are never active during a homing cycle.

C.2.95. STATUS

This is a bitwise axis parameter, providing the current status of this axis. For example;

If(STATUS.X BAND 0h00000004)

could be used to test for the X axis CW end of travel limit bit being TRUE.

Table C-8. STATUS_xxxx Constants

Name	Hex Value	Description
DRIVE	0h00000001	drive active
AUX	0h00000002	auxiliary (mode) output active
CW_LIMIT	0h00000004	CW hardware limit active
CCW_LIMIT	0h00000008	CCW hardware limit active
HOME	0h00000010	home switch active
DRIVE_FLT	0h00000020	drive fault input active
ATHOME	0h00000040	axis at home position
DONE	0h00000080	motion done
INPOS	0h00000100	axis is within the in-position limit
FAULTED	0h00000200	axis is faulted
PROBE_INPUT	0h00000400	probe input active
MARKER	0h00000800	Marker
HALL INPUT B	0h00001000	Hall-effect input B (#1)
HALL INPUT A	0h00002000	Hall-effect input A (#2)
HALL INPUT C	0h00004000	Hall-effect input C (#3)
HALL 4 (Unused)	0h00008000	Hall-effect input 4 (not used)
MOVE_DIR	0h00010000	move direction
MOVING	0h00020000	G0, Home or Async. Motion Active
ACCEL	0h00040000	axis in acceleration phase
DECEL	0h00080000	axis in deceleration phase
HOMING	0h00100000	axis homing
FEED_OVER	0h00200000	feedrate override
PROFILE	0h00400000	G1/G2/G3 command executing
SYNC	0h00800000	axis in sync mode
CAM_TABLE	0h01000000	cam table enabled
HOME_DIR	0h02000000	home direction
CONT_MOVE	0h04000000	continuous move
QUEUE	0h08000000	motion queue active
HOLD	0h10000000	hold active
AUX_MODE	0h20000000	aux mode
BLOCK_MOTION	0h40000000	block motion
HOLD_QUEUE	0h80000000	hold queue

C.2.96. SYNCSPEED

To understand this axis parameter, the user must be familiar with the operation of the synchronized motion through the CAM tables on the UNIDEX 600 Series motion controller. For a brief discussion of this feature, refer to the discussion of the **MASTERPOS** axis parameter. There are two modes that the user can perform CAM table execution. In the first mode, the system assumes that the current slave position is the starting point of the CAM table. Also, the system assumes that all slave position entries are relative to that starting point. The second mode of CAM table execution does not make that assumption. Instead, it interprets the slave positions found within the CAM table as absolute positions. With synchronization enabled, the system determines the current location within the CAM table based on the current master position. The slave axis then moves to the position that corresponds to the current master position. This parameter defines the speed at which the slave axis is to move.

C.2.97. SYSTEMCLOCK

This axis parameter is the same as the **CLOCK** axis parameter, but is read-only.

C.2.98. VELCMDTRAP

This axis parameter determines the maximum commanded velocity that the axis may move, in counts per second. A command trap occurs if the commanded velocity exceeds the amount specified in this parameter. The user may enter a zero to disable the commanded velocity trap detection.

Be sure to set the Velocity Command Trap bit in the **FAULTMASK** axis parameter to enable the detection of this fault, then set the bit in the appropriate mask parameter (**DISABLEMASK**, **HALTMASK**, **AUXMASK**, **ABORTMASK**, **INTMASK** and **BRAKEMASK**) for the action to occur on this fault.

This parameter applies to all types of motion.

When using the **MaxFeedRateIPM** (or **MaxFeedRateRPM** for rotary axis) machine parameters, this parameter cannot be used. Its value will be overwritten. Consider using the **MaxFeedRateIPM** or **MaxFeedRateRPM** parameters as an alternative method to limit axis speeds.

C.2.99. VELPOSITION

This axis parameter is the axis position as calculated by integrating the velocity from the secondary feedback device. The primary feedback device will be used for this calculation, if no secondary feedback device is present.

C.2.100. VELTIMECONST

This axis parameter is used by the velocity filter to smooth out the velocity command. The units are milliseconds and approximately specify the minimum time to reach the commanded velocity. In other words, this insures that no acceleration can take place in less than VELTIMECONST milliseconds. A zero value disables filtering. The filtering is achieved by modifying the velocity command. This parameter is manipulated by the **G23** / **G24** commands to implement corner-rounding on the UNIDEX 600 Series controller and may be set by the user, but, not while G23 is active. Setting this parameter to non-zero activates filtering. Increasing the value causes more filtering and more corner rounding. This parameter affects all types of motion (including cam motion), except homing.

If this filter is used in conjunction with the **GANTRYMODE**, it must be applied to the master and slave axes.

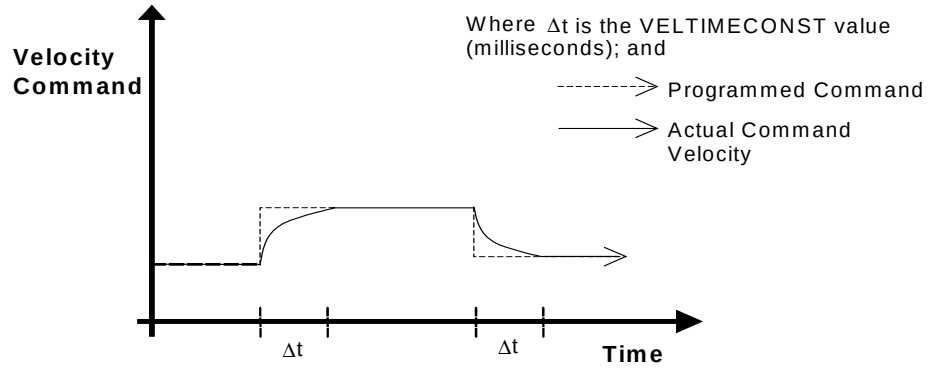


Figure C-7. Velocity Time Constant Effect on Velocity Change

The observed value of Δt, for contoured moves (**G1**, **G2**, **G3**, **G12**, **G13**) may actually be larger than the VELTIMECONST value provided, You must experiment, under benign conditions to insure an appropriate setting for this axis parameter.

The acceleration of contoured motion may be limited by the following task parameters:

BlendMaxAccelLinearIPS2 - Maximum acceleration for linear axes

BlendMaxAccelRotaryDPS2 - Maximum acceleration for rotary axes

BlendMaxAccelCircleIPS2 - Limiting acceleration during circle/arc's (**G2**/**G3**)

C.2.101. VFF

This axis parameter enables or disables the Velocity Feed Forward portion of the servo loop. Once enabled, this function minimizes the position following error. A one enables this function, while a zero disables it.

C.2.102. VGAIN

This axis parameter scales the velocity command within the servo loop. VGAIN is used only In tachometer based (velocity command) systems, VGAIN is multiplied by the velocity command and added to the velocity command output to the servo amplifier. This parameter is used to reduce position error during the constant velocity portion of the move, (much like **VFF** would be used in a torque loop) by feed forward.

Note, that you can use the AutoTune feature, within the AerTune.exe utility (on the Tools menu) to automatically determine servo loop gains for a torque mode axis.

C.3. Machine Parameters

Machine parameters are only used by the CNC interface. Unless CNC motion is used, these parameters can be ignored. These values are used to specify additional information required by the controller to calculate axis motion. Each axis has its own set of machine parameters.

Table C-9. Machine Parameters

Name	Parameter #	Access	Minimum	Maximum	Default
AvgVelUnits	20	RU	<NA>	<NA>	<NA>
AxisState	15	RU	0	2	0
CntsPerDeg	2	RW	<NA>	<NA>	25,400.0
CntsPerInch	1	RW	<NA>	<NA>	25,4000.0
ControllingTask	16	RU	-1	3	-1
FixtureOffset	21	RW	<NA>	<NA>	<NA>
FixtureOffset2	24	RW	<NA>	<NA>	<NA>
FixtureOffset3	25	RW	<NA>	<NA>	<NA>
FixtureOffset4	26	RW	<NA>	<NA>	<NA>
FixtureOffset5	27	RW	<NA>	<NA>	<NA>
FixtureOffset6	28	RW	<NA>	<NA>	<NA>
HomeDirection	8	RW	-1	1	1
HomeFeedRateIPM	9	RW	0.0	1,000,000	120.0
HomeFeedRateRPM	10	RW	0.0	1,000,000	60.0
HomeOffsetDeg	12	RW	<NA>	<NA>	0.0
HomeOffsetInch	11	RW	<NA>	<NA>	0.0
HomeType	7	RW	0	4	1
JogDistanceDeg	30	RW	0	1,000,000	5
JogDistanceInch	29	RW	0	1,000,000	1
JogVelocityRPM	32	RW	0	1,000,000	0
JogVelocityIPM	31	RW	0	1,000,000	0
MaxFeedRateIPM	3	RW	0.0	1,000,000	500.0
MaxFeedRateRPM	4	RW	0.0	1,000,000	300.0
NumDecimalsEnglish	13	RW	1.0	14.0	4.0
NumDecimalsMetric	14	RW	1.0	14.0	3.0
PositionCmdUnits	18	RU	<NA>	<NA>	<NA>
PositionUnits	17	RU	<NA>	<NA>	<NA>
PresetCmdUnits	19	RU	<NA>	<NA>	<NA>
RapidFeedRateIPM	5	RW	0.0	1,000,000	120.0
RapidFeedRateRPM	6	RW	0.0	1,000,000	60.0
ReverseSlewDir	34	RW	-1	1	0
ScaleFactor	22	RW	-1,000,000	1,000,000	1
Type	0	RW	0	3	0
UnusedAxis	33	RW	0	1	0

C.3.1. Modifying a Machine Parameter within a CNC Program

Any machine parameter may be modified within a CNC program (or MDI command line), by specifying the machine parameter name followed by a decimal point and the axis name. The case of these machine parameters is significant, as defined in the machine parameter table.

The axis name is that assigned when the axis is configured and bound to the task within the axis configuration wizard. If the default axis name is used, the task axis names would apply.

Example:

```
MaxFeedRateIPM.X = 30          ; Write to machine parameters
MaxFeedRateRPM.Y = $GLOB3


$GLOB0 = MaxFeedRateIPM.X      ; Read from machine parameters
$GLOB1 = MaxFeedRateIPM.Y
```

C.3.2. AvgVelUnits

This is a read-only machine parameter that indicates the average velocity of the axis, averaged over the time period specified by the **AVGVELTIME** axis parameter. It is a signed value, meaning that a negative value indicates a negative velocity. The units of this parameter are user-units /minute: inches/minute for linear axes in the **G71** mode, millimeters/minute for linear axes in the G71 mode and always degrees/minute for rotary axes.

C.3.3. AxisState

This is a read only machine parameter indicating the current state of the axis. It may be either free (0), captured (1), or bound (2). See the BIND extended command for details.

C.3.4. CntsPerDeg

This machine parameter is used by the CNC interface to convert degrees into machine steps, it is specified in counts per degree. The sign of this parameter determines motor direction, a positive value indicates that a positive command will cause motion in the CW motor direction. A negative value indicates a positive command will cause motion in the CCW motor direction. A negative value indicates a positive command will cause motion in the CCW motor direction, which will cause the end of travel limits to appear to be inverted. A negative scale factor will cause a more positive display position, to encounter the CCW limit and a more negative position will encounter the CW limit position. This conversion factor is only used for rotary axes (axes whose Type machine parameter is greater than 0).

For brushless linear motors, the value entered here is the number of counts per electrical cycle of the motor.

Normally, this parameter is automatically set by the Parameter Configuration Wizard during the motor configuration process. This parameter should not be changed during program execution, doing so may cause unusual motion. Use the ScaleFactor task machine parameter to rescale the system coordinates during program execution.

The error "Parameter too high" can occur when setting this parameter from a CNC program. The message may not refer to this parameter, but can also refer to the MaxFeedRateRPM machine parameter, which is "reset" when this parameter is set.

C.3.5. CntsPerInch

This machine parameter is used by the CNC interface to convert user units into machine steps, it is specified in counts per inch, even if programming in millimeters (**G71**). The sign of this parameter determines motor direction, a positive value indicates that a positive command will cause motion in the CW motor direction. A negative value indicates a positive command will cause motion in the CCW motor direction, which will cause the end of travel limits to appear to be inverted. A negative scale factor will cause a more positive display position, to encounter the CCW limit and a more negative position will encounter the CW limit position. This conversion factor is only used for linear axes (axes whose Type machine parameter is equal to 0).

For brushless linear motors, the value entered here is the number of counts per electrical cycle of the motor.

Normally this parameter is automatically set by the Parameter Configuration Wizard during the motor configuration process. This parameter should not be set "on-the-fly" during program execution is dangerous and not recommended as it will rescale the system and may cause unusual motion. Use the ScaleFactor parameter if you want to deliberately rescale the system coordinates during program execution.

The error "Parameter too high" can occur when setting this parameter from a CNC program. The message may not refer to this parameter, but can also refer to the MaxFeedRateIPM machine parameter, which is "reset" when this parameter is set.

C.3.6. ControllingTask

This read only machine parameter indicates which task, if any, controls the axis. The values 0 through 3 indicate the four tasks respectively and a value of -1 indicates that the axis is not controlled by any task. An axis is controlled by a task if it is bound or captured by it.

For example, when the controller is reset, the ControllingTask value for axis 1 is –1. After task 1 binds axis 1 to X, then the ControllingTask value for axis 1 is 0. If task 3 later captures axis 1, then the ControllingTask value is 2. After task 3 releases axis 1, the ControllingTask value reverts back to 0.

C.3.7. FixtureOffset

The FixtureOffset machine parameter indicates the current fixture offset #1, as defined by the **G54** CNC G code.

C.3.8. FixtureOffset2

The FixtureOffset2 machine parameter indicates the current fixture offset #2, as defined by the **G55** CNC G code.

C.3.9. FixtureOffset3

The FixtureOffset3 machine parameter indicates the current fixture offset #3, as defined by the **G56** CNC G code.

C.3.10. FixtureOffset4

The FixtureOffset4 machine parameter indicates the current fixture offset #4, as defined by the **G57** CNC G code.

C.3.11. FixtureOffset5

The FixtureOffset5 machine parameter indicates the current fixture offset #5, as defined by the **G58** CNC G code.

C.3.12. FixtureOffset6

The FixtureOffset6 machine parameter indicates the current fixture offset #6, as defined by the **G59** CNC G code.

C.3.13. HomeDirection

This machine parameter specifies the feedrate in inches per minute, to be used by the CNC home and homeasync commands. This feedrate is only used for linear axes.

The direction defined by this parameter is independent of the sign of the CntsPerInch (CntsPerDeg for rotary axes) scaling machine parameter.

C.3.14. HomeFeedRateIPM

This machine parameter specifies the feedrate in inches per minute, to be used by the CNC home and homeasync commands. This feedrate is only used for linear axes.

C.3.15. HomeFeedRateRPM

This machine parameter specifies the feedrate in RPM to be used by the CNC home and homeasync commands. This feedrate is only used for rotary axes.

C.3.16. HomeOffsetDeg

This machine parameter specifies the axis position after a home cycle is complete. It does not produce motion from the home position at the completion of the home cycle. The **HOMEOFFSET** axis parameter is overwritten by this value converted by the appropriate user units to counts conversion factor. This offset is only used for rotary axes.

If this parameter is set to a non-zero value with axis calibration active, the axis may not stop on the marker at the completion of the home cycle. If it desired to do so, for a positioning test, you must temporarily disable axis calibration, within the axis configuration wizard.

C.3.17. HomeOffsetInch

This machine parameter specifies the axis position after a home cycle completes. It does not produce motion from the home position at the completion of the home cycle. The **HOMEOFFSET** axis parameter is overwritten by this value converted by the appropriate user units to counts conversion factor. This parameter is only used for linear axes.

If this parameter is set to a non-zero value with axis calibration active, the axis may not stop on the marker at the completion of the home cycle. If it desired to do so, for a positioning test, you must temporarily disable axis calibration, within the axis configuration wizard.

C.3.18. HomeType

The HomeType machine parameter defines the homing cycle as one of the five possible homing cycles. The home position is the absolute zero reference position found by the home cycle. At the completion of all home cycles, the position register will be loaded with the value of the HomeOffsetInch (or HomeOffsetDeg) machine parameter. No motion will occur for a non-zero home offset value.

The software limits are not monitored during homing, even if they are set in the **FAULTMASK** axis parameter. However, immediately after homing, they will begin to be monitored.

All types of the home cycle begin movement in the direction specified by the HomeDirection machine parameter. The HomeFeedRateIPM (or HomeFeedRateRPM) machine parameters allow the feedrate to be defined for the home cycle. Typically, the home feedrate is a low velocity that produces an accurate, repeatable home reference point. A low speed is not detrimental to machine throughput, since it is only done occasionally or when the machine is first powered up. While homing, the axis follows the accel/decel axis parameters (see **ACCELMODE** axis parameter). However, the accel and

decel axis parameters will not be used in the direction reversal from the home limit, if that occurs, it is instantaneous.

The home cycle feedrate may be increased if the **HOMEVELMULT** axis parameter is used.

For all home types, other than type 4, the axis will perform the homing motion even if it is currently on the home position.

If the axis encounters an EOT limit while homing and the **FAULTMASK** axis parameter is set so the EOT limits do not generate faults (this is the default), then it will not generate faults, but reverse direction and continue the homing cycle. However, if other faultmask bits are set (such as **ABORTMASK**) that stop motion on an EOT, then the motion stops instead of reversing.

The five types of homing cycles are:

HomeType 0 – To Home Limit AND Reference Pulse
HomeType 1 – To Home Limit & then Reverse to Reference pulse, (Aerotech Std.)
HomeType 2 – To Marker
HomeType 3 – To Home Limit Switch
HomeType 4 – Home Position at current Position

C.3.18.1. TYPE 0 - Home to Limit AND Reference Pulse

The home position is found at the home limit switch when the reference pulse is true. There must be an absolute position where both occur simultaneously. The HomeDirection is the direction to begin traveling towards the home limit. If the axis encounters the home switch while traveling in the opposite direction specified by the HomeDirection (this happens if the axis hits an EOT before the home switch and reverses direction), then it will pass up the home switch and reverse direction in order to strike the home switch from the proper direction.

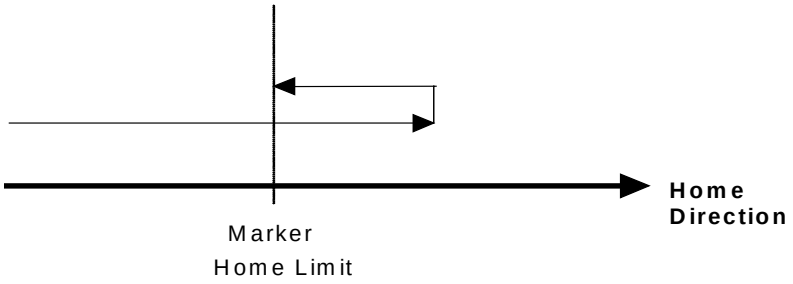


Figure C-8. Home to Limit Illustration

C.3.18.2. TYPE 1 - Home into Limit & Reverse to Reference Pulse, (Aerotech Std.)

The axis will begin moving in the direction specified by the HomeDirection task parameter. Once the home limit switch is found, it reverses direction. The home position is found at the first reference pulse that occurs after the home limit switch is false following the direction reversal. However, the axis will pass over the reference pulse (after the direction reversal), reverse direction again (now heading in the original

HomeDirection specified direction), before heading back towards, and stopping at, the reference pulse.

The HomeDirection axis parameter specifies the initial home direction. Typically, the home limit switch is the same as one of the EOT limits and the HomeDirection parameter is set accordingly. However, this is not necessary as long as the **FAULTMASK** axis parameter is set so that direction reversals occur when hitting the EOT limit. If the home switch is not on the proper EOT, then the axis may hit an EOT before it hits the home switch.

The speed of the home cycle is determined by the HomeFeedRateIPM (or HomeFeedRateRPM for rotary axes) task parameters. However, after the Home limit is encountered, the **HOMEVELMULT** axis parameter can be used to slow down the marker pulse (reference pulse) search speed.

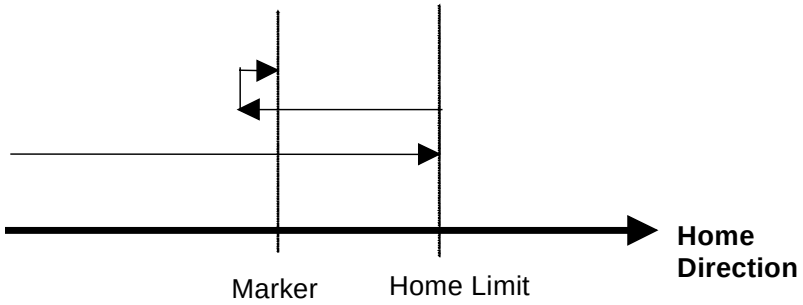


Figure C-9. Home into Limit Illustration

C.3.18.3. TYPE 2 - Home to Marker

The home position is the reference pulse. It rotates in the specified home direction and stops on the first marker or resolver null.

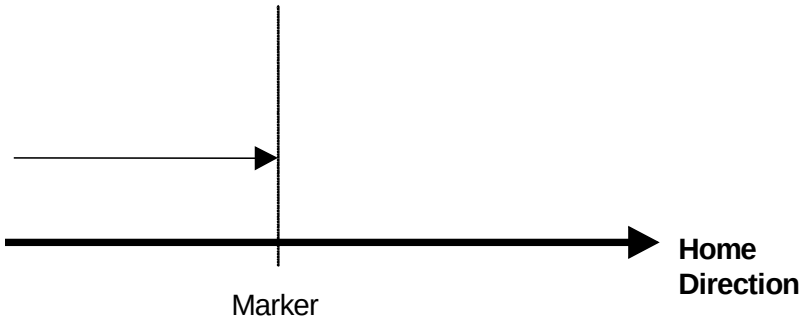


Figure C-10. Home to Marker Illustration

C.3.18.4. TYPE 3 - Quick Home to Limit Switch

The home position is the home limit switch. If the axis encounters the home limit switch while traveling in the opposite direction specified by the HomeDirection (this happens if the axis hits an EOT before the home switch and reverses direction), then it passes up the home switch and reverses direction in order to strike the home switch from the proper direction.

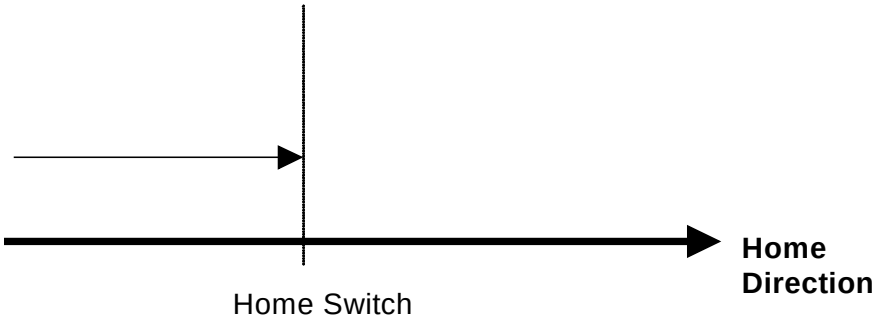


Figure C-11. Quick Home to Limit Switch Illustration

C.3.18.5. Type 4 - Home Position at Current Position

The home position will be set to the current position. No homing motion is generated, axis calibration and backlash compensation will be activated, if defined for the axis.

This home type may not be used with axis calibration.

C.3.19. JogDistanceDeg

This machine parameter specifies the distance for this axis on the Jog Page (if it is a rotary Type axis), when the jog distance or jog distance (hold) modes are selected.

C.3.20. JogDistanceInch

This machine parameter specifies the distance for this axis on the Jog Page(if it is a linear Type axis), when the jog distance or jog distance (hold) modes are selected.

C.3.21. JogVelocityIPM

This machine parameter specifies the speed for this axis on the Jog Page(if it is a linear Type axis), when the High speed range is selected. If this parameter is 0, the value of the RapidFeedRateIPM machine parameter will be used.

C.3.22. JogVelocityRPM

This machine parameter specifies the speed for this axis on the Jog Page(if it is a rotary Type axis), when the High speed range is selected. If this parameter is 0, the value of the RapidFeedRateRPM machine parameter will be used.

C.3.23. MaxFeedRateIPM

This machine parameter specifies the maximum speed allowed for this axis if it is defined as a linear Type axis. If the absolute value of the directed speed of the axis exceeds this value, during contoured, **G0**, asynchronous or spindle motion, feedrate limiting will occur, such that, the controller will decrease the speed so that the axis will move at its maximum speed defined by this parameter. Contoured moves involving multiple axes will have their vector speed decreased until no axes in the move violates its maximum feedrate. If a contoured move has its speed decreased for this reason, the UNIDEX 600 MMI, will change the color of the actual feedrate displayed to yellow.

This value must be set in inches per minute, even if the programmer intends to program in millimeters. If this parameter is set to zero, there is no maximum feedrate for this axis. This parameter does not limit the speed of any camming motion. If you change this parameter while a move is in progress, it will not take effect until the move is complete.

The MaxFeedRateIPM parameter will scale down the feedrate, when an axis speed limit is exceeded. However in some cases, the user may desire to generate a fault when the speed is violated, rather than scale down the feedrate. To do this, set this parameter to zero, and set the **VELCMDTRAP** axis parameter.

A drawback of the MaxFeedRateIPM machine parameter, is the limiting will take place even if the axis never reaches the commanded speed (never accelerates to full speed). So for short moves or paths, this parameter can limit the feedrate excessively. In these cases, when a speed limit is still required, use the **VELCMDTRAP** axis parameter instead.

This parameter, and the MaxFeedRateIPM parameter can be limited to a lower value than expected, due to the fact that the velocity value stored internally, in machine counts, is only 16 bits. Therefore an upper limit to this parameter is:

(65,536,000 * 60) / (CntsPerInch)

C.3.24. MaxFeedRateRPM

This machine parameter specifies the maximum speed allowed for this axis if it is defined as a rotary Type axis. This parameter is in user units of revolutions per minute. See MaxFeedRateIPM for linear axes and more details, as it relates to this parameter.

This parameter, and the MaxFeedRateRPM parameter can be limited to a lower value than expected, due to the fact that the velocity value stored internally, in machine counts, is only 16 bits. Therefore an upper limit to this parameter is:

(65,536,000) / (CntsPerDeg * 6)

C.3.25. NumDecimalsEnglish

This parameter allows the user to define the position display when the **G70** English mode is active. This is the total number of digits displayed after the decimal point.

C.3.26. NumDecimalsMetric

This parameter allows the user to define the position display when the **G71** Metric mode is active. This is the total number of digits displayed after the decimal point.

C.3.27. PositionCmdUnits

This machine parameter indicates the current commanded position in user units. The value is continuously updated by the servo loop. This position is unaffected by an offset value (**G92**).

The units of this parameter are the user-units of the controlling task: degrees for rotary axes, inches for linear axes when the task is in **G70** mode, millimeters for linear axes when the task is in **G71** mode.

C.3.28. PositionUnits

This read–only machine parameter indicates the current actual position in user units. The value is continuously updated by the servo loop. This position is unaffected by an offset value (**G92**).

The units of this parameter are the user-units of the controlling task: degrees for rotary axes, inches for linear axes when the task is in **G70** mode, millimeters for linear axes when the task is in **G71** mode. The **POS** axis parameter indicates actual position in machine counts and may also be used to set the position directly.

Note, this parameter may be selected for display on the position tracking display of the UNIDEX 600 MMI by selecting "machine", from the column display selection key of the position display configuration page.

C.3.29. PresetCmdUnits

This machine parameter indicates the preset position register in user units. This value is updated continuously by the servo loop. This position is shown relative to any offset value (**G92**).

C.3.30. RapidFeedRateIPM

This machine parameter specifies the feedrate to be used by the controller for rapid (or point-to-point **G0**) motion, the maximum feedrate that may be commanded by the optional joystick, as well as the maximum velocity for jogging the axes within the jog page of the UNIDEX 600 MMI. This feedrate is only used for linear axes, as defined by the Type machine parameter.

C.3.31. RapidFeedRateRPM

This machine parameter specifies the feedrate to be used by the controller for rapid (or point-to-point **G0**) motion, the maximum feedrate that may be commanded by the optional joystick, as well as the maximum velocity for jogging the axes within the jog page of the UNIDEX 600 MMI. This feedrate is only used for rotary axes, as defined by the Type machine parameter.

C.3.32. ReverseSlewDir

This machine parameter will reverse the direction of travel commanded by the joystick. To invert the direction, set this parameter to –1.

C.3.33. ScaleFactor

This machine parameter is used by the **G151** code to scale axes. You may set the scale factors independently for each axis via this parameter.

If you set this parameter to zero, it sets the scale factor to one, and resets the scale center to zero.

C.3.33.1. Scaling Limitations

You cannot generate an ellipse from an arc by setting the ScaleFactor machine parameter to different values on different axes. The controller will always try to draw a circular arc, and therefore programs that draw legal arcs may generate circular radius errors when different scale factors are used on different axes.

Negative scale factor values may be used for the ScaleFactor machine parameter (or **G151**), which, will in effect cause mirroring, however, **G2**/**G3** command polarity will not be reversed, causing the wrong rotational direction to be generated. A negative scale factor should not be used in this case. A positive scale factor should be used with the **G83**-Mirroring command.

C.3.34. Type

This machine parameter specifies whether the axis is a linear or rotary axis. There are three types of axis types.

0 – Linear axis.
1 - Rotary axis with modulo position rollover at 360 degrees.
2 - Rotary axis without modulo position rollover.

Distances and velocities that are not expressed in machine counts, will be expressed as this parameter dictates. Distances for linear Type axis are in inches/mm (**G70** / **G71**), and distances for rotary Type axis are in degrees. Velocities for linear Type axes are in inches or mm per minute (see **G70** / **G71**), velocities for rotary Type axis are in RPM.

Many parameters have parallel parameters dependent on the axis Type. For example, a rotary Type axis uses the HomeFeedRateRPM parameter, while a linear type axis uses the HomeFeedRateIPM parameter.

Furthermore, rotary type axes may have their reported position modulo to 360 degrees. Rotary type axes may also modulo absolute target positions to 360 degrees. Absolute target positions include MOVETO targets, and **G0**/**G1**/**G2**/**G3** targets when you are in **G90** mode. For example, **G90** A500, where A is a rotary axis, moves to 140 degrees. Use non-modulo rotary type 2 to prevent modulo position rollover.

The error "Parameter too high" can occur when setting this parameter from a CNC program. The message may not refer to this parameter, but can also refer to the MaxFeedrateIPM and MaxFeedRateRPM machine parameters, which are "reset" when this parameter is set.

C.3.35. Unused Axis

This machine parameter is used in very limited circumstances to provide a method for people to mix C programming calls with CNC programs. By setting this parameter to 1, the CNC marks the axis "Not In Use". Internally, the CNC keeps track of axes that it is using and makes the assumption that axes that are moving have been started by CNC. However, calls made by C Library functions (AerMove functions) do not rely on the CNC in any way and in some circumstances, conflicts can occur. The following situation demonstrates the problem

1) From a CNC program (or MDI Line) execute the following command:

```
G1 X1 Y1          ; The CNC removes X and Y from NotInUseMask
                  ; and places in ProfileMask
```

2) From AerDebug on Axis 1(Or a C Program that calls AerMoveFreerun) start the axis moving

```
MFREE 1 100000
```

3) From a CNC program (or MDI Line) execute the following command:

```
G1 Y2             ; >>This line causes a task fault
```

This line will cause a TaskFault. The CNC thinks it still owns the X axis and it is moving because of something that it did.

To correct the problem:

1) Modify the procedure in step 1 above to the following

```
a) G1 X1 Y1       ; The CNC removes X and Y from NotInUseMask
                  ; and places in ProfileMask
b) UnusedAxis.X = 1   ; Tell the CNC we're no longer using the X Axis
```

Now perform steps 2) and 3)

C.4. Task Parameters

Task parameters are only used by the CNC interface. Unless CNC motion is used these parameters may be ignored. These values are used to specify task specific information. Each task has its own independent set of task parameters.

Table C-10. Task Parameters

Name	Parameter #	Access	Minimum	Maximum	Default
AccelRate	141	RWU	0.0	100,000.0	30.0
AccelRateDPS2	33	RW	0.0	360,000.0	60.0
AccelRateIPS2	31	RW	0.0	100,000.0	30.0
AccelTimeSec	29	RW	0.0	100.0	0.1
AnalogMFOInput	56	RW	-1	7	-1
AnalogMSOInput	63	RW	-1	7	-1
BlendMaxAccelLinearIPS2	97	RW	0	1.0e+037	0
BlendMaxAccelRotaryDPS2	98	RW	0	1.0e+037	0
BlendMaxAccelCircleIPS2	99	RW	0	1.0e+037	0
CannedFunctionID	124	RW	0	100	0
ChordicalSlowdownMsec	130	RW	-20	20	0
ChordicalToleranceInch	127	RW	0	1.0e+037	0
CommandVelocityVariance	131	RW	0	1.0e+037	.1
Coord1I	39	RW	0	15	0
Coord1J	40	RW	0	15	1
Coord1K	41	RW	0	15	2
Coord1Plane	42	RW	1	3	1
Coord2I	43	RW	0	15	3
Coord2J	44	RW	0	15	4
Coord2K	45	RW	0	15	5
Coord2Plane	46	RW	1	3	1
CutterActive	139	RWU	0	1.0e+037	0
CutterLength	134	RWU	0	1.0e+037	0
CutterOffsetX	136	RWU	0	1.0e+037	0
CutterOffsetY	137	RWU	0	1.0e+037	0
CutterRadius	138	RWU	0	100.0	.5
CutterRadiusInch	49	RW	0	100.0	.5
CutterToleranceDeg	132	RW	0	1.0e+037	.5
CutterWear	135	RWU	0	1.0e+037	0
CutterX	47	RW	0	15	0
CutterY	48	RW	0	15	1
CutterZ	133	RW	0	15	1
DecelOnProgramAbortMask	125	RW	0	65,535	-1
DecelRate	142	RWU	0	100,000	30.0
DecelRateDPS2	34	RW	0.0	360,000.0	60.0
DecelRateIPS2	32	RW	0.0	100,000.0	30.0
DecelTimeSec	30	RW	0.0	100.0	0.1
DryRunLinearFeedRateIPM	119	RW	0.0	1.0e+037	0
DryRunRotaryFeedRateRPM	120	RW	0.0	1.0e+037	0
ErrCode	64	RW	-2,147,483,648	2,147,483,648	0

Table C-10. Task Parameters (Continued)

Name	Parameter #	Access	Minimum	Maximum	Default
EstopInput	8	RW	-1	511	-1
ExecuteNumLines	102	RW	-2,147,483,648	2,147,483,648	1
ExecuteNumMonitors	147	RW	-2,147,483,648	2,147,483,648	-1
ExecuteNumSpindles	148	RW	-2,147,483,648	2,147,483,648	1
FeedHold	57	RW	0	1	0
FeedHoldEdgeInput	10	RW	-1	511	-1
FeedHoldInput	9	RW	-1	511	-1
GlobalEStopDisable	65	RW	0	1	0
HaltTaskOnAxisFault	68	RW	0	1	1
IgnoreAxesMask	126	RW	0	65,535	0
InterruptMotion	69	RW	0	1	0
InterruptMotionReturnType	70	RW	0	4	0
JogPair1Axis1MinusIn	107	RW	-1	511	-1
JogPair1Axis1PlusIn	106	RW	-1	511	-1
JogPair1Axis2MinusIn	110	RW	-1	511	-1
JogPair1Axis2PlusIn	109	RW	-1	511	-1
JogPair1EnableIn	103	RW	-1	511	0
JogPair1Axis1	105	RW	0	65,535	0
JogPair1Axis2	108	RW	0	65,535	0
JogPair1Mode	104	RW	0	3	0
JogPair2Axis1MinusIn	115	RW	-1	511	-1
JogPair2Axis1PlusIn	114	RW	-1	511	-1
JogPair2Axis2MinusIn	118	RW	-1	511	-1
JogPair2Axis2PlusIn	117	RW	-1	511	-1
JogPair2EnableIn	111	RW	-1	511	0
JogPair2Axis1	116	RW	0	65,535	0
JogPair2Axis2	113	RW	0	65,535	0
JogPair2Mode	112	RW	0	3	0
JoyStickPort	80	RW	0	3	0
LinearFeedRate	35	RW	0.0	1.0e+037	0
LinearFeedRateActual	66	RU	0.0	1.0e+037	0
LineNumberUser	122	RU	0	2,147,483,648	0
LineNumber960	123	RU	0	2,147,483,648	0
MaxCallStack	3	RW	0	100	10
MaxLookAheadMoves	121	RW	1	1,000,000	10
MaxModeStack	4	RW	0	100	10
MaxMonitorData	54	RW	0	1,000	10
MaxOnGosubData	55	RW	0	1,000	10
MaxRadiusAdjust	143	RW	0	360	.05
MaxRadiusError	58	RW	0	100	5
MFO	37	RWU	0.0	2.0	1.0
Mode1	62	RW	0	-1	39
MSO	38	RWU	0.0	2.0	1.0
NormalcyToleranceDeg	129	RW	0	20	0

Table C-10. Task Parameters (Continued)

Name	Parameter #	Access	Minimum	Maximum	Default
NormalcyAxis	52	RW	0	15	7
NormalcyX	50	RW	0	15	0
NormalcyY	51	RW	0	15	1
Number	0	R	0.0	3.0	1.0
NumTaskAxisPts	7	RW	0	1,000	10
NumTaskDoubles	5	RW	0	1,000	10
NumTaskStrings	6	RW	0	1,000	10
RIAction1	78	RW	-1	127	-1
RIActionAxis	90	RW	0	15	0
RIActionParm1	91	RW	-2,147,483,648	2,147,483,648	0
RIActionParm2	92	RW	-2,147,483,648	2,147,483,648	0
RIActionOpCode	89	RW	0	17	0
ROAction1	79	RW	-1	127	-1
ROReq1	77	RW	-1	127	-1
ROReq1Mask	128	RW	0	65,535	-1
RotaryFeedRate	36	RW	0.0	1.0e+037	0
RotaryFeedRateActual	67	RU	0.0	1.0e+037	0
RotateAngleDeg	21	RWU	0.0	360.0	0.0
RotateX	19	RW	0	15	0
RotateY	20	RW	0	15	1
RthetaEnabled	27	RWU	0	2	0
RthetaR	24	RW	0	15	2
RThetaRadius	140	RWU	0.0	1,000.0	0.0
RthetaRadiusInch	26	RW	0.0	1,000.0	0.0
RthetaT	25	RW	0	15	3
RthetaX	22	RW	0	15	0
RthetaY	23	RW	0	15	1
S1_Index	11	RW	0	15	0
S1_RPM	12	RW	0.0	1.0e+006	300.0
S1_SpindleRadius	93	RWU	-99,999,999	99,999,999	0
S2_AnalogMSOInput	71	RW	-1	7	-1
S2_Index	13	RW	0	15	1
S2_MSO	74	RWU	0	2.0	0
S2_RPM	14	RW	0.0	1.0e+006	300.0
S2_SpindleRadius	94	RWU	-99,999,999	99,999,999	0
S3_AnalogMSOInput	72	RW	-1	7	-1
S3_Index	15	RW	0	15	2
S3_MSO	75	RWU	0	2.0	0
S3_RPM	16	RW	0.0	1.0e+006	300.0
S3_SpindleRadius	95	RWU	-99,999,999	99,999,999	0
S4_AnalogMSOInput	73	RW	-1	7	-1
S4_Index	17	RW	0	15	3
S4_MSO	76	RWU	0	2.0	0
S4_RPM	18	RW	0.0	1.0e+006	300.0
S4_SpindleRadius	96	RWU	-99,999,999	99,999,999	0
SlewPair1	81	RW	1	49,152 (0xc000)	3

Table C-10. Task Parameters (Continued)

Name	Parameter #	Access	Minimum	Maximum	Default
SlewPair2	82	RW	0	49,152 (0xc000)	12
SlewPair3	83	RW	0	49,152 (0xc000)	0
SlewPair4	84	RW	0	49,152 (0xc000)	0
SlewPair5	85	RW	0	49,152 (0xc000)	0
SlewPair6	86	RW	0	49,152 (0xc000)	0
SlewPair7	87	RW	0	49,152 (0xc000)	0
SlewPair8	88	RW	0	49,152 (0xc000)	0
Status1	59	R	0	-1	0
Status2	60	R	0	-1	16
Status3	61	R	0	-1	75,498,560
TaskFault	1	RWU	-2,147,483,648	2,147,483,648	0
TaskWarning	2	RWU	0	4.1B	0
UpdateNumEntries	144	RW	1	30	30
UpdateTimeSec	28	RW	0.0	0.05	0.01
UserFeedRateMode	53	RW	0	5	0

C.4.1. Modifying the Task Parameters of another Task

Another tasks, task parameters may be referenced within another task, by appending a decimal point and then the desired task number to the end of the task parameter. The case of these task parameters is significant, as defined in the task parameter table. A task parameter on the current task may be modified directly, without the decimal point and task number as shown in the examples below, i.e.; RIAction1 = RIO_CYCLESTART.

For example, a single CNC program could start a CNC program running on tasks 1, 2 and 3 from within task 4, as shown below;

```
RIAction1.1 = RIO_CYCLESTART    ; Source the cycle start action on task 1

RIAction1.2 = RIO_CYCLESTART    ; Source the cycle start action on task 2

RIAction1.3 = RIO_CYCLESTART    ; Source the cycle start action on task 3
```

The status of those programs could then be read via that tasks Status1 word.

```
$GLOB1 = Status1.1      ; read status word 1 from task 1

$GLOB2 = Status1.2      ; read status word 1 from task 2

$GLOB3 = Status1.3      ; read status word 1 from task 3
```

A full example is illustrated in TN0003, within the online help file.

C.4.2. AccelRate

This task parameter will indicate or may be used to specify the current Acceleration rate of the task in the current user units. This parameter is inconsequential, if rotary axes are dominant. In this case, refer to the AccelRateDPS2 task parameter.

C.4.3. AccelRateDPS2

This task parameter specifies the acceleration rate in degrees per second squared used by the CNC for contoured motion acceleration (**G1**, **G2**, **G3**, **G12**, **and G13**), when the **G68** mode is active. Otherwise, the acceleration will be time based and AccelTimeSec parameter will be used.

The linear AccelRateIPS2/DecelRateIPS2 are used for linear dominant moves while the rotary AccelRateDPS2 /DecelRateDPS2 rates are used for rotary dominant moves. Refer to the **G98**/**G99** commands for information on rotary/linear axes dominance.

When accelerating/decelerating in rate based mode, the controller will not perform the acceleration/deceleration in a time period that is shorter than the UpdateTimeSec task parameter value. Therefore, for high rates of acceleration/deceleration and/or acceleration/decelerations between very similar speeds, the rate can be less than that specified.

C.4.4. AccelRateIPS2

This task parameter specifies the acceleration rate in inches per second squared used by the CNC for contoured motion acceleration (**G**1, **G2**, **G3**, **G12**, **G13**), in linear dominant moves, when the **G68** command is active. Otherwise, **G67** will be active causing the acceleration to be time based and the AccelTimeSec task parameter will be used. The linear AccelRateIPS2 /DecelRateIPS2 are used for linear dominant moves while the rotary AccelRateDPS2 /DecelRateDPS2 rates are used for rotary dominant moves. Refer to the G98/G99 command for information on rotary/linear axes dominance.

The units will always be in Inches per second squared, regardless of the current **G70**/**G71** (metric/english) mode.

When accelerating/decelerating in rate based mode, the controller will not perform the acceleration/deceleration in a time period that is shorter than the UpdateTimeSec task parameter value. Therefore, for high rates of acceleration/deceleration and/or acceleration/decelerations between very similar speeds, the rate can be less than that specified.

C.4.5. AccelTimeSec

This task parameter specifies the acceleration time in seconds used by the controller for contoured motion acceleration (**G1**, **G2**, **G3**, **G12**, **and G13**). The **G67** mode must be set. Otherwise, **G68** will be active and the acceleration will be rate based and one of the acceleration rate task parameters will be used.

C.4.6. AnalogMFOInput

This task parameter specifies which analog input channel is used to update the MFO task parameter. A value of -1 disables the analog MFO input. However, when this parameter is set to a valid analog input, that will be used to automatically update the MFO parameter. When under control of the AnalogMFOInput, the MFO parameter cannot exceed a value of 2. The analog input is averaged over 100 milliseconds and scaled from -10 volts to 10 volts to match a 0 to 2 ratio MFO range. Also, the MFO value will be rounded to the nearest .05 (or 5%), when under control of the AnalogMFOInput. In other words, the MFO will increment from 0, .1, .15 etc. up to 2.0.The MFO value will not be changed if the MFOLock task mode is active. The MFO slider bar on the Run and Manual (MDI) screens of the MMI 600 can only be used when this parameter is set to –1, other values will enable external control, causing the slider bar to display the set value. This task parameter is sampled at the rate indicated by the AvgPollTimeSec global parameter.

C.4.7. AnalogMSOInput

This task parameter specifies which analog input channel is used for the analog MSO for the first spindle, as defined by S1_Index. A value of -1 disables the analog MSO input. However, when this parameter is set to a valid analog input, that will be used to automatically update the MSO parameter. When under control of the AnalogMSOInput, the MSO parameter cannot exceed a value of 2. The analog input is averaged over 100 milliseconds and scaled from -10 volts to 10 volts to match a 0 to 2 ratio for the MSO range. Also, the MSO value will be rounded to the nearest .05 (or 5%), when under control of the AnalogMSOInput.

The MSO value will not be changed if the MSOLock task mode is active. The MSO ratio used is not the instantaneous analog reading, but an average over the last 100 milliseconds. The MSO slider bar on the Run and Manual (MDI) screens of the MMI can only be used when this parameter is set to –1, other values will enable external control, causing the slider bar to display the set value. This task parameter is sampled at the rate indicated by the AvgPollTimeSec global parameter.

C.4.8. BlendMaxAccelLinearIPS2

Setting this task parameter non-zero allows the controller to automatically detect high accelerations caused by large changes in velocities between non-tangential (corners), contoured moves (**G108** mode), on linear Type axes and scale down each of the axes component velocities to minimize acceleration. See the BlendMaxAccelRotaryDPS2 task parameter for rotary axes. The units of this parameter are acceleration, in inches per second squared. For best results blending motion with this parameter, it is recommended that rate based (G68) linear acceleration and deceleration (**G64**) be used.

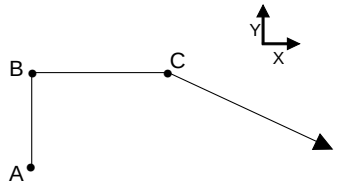
This parameter may be used via the methods described below:

C.4.8.1. Force Deceleration to Zero (G9), if Maximum Acceleration is Exceeded

If the BlendMaxAccelLinearIPS2 task parameter is set to a positive value, it causes the controller to execute a **G9** command (deceleration at the end of the CNC block) for each CNC block between contoured moves that involve a change in (a component) velocity, generating an acceleration greater than this parameter's value. This parameter is in inches per second squared. If any linear axis is subject to acceleration exceeding this value between two blended moves, the controller abandons blending, and behaves as if a **G9** command was present on the CNC line.

EXAMPLE:

```
G70 G91 G1      ; inches, relative pgm. mode, linear moves
F600            ; 600 inches/minute
G108            ; tell controller to not stop between moves.
Y10             ; Move from point A to point B
X10             ; Move from point B to point C
```



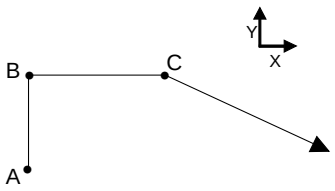
From point A to point B, the X axis moves at 600 inches/min, or 10 inches/sec, and the Y axis moves at 0 inches/sec. But, from point B to point C, X travels at 0 inches/sec, and Y at 10 inches/sec. So, at point B, both the X and Y axes are subjected to an "instantaneous" change of velocity of 10 inches/sec. The velocity change cannot of course be "instantaneous", and the controller will make that change of velocity (for each axis) in one millisecond, so the X axis will generate an acceleration of 10,000 inches/sec squared, and the Y axis will generate a deceleration of 10,000 inches/sec squared. However, if the BlendMaxAccelLinearIPS2 task parameter is set to a positive non-zero value less than 10,000 inches/sec squared, then, the controller will force deceleration to zero velocity at point B, therefore, avoiding the 10,000 inches/second squared acceleration/deceleration.

C.4.8.2. Limit Acceleration without Full Deceleration

If the BlendMaxAccelLinearIPS2 task parameter is set to a negative value, then the parameter's absolute value is used as an acceleration limit (for linear axes). However, instead of forcing the move to decelerate all the way to a stop (forcing a **G9**), it instead forces the move to decelerate only to a speed at which the acceleration limit is not exceeded.

EXAMPLE:

```
G70 G91 G1      ; inches, relative mode, linear moves
F600            ; 600 inches/minute
G108            ; tell controller to not stop between moves.
Y10             ; Move from point A to point B
X10             ; Move from point B to point C
```



From point A to point B, the X axis moves at 600 inches/min, or 10 inches/sec, and the Y axis moves at 0 inches/sec. But, from point B to point C, X travels at 0 inches/sec, and Y at 10 inches/sec. So, at point B, both the X and Y axes are subjected to a "instantaneous" change of velocity of 10 inches/sec. The velocity change cannot of course be "instantaneous", and the controller will make that change of velocity (for each axis) in one millisecond, so the X axis will generate an acceleration of 10,000 inches/sec squared, and the Y axis will generate a deceleration of 10,000 inches/sec squared. However, if the BlendMaxAccelLinearIPS2 parameter is set to 1,000 inches/sec squared, then the controller will force deceleration to a velocity of 60 inches/minute, at point B, therefore generating only a 1,000 inches/second squared acceleration/deceleration.

C.4.8.3. Blending Limitations of BlendMaxAccelLinearIPS2 and BlendMaxAccelRotaryDPS2 Task Parameters

There are certain conditions where a **G9** will be enforced, regardless of the value of this parameter. There are some instances where this parameter will not work properly, due to limitations in the look-ahead process.

If **G301** mode is active, the controller will look-ahead multiple moves, and slowdown many moves before the change in velocity. However, the maximum number of moves ahead of the CNC block, the controller is able to slow down as determined by the MaxLookAheadMoves task parameter. Also, there are also some other instances where this parameter will do nothing, due to limitations in the look-ahead process.

Note that a **G9** (decelerate to zero velocity) will be enforced, regardless of the value of this parameter, if you execute a **G92** command, are in **G109** mode, or have a **G9** on the line.

If a **G8** is on the CNC program line, this parameter will have no effect.

C.4.8.4. Calculating the value for the BlendMaxAccelLinearIPS2 and BlendMaxAccelRotaryDPS2 Task Parameters

These parameters should be set to the maximum value that does not cause excessive excitation to the mechanics, adversely affecting part quality. The exact value required for your system must ultimately be determined by trial and error; dependant upon your mechanics, support structure, part geometry, required accuracy, etc. They may be set to the lesser value of the two axes **ACCELRATE** axis parameters, as a very rough starting point. The **ACCELRATE** axis parameter must be converted from machine counts / second / second to inches / second / second, as follows:

BlendMaxAccelLinearIPS2 <= ACCELRATE / CntsPerInch

BlendMaxAccelRotaryDPS2 <= ACCELRATE / CntsPerDeg

C.4.9. BlendMaxAccelRotaryDPS2

Setting this task parameter non-zero allows the controller to automatically detect high accelerations caused by large changes in velocities between non-tangential contoured moves (**G108** mode), on rotary Type axes and scale down each of the axes component velocities to minimize acceleration. See the BlendMaxAccelLinearIPS2 task parameter for linear axes. The units of this parameter are acceleration, in inches per second squared. For best results blending motion with this parameter, it is recommended that rate based (**G68**) linear acceleration and deceleration (**G64**) be used.

This parameter may be used via the methods described below.

C.4.9.1. Force Rotary Axes Deceleration to Zero (G9), if Maximum Acceleration is exceeded

If the BlendMaxAccelRotaryDPS2 task parameter is set to a positive value, it causes the controller to execute a **G9** command (deceleration at the end of the CNC block) for each CNC block between contoured moves that involve a change in (a component) velocity, having a change in velocity greater than this parameter's value. This parameter is in RPM. If any rotary axis is subject to a change in its component velocity exceeding this value between two blended moves, the controller abandons blending, and behaves as if a **G9** command was present on the CNC line.

For example, if this parameter is set to 3 (RPM), any CNC program block followed by another, with a velocity of >6, or <0 (RPM), implying a direction change, will be forced to decelerate to zero velocity before executing (and accelerating up to the new velocity) in the following CNC block.

C.4.9.2. Limit Rotary Acceleration without Full Deceleration

If the BlendMaxAccelRotaryDPS2 task parameter is set to a negative value, then the parameter's absolute value is used as an acceleration limit. However, when this parameter value is negative, instead of forcing the move to decelerate all the way to a stop (forcing a **G9**), it instead forces the move to decelerate to a speed at which the velocity change (acceleration) is not violated.

For example, if this parameter is set to 3 (RPM), and a CNC program block follows with a direction reversal, the current CNC block will be limited to 1.5 and the following will be limited to -1.5 (1.5 in the opposite direction).

C.4.10. BlendMaxAccelCircleIPS2

This task parameter is used when traveling around an arc with a short radius and/or a high feedrate, either of the two axes in the arc may be subject to excessively high accelerations caused by large changes in velocity. If this parameter is non-zero, the controller will limit these accelerations, by slowing down (or clamping) the feedrate to a lower value during the arc.

This capability of clamping feedrates due to large accelerations caused during a circular interpolated contoured move is also known as "Feedrate Attenuation by Arc Radius". It should be contrasted with a related capability, BlendMaxAccelLinearIPS2, which clamps the feedrate due to accelerations caused by discontinuities between contoured moves, see Corners for more information. For best results blending motion with this parameter, it is recommended that rate based (**G68**) linear acceleration and deceleration (**G64**) be used.

The lower feedrate value enforced by this task parameter will be such that the largest axis acceleration/deceleration that will occur during the arc/circle, will be equal to the limit provided by this parameter plus the value of the AccelRateIPS2 task parameter (maximum acceleration during a 360 degree circle is F * F / R, where; F is the feedrate, and R is the circle radius). The units of the parameter value are inches / (second squared).

The controller computes the arc acceleration for an axis, as the maximum acceleration that the axis would be subjected to during an entire circle at the given arc radius, at the given feedrate. For arcs traveling through short angles, the actual acceleration an axis is subjected to may be smaller than that the controller calculates, causing the controller to slow down more than necessary. Note, that the maximum acceleration of an axis during an arc occurs when the arc is tangent to that axis. For example, the maximum acceleration of the X axis (F*F/R) during an arc occurs when the tangent to the arc is parallel to the X axis.

If **G301** mode is active, the controller will look-ahead multiple moves, and slowdown many moves before the change in velocity. However, the maximum number of moves ahead of the CNC block, the controller is able to slow down as determined by the MaxLookAheadMoves task parameter. Also, there are also some other instances where this parameter will do nothing, due to limitations in the look-ahead process.

C.4.10.1. Calculating the value of the BlendMaxAccelCircleIPS2 Task Parameter

This parameter should be set to the maximum value that does not cause excessive excitation to the mechanics, adversely affecting part quality. The exact value required for your system must ultimately be determined by trial and error; dependant upon your mechanics, support structure, part geometry, required accuracy, etc. The BlendMaxAccelCircleIPS2 task parameter may be set to the lesser value of the two axes **ACCELRATE** axis parameters, as a very rough starting point.

The **ACCELRATE** must be converted from machine counts / second / second to inches / second / second, as follows:

BlendMaxAccelCircleIPS2 <= ACCELRATE / CntsPerInch

C.4.11. CannedFunctionID

This task parameter causes the specified Canned Function to execute on the implied (current) or specified task. The Canned Function must first be defined via the **SETCANNEDFUNCTION** CNC command. It may also be executed via the **EXECANNEDFUNCTION** CNC command.

C.4.12. ChordicalSlowdownMsec

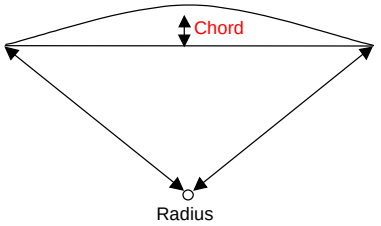
This task parameter affects the speed and accuracy of contoured CNC program blocks. This is accomplished by varying the Update Time of the CNC Profiler or the velocity of a CNC Program block. A CNC program block will always move at the lesser of the two velocities defined below.

The absolute value of this parameter (in milliseconds) determines the minimum time that a CNC program block may execute in. Attempting to exceed this limit will cause the velocity of the CNC block to be scaled down.

The sign, if negative, will force the velocity of **G2** / **G3** (**G12** / **G13**) commands to slow down only enough, such that the CNC Profiler is able to calculate 3 points during the arc. This guarantees that the arc will not be a linear segment, created from 2 points (starting and ending point).

C.4.13. ChordicalToleranceInch

This task parameter will disable warning messages for arc's (**G2**/**G3**/**G12**/**G13**) that execute in less than the time specified by UpdateTimeSec (creating a splined arc with only 2 points), whose chord distance is less than the value of this parameter. The chord distance is defined as the maximum distance from the center of a line (drawn from the starting point to the ending point of the arc) to the circumference of the arc. In the illustration below, the programmed arc will not be generated, but, the linear move segment will be generated from the start point to the end point.



When a line is generated instead of an arc, acceleration limiting will not occur for the angle of the line, it will compensate only for the programmed arc. If the line generated, is non-tangential to the previous move, over-shoot will occur.

An error (not just a warning) may be generated on this condition by setting the ThrowWarningsAsTaskFaults task parameter to one.

C.4.14. CommandVelocityVariance

This task parameter will, in certain situations (during deceleration within a series of short contoured moves) due to the limitations of integer times (in milliseconds) the CNC Profiler will produce smoother contours, if it could produce non-linear behavior of velocity vs. time during one particular time slice. Increasing the value of this parameter will reduce the frequency of extra long or extra short slices, which may improve the smoothness of high speed machining profiles.

C.4.15. Coord1Plane

This task parameter specifies which plane is active in coordinate system 1. Coordinate system 1 is used for the circular G-codes **G2** and **G3**. Plane 1 consists of the axis specified by Coord1I and Coord1J task parameters. Plane 2 is defined by Coord1J and Coord1K task parameters. Plane 3 is defined by Coord1K and Coord1I task parameters. Any circular motion performed in coordinate system 1 must be in the active plane. The active plane determines the default axes for incomplete targets of circular moves, refer to the figure below. This parameter is 1 based, where 1 represents plane 1, comprised of the axes defined by the Coord1I and Coord1J task parameters, as shown in Figure C-12. Refer to Figure C-13 also.

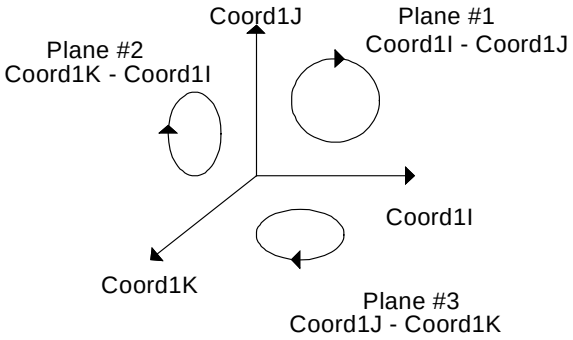


Figure C-12. Coordinate System 1 (Clockwise or G2 motion)

C.4.15.1. Clockwise Circular Axes Plane

By default, this command will produce circular motion in the X and Y axis plane as shown in Figure C-13. However, this command can produce motion in any one of three planes, where, each plane is defined by two of the three axes of the coordinate system. The three axes of coordinate system #1 and #2 are defined by the **G16** and **G26** commands, respectively. The 2 axes plane that is used within coordinate system 1 and 2, is selected by the **G17**/**G18**/**G19** and **G27**/**G28**/**G29** commands, respectively.

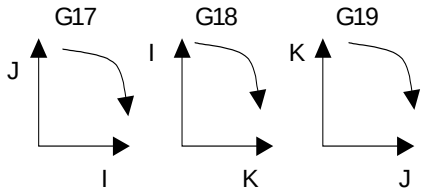


Figure C-13. Orientation of G2, in various planes in Coordinate System #1

C.4.16. Coord1I

This task parameter specifies which task axis is the Coord1I axis for coordinate system 1. This axis must be a linear type. The task X axis does not have to be the Coord1I axis. Coordinate system axes Coord1I, Coord1J, and Coord1K are used only as axes identifiers of a three dimensional system. Any task axis can be assigned to any of the three coordinate system axes. This mapping in conjunction with the selected plane (see Coord1Plane parameter) determine which task axes can be used for circular motion. This parameter is 0 based, where 0 represents the first axis, typically named X.

The **G16** command may be used to assign this parameter more easily within a CNC program however, it will not be saved to the task parameter .Ini file, requiring it to be set within all CNC programs.

C.4.17. Coord1J

This task parameter specifies which task axis is the Coord1J axis for coordinate system 1. This axis must be a linear type. The task Y axis does not have to be the Coord1J axis. Coordinate system axes Coord1I, Coord1J, and Coord1K are used only as axes identifiers of a three dimensional system. Any task axis can be assigned to any of the three coordinate system axes. This mapping in conjunction with the selected plane (see Coord1Plane parameter) determine which task axes can be used for circular motion. This parameter is 0 based, where 0 represents the first axis, typically named X.

The **G16** command may be used to assign this parameter more easily within a CNC program however, it will not be saved to the task parameter .Ini file, requiring it to be set within all CNC programs.

C.4.18. Coord1K

This task parameter specifies which task axis is the Coord1K axis for coordinate system 1. This axis must be a linear type. The task Z axis does not have to be the Coord1K axis. Coordinate system axes Coord1I, Coord1J, and Coord1K are used only as axes identifiers of a three dimensional system. Any task axis can be assigned to any of the three coordinate system axes. This mapping in conjunction with the selected plane (see Coord1Plane parameter) determine which task axes can be used for circular motion. This parameter is 0 based, where 0 represents the first axis, typically named X.

The **G16** command may be used to assign this parameter more easily within a CNC program however, it will not be saved to the task parameter .Ini file, requiring it to be set within all CNC programs.

C.4.19. Coord2Plane

This task parameter specifies which plane is active in coordinate system 2. Coordinate system 2 is used for the circular G-codes **G12** and **G13**. Plane 1 consists of the axis specified by Coord2I and Coord2J task parameters. Plane 2 is defined by Coord2J and Coord2K task parameters. Plane 3 is defined by Coord2K and Coord2I task parameters. Any circular motion performed in coordinate system 2 must be in the active plane. The active plane determines the default axes for incomplete targets of circular moves, refer to the figure below. This parameter is 1 based, where 1 represents plane 1, comprised of the axes defined by the Coord2I and Coord2J task parameters, Figure C-14. Refer to Figure C-15 also.

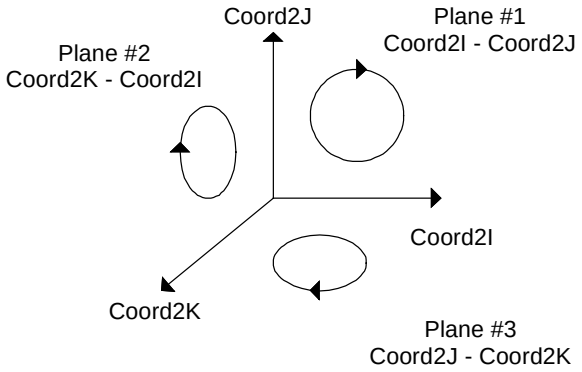


Figure C-14. Coordinate System 2 Orientation (Clockwise or G2 Motion)

C.4.19.1. Clockwise Circular Axes Plane

By default, this command will produce circular motion in the X and Y axis plane as shown in Figure C-15. However, this command can produce motion in any one of three planes, where, each plane is defined by two of the three axes of the coordinate system. The three axes of coordinate system #1 and #2 are defined by the **G16** and **G26** commands, respectively. The 2 axes plane that is used within coordinate system 1 and 2, is selected by the **G17**/**G18**/**G19** and **G27**/**G28**/**G29** commands, respectively.

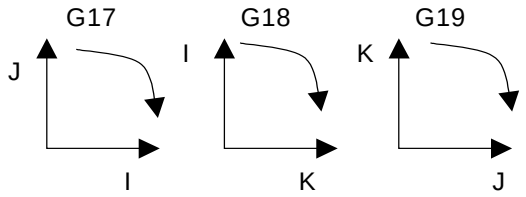


Figure C-15. Orientation of G2, in various planes in Coordinate System #2

C.4.20. Coord2I

This task parameter specifies which task axis is the Coord2I axis for coordinate system 2. This axis must be a linear type. The task X axis does not have to be the Coord2I axis. Coordinate system axes Coord2I, Coord2J, and Coord2K are used only as axes identifiers of a three dimensional system. Any task axis can be assigned to any of the three coordinate system axes. This mapping in conjunction with the selected plane (see Coord2Plane parameter) determine which task axes can be used for circular motion. This parameter is 0 based, where 0 represents the first axis, typically named X.

The **G26** command may be used to assign this parameter more easily within a CNC program however, it will not be saved to the task parameter .Ini file, requiring it to be set within all CNC programs.

C.4.21. Coord2J

This task parameter specifies which task axis is the Coord2J axis for coordinate system 2. This axis must be a linear type. The task Y axis does not have to be the Coord2J axis. Coordinate system axes Coord2I, Coord2J, and Coord2K are used only as axes identifiers of a three dimensional system. Any task axis can be assigned to any of the three coordinate system axes. This mapping in conjunction with the selected plane (see Coord2Plane parameter) determine which task axes can be used for circular motion. This parameter is 0 based, where 0 represents the first axis, typically named X.

The **G26** command may be used to assign this parameter more easily within a CNC program however, it will not be saved to the task parameter .Ini file, requiring it to be set within all CNC programs.

C.4.22. Coord2K

This task parameter specifies which task axis is the Coord2K axis for coordinate system 2. This axis must be a linear type. The task Z axis does not have to be the Coord2K axis. Coordinate system axes Coord2I, Coord2J, and Coord2K are used only as axes identifiers of a three dimensional system. Any task axis can be assigned to any of the three coordinate system axes. This mapping in conjunction with the selected plane (see Coord2Plane parameter) determine which task axes can be used for circular motion. This parameter is 0 based, where 0 represents the first axis, typically named X.

The **G26** command may be used to assign this parameter more easily within a CNC program however, it will not be saved to the task parameter .Ini file, requiring it to be set within all CNC programs.

C.4.23. CutterRadiusInch

This task parameter specifies the cutter compensation radius in inches. When enabled, the contoured motion path is compensated by this amount to account for the cutting tool's radius. The CutterX and CutterY parameters specify the compensated plane of motion.

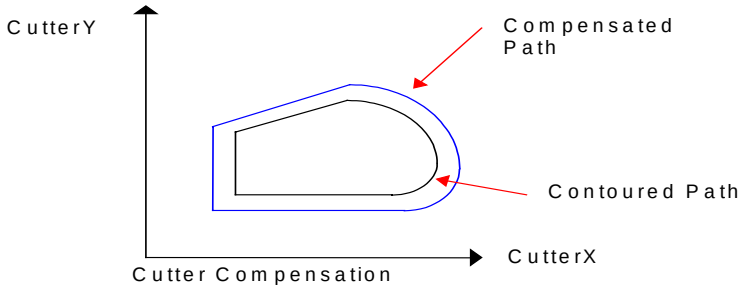


Figure C-16. Cutter Compensation Radius

C.4.24. CutterToleranceDeg

This task parameter may be used to avoid short circular arc segments known as link moves when operating in cutter compensation, around "outside corners" between nearly parallel moves. These short link moves may cause Motion Queue Starvation, or the error message, "Link Arc Degenerates to Line", usually due to the Imprecision of Target Values.

If during cutter compensation, the angle between two moves on an outside corner will be less than the angle specified by this parameter, then, instead of generating a link move between points A and C, around the angle θ, the controller will instead execute an offset move, and follow that path. For example travel through point A to point B, then through point C, as illustrated in the picture.

Large values of CutterToleranceDeg can be dangerous. The offset method is desirable when the angle between move 1 and move 2 is small, as discussed above. However, the "offset method", is less desirable, the larger the angle between the moves, because it will also cause the tool to leave the part temporarily (at point b the tool will not contact the part). This departure increases dramatically with move angle, and in fact if the angle between move 1 and move 2 is 180 degrees, the offset method yields an infinite departure, and the controller reverts to the link method regardless of the value of the value of CutterToleranceDeg.

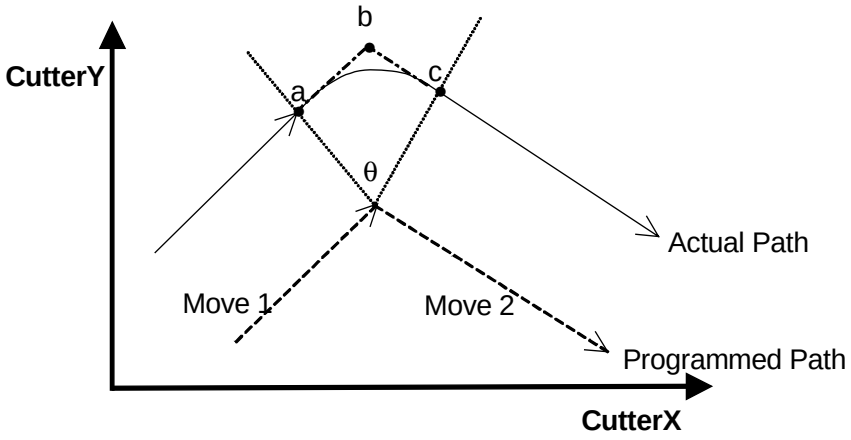


Figure C-17. Cutter Compensation Illustration

C.4.25. Link Move

A link move is generated by the CutterX and CutterY axes during Cutter Compensation such that a round tool will produce a square corner. The link move is from point A to point C, as illustrated in Figure C-17. The tool is always in contact with the part.

C.4.26. Offset Move

An offset move eliminates the link move during Cutter Compensation, preventing a small radius from being attempted at high cutting speeds. The offset move is from point A to point B to point C, as illustrated in Figure C-17. While at point B, the tool is not in contact with the part.

C.4.27. Link Method

The link method, generates a small arc around a square corner during Cutter Compensation such that a circular tool will actually cut a square corner, as illustrated in Figure C-17. This method maintains contact between the tool and the part.

C.4.28. Offset Method

The offset method, generates an offset move during Cutter Compensation causing the tool to move off the part, but eliminating the small radius created by the Link method, as illustrated in Figure C-17. This method does not maintain contact between the tool and the part.

C.4.29. CutterX

This task parameter specifies which task axis is used for the X axis of the cutter compensation plane. This axis must be a linear type. The CutterX and CutterY parameters are used to determine the cutter compensation plane. When enabled the contoured motion will be compensated by the CutterRadiusInch parameter to account for the cutting tool's radius.

This parameter may not be changed while cutter compensation is active.

C.4.30. CutterY

This task parameter specifies which task axis is used for the Y axis of the cutter compensation plane. This axis must be a linear type. The CutterX and CutterY parameters are used to determine the cutter compensation plane. When enabled the contoured motion will be compensated by the CutterRadiusInch parameter to account for the cutting tool's radius.

This parameter may not be changed while cutter compensation is active.

C.4.31. CutterY

This task parameter specifies which task axis is used for the Z or "tool length" axis used by the cutter offset compensation. This parameter may not be changed while cutter offset compensation is active.

C.4.32. DecelOnProgramAbortMask

This task parameter is an axis mask. Any axes specified in the mask will follow the **DECEL** and **DECELMODE** axis parameters when the Abort key is selected within the UNIDEX 600 MMI. If an axes is not specified within this parameter to decelerate on aborting a CNC program, it will abruptly stop (without deceleration).

C.4.33. DecelRate

This task parameter will indicate or may be used to specify the current Deceleration rate of the task in the current user units. This parameter is inconsequential, if rotary axes are dominant. In this case, refer to the DecelRateDPS2 task parameter.

C.4.34. DecelRateDPS2

This task parameter specifies the deceleration rate in degrees per sec per second used by the controller for contoured motion deceleration (**G1**, **G2**, **G3**, **G12**, and **G13**), when the CNC **G68** command is active. Otherwise, **G67** will be active causing the deceleration to be time based and DecelTimeSec will be used. The linear AccelRateIPS2 /DecelRateIPS2 are used for linear dominant moves while the rotary AccelRateDPS2 */DecelRateDPS2* rates are used for rotary dominant moves. Refer to the **G98** command for information on rotary/linear axes dominance.

When accelerating/decelerating in rate based mode, the controller will not perform the acceleration/deceleration in a time period that is shorter than the UpdateTimeSec task parameter value. Therefore, for high rates of acceleration/deceleration and/or acceleration/decelerations between very similar speeds, the rate can be less than that specified.

C.4.35. DecelRateIPS2

This task parameter specifies the deceleration rate in degrees per sec per second used by the controller for contoured motion deceleration (**G1**, **G2**, **G3**, **G12**, **and G13**), when the CNC **G68** command is active. Otherwise, the acceleration will be time based and DecelTimeSec will be used. The linear AccelRateIPS2 /DecelRateIPS2 are used for linear dominant moves while the rotary AccelRateDPS2 /DecelRateDPS2 rates are used for rotary dominant moves. Refer to the **G98** command for information on rotary/linear axes dominance.

When accelerating/decelerating in rate based mode, the controller will not perform the acceleration/deceleration in a time period that is shorter than the UpdateTimeSec task parameter value. Therefore, for high rates of acceleration/deceleration and/or acceleration/decelerations between very similar speeds, the rate can be less than that specified.

C.4.36. DecelTimeSec

This task parameter specifies the deceleration rate in seconds used by the controller for contoured motion deceleration. The **G67** mode must be set, otherwise **G68** will be active and the deceleration will be rate based and the DecelRate parameters will be used.

C.4.37. DryRunLinearFeedRateIPM

This task parameter is used as the feedrate for linear Type axes in the Dry Run mode, **G116**. The units are in IPM. This parameter has no effect on **G0** commands or asynchronous motion commands.

C.4.38. DryRunRotaryFeedRateRPM

This task parameter is used as the feedrate for rotary Type axes in the Dry Run mode, **G116**. The units are in RPM.This parameter has no effect on **G0** commands or asynchronous motion commands.

C.4.39. ErrCode

This task parameter is set by the indicated commands (such as **FILEOPEN**, **DATASTART**, etc.) to return an error code to the CNC program for unsuccessful commands. It must be tested immediately following a command, which indicates it sets this task parameter. Otherwise, another command may subsequently be executed and successfully completed by setting this variable to a non-error value, masking the first error.

The meaning of the value of this parameter can be determined by setting the TaskFault task parameter equal to the value of the ErrCode. If this is done during CNC program execution, this will however halt the CNC program.

C.4.40. EStopInput

This task parameter specifies which binary input is used as an E-Stop input for this task. A value of -1 disables the task E-Stop check. The global E-Stop will still be tested even if this parameter is disabled. Each task may have a separate E-Stop input. This is a level sensitive input that is sampled at the rate indicated by the AvgPollTimeSec global parameter. This input is active high.

Be sure to set the ESTOP bit in the **FAULTMASK** axis parameter to enable the detection of this fault, then set the bit in the appropriate mask parameter (**DISABLEMASK**, **HALTMASK**, **AUXMASK**, **ABORTMASK**, **INTMASK** and **BRAKEMASK**) for the action to occur on this fault.

C.4.41. ExecuteNumLines

This task parameter controls the relative program execution priority of the tasks, and the priority of task program execution relative to the other chores a task performs. Normally, it is used to speed up execution of a particular "high priority task", or to avoid profile queue starvation during a series of short contoured moves.

The controller has four tasks available for running programs. However each task can also perform a number of functions in addition to program execution; such as ESTOP and

MFO monitoring, watch monitoring (see ON and ONGOSUB CNC commands), and immediate command execution. Normally, the execution of tasks is performed as follows:

FOR EACH TASK (4)

Generate an interrupt back to PC, as required.
Check ESTOP, MFO, MSO, and feedhold, react as required.
Monitor spindle and asynchronous motion (if any).
Test ONGOSUBS and ON's, execute the indicated action, as required.
Test for RIAction1 or ROAction1 commands.
Execute canned functions, if commanded to.
Execute external jogging, if commanded to.
Execute immediate command (if any requested).
Execute the next CNC statement (if that CNC is running a program), or if a statement was still running (like a G1, or G4) then check to see if its finished or not.

END FOR

The ExecuteNumLines parameter can be used to change the above behavior as follows. Note, that the default value for ExecuteNumLines is one.

If ExecuteNumLines = 0 for a task, then that task is inactive (none of the chores listed above, including program execution and ESTOP monitoring, are executed for that task. Setting ExecuteNumLines to 0 for a given task, disables that task, significantly increasing the execution speed of the other tasks. However, when ExecuteNumLines is zero, none of the steps listed in the "for loop" above, are executed for that task.

If ExecuteNumLines > 0 for a task, then that task will execute all the steps in the "For Loop" above, and try to execute that many lines in the CNC program, before giving control to the next task. However, if an error occurs at any time during CNC program execution, or the statement can not be completed immediately (a **G1**, **G2**, **G3**, **G4**, **M3**, etc. or any other statement requiring a certain amount of time), then that task will stop trying to execute lines and release execution to the next task. Setting ExecuteNumLines > 0 for a given task, increases the execution speed of that task relative to the other tasks, if it is executing CNC statements that can be completed immediately.

If ExecuteNumLines < 0 for a task, then it will behave as described for ExecuteNumLines > 0 (with the number of lines to execute being the absolute value of the parameter value), except, that it will not stop executing for a statement that can not be completed immediately (a **G1**, **G2**, **G3**, **G4**, **M3**, etc.). These settings are more drastic than their corresponding positive values, and will increase the execution speed of the current task. However, negative values are drastic, and high negative values will starve everything else, including library calling applications like the MMI600. High negative values will cause the MMI600 to freeze up.

The user should be warned that changing this parameter (especially to negative values) can adversely effect the performance of any critical controller function, such as I/O or ESTOP monitoring, as well as the update rate of the positions in the UNIDEX 600 MMI. The user must understand the tradeoffs involved, and be willing to accept reduced performance is some areas, in order to obtain the increased performance in others.

C.4.42. ExecuteNumMonitors

This task parameter controls the execution priority of the monitor commands, relative to task program execution. Normally, the task will test all "ON" monitors for a given task, once each task cycle. If this is set non-zero, then it will test only that number of monitors each task poll cycle. For example, if this is set to 3, and there are 8 monitors, then the first task cycle will test monitors 1, 2 and 3, the second task cycle will test monitors 4, 5, and 6, the third task cycle will test monitors 7, and 8. The fourth task cycle will test monitors 1, 2, and 3, and so on.

The time to complete one task cycle may be determined by the AvgPollTimeSec global parameter.

C.4.43. ExecuteNumSpindles

This task parameter defines the number of spindles available to each task. You must increase this from 1, in order to use one of the three additional spindles. For example, this must be set to 2, 3, or 4 to use Spindle #2. Keep this value as low as possible, in order to improve task polling and profiling efficiency.

C.4.44. FeedHold

This task parameter provides software feedhold control of a task. A value of "1" activates feedhold and a value of zero disables feedhold, or releases it. This parameter's feedhold action is logically "OR" ed with the state of the hardware feedhold input (if it is defined). This implies either feedhold (software or hardware) will enable feedhold and not release it until both (software and hardware) inputs are false. See FeedHold Operation for more details. This task parameter is sampled at the rate indicated by the AvgPollTimeSec global parameter.

The FeedHoldEdgeInput has precedence over the FeedHoldInput and FeedHold task parameters, so if the FeedHoldEdgeInput is not set to –1, the FeedHoldInput and FeedHold task parameters will be ignored.

C.4.45. FeedHoldEdgeInput

This task parameter specifies which binary input is used as a feedhold input for this task. This input is treated as a positive edge sensitive signal (i.e. a push button). A value of -1 disables the feedhold edge sensitive check. Each task may have a separate feedhold edge sensitive input. This task parameter is sampled at the rate indicated by the AvgPollTimeSec global parameter.

The FeedHoldEdgeInput has precedence over the FeedHoldInput and FeedHold task parameters, so if the FeedHoldEdgeInput is not set to –1, the FeedHoldInput and FeedHold task parameters will be ignored.

C.4.46. FeedHoldInput

This task parameter specifies which binary input is used as a feedhold input for this task. This input is treated as a level sensitive signal. A value of -1 disables the feed hold check. Each task may have a separate feed hold input. The FeedHold Input takes precedence over the FeedHold task parameter. In other words, if he FeedHoldInput is not set to –1,

the FeedHold task parameter is ignored. This task parameter is sampled at the rate indicated by the AvgPollTimeSec global parameter.

C.4.47. GlobalEstopDisabled

This task parameter will disable the Global opto-isolated E-stop input from halting the program running on this task, if it is set true (1).

C.4.48. HaltTaskOnAxisFault

If this task parameter is set to 1, then the task will generate a 'Physical Axis Fault' when any axis fault occurs on an axis bound to that task. This allows the user to cause a CNC program to halt when an axis fault occurs. If the parameter is set to zero, axis faults will not generate task faults, which would stop the CNC program on that task.

C.4.49. IgnoreAxesMask

This task parameter is used for a form of debugging, which allows you to run a CNC program without moving certain axes. Its value indicates a set of axes that will be ignored during motion. That is, the controller will behave as though the indicated axes were not in the move command. This parameter applies to all motion, including synchronous, asynchronous, and camming.

It should be differentiated from the **SIMULATION** axis parameter, in that the move command will not take the time necessary to move the axis (see example below, specifically the difference between behavior of X and Z axes.).

Example:

```
G91 G70 F10              ; 10 inches/minute
IgnoreAxesMask = 1       ; X axis is ignored
SIMULATION.Z = 1         ; Z axis is in simulation
G1 X10                   ; This command is ignored, takes no time to execute
G1 X10 Y10               ; X ignored in this command, Y moves, command
                         ; takes 1 minute
G1 Z10                   ; Z axis is not moved, but this command takes 1
                         ; minute to execute
G1 Z10 Y10               ; Z axis not moved, Y moves, command takes 1.414
                         ; minutes
```

C.4.50. InterruptMotion

Setting this task parameter to 1 will cause the CNC to feedhold all axes bound to the task. After feedhold has completed, you may move the axes via any valid Immediate Mode commands. The interrupt mode may be exited, by resetting the InterruptMotion task parameter back to zero. The way that the interrupt mode exits, depends on the setting of the InterruptMotionReturnType task parameter. If the InterruptMotionReturnType task parameter defines that interrupted motion is to be resumed, then feedhold is released.

C.4.51. InterruptMotionReturnType

This task parameter defines the way the axes return, after their motion was interrupted via the InterruptMotion task parameter. After axes motion has been interrupted, you may move the axes via any valid Immediate Mode commands. The default return type is 0. Typically, return type 2 or 4 would be used to maintain programmed position.

0 - RETURNTYPE_NULL

This return type will not return the axes to the absolute positions that they were at, at the time when the interrupt occurred. The motion command that was interrupted will not be completed upon return from the interrupt. Program execution will continue on the CNC line after the CNC line that was interrupted.

1 - RETURNTYPE_START

This return type will return the axes to the absolute positions that they were at, at the start of the interrupted line. Program execution will continue on the CNC line that was interrupted, restarting the interrupted CNC command line from the start of the line.

2 - RETURNTYPE_INTERRUPT (used by the Jog and Return Mode)

This return type will return the axes to the absolute positions that they were at, when the interrupt occurred. The interrupted move will be completed upon return from the interrupt and program execution will continue on the CNC line after the interrupted line.

3 - RETURNTYPE_END

This return type will return the axes to the absolute positions that they would have been at, at the end of the interrupted line, had it not been interrupted. Program execution will continue on the CNC line after the interrupted line.

4 - RETURNTYPE_OFFSET (used by the Jog and Offset Mode)

This return type will not return the axes to the absolute positions that they were at, at the time when the interrupt occurred. The interrupted move will be completed upon return from the interrupt and program execution will continue on the CNC line after the interrupted line.

C.4.52. External Jog Key Example

All of the following applies to the JogPair2Axis1, JogPair2Axis2, JogPair2Mode, etc. task parameters also, which allow two other axes to be jogged simultaneously.

Define the first axis via the JogPair1Axis1 task parameter.
Define the second axis via the JogPair1Axis2 task parameter.
Define the Jog Mode via the JogPair1Mode task parameter.
Define the virtual inputs for the Plus and Minus Inputs for Axis1.
Define the Plus and Minus Inputs for Axis 2.
Enable the jog keys via the JogPair1EnableIn task parameter.

All of these parameters may be changed on the fly.

Changing the axes, or jog mode on the jog page will overwrite these task parameters. Likewise, changing these parameters will change their complementary values on the jog page.

C.4.53. JogPair1Axis1MinusIn

This task parameter defines the virtual input number that when True (active high) will cause Axis 1 of Jog Pair 1 (as defined by JogPair1Axis1) to jog minus in the current jog mode (as defined by the JogPair1Mode task parameter).

C.4.54. JogPair1Axis1PlusIn

This task parameter defines the virtual input number that when True (active high) will cause Axis 1 of Jog Pair 1 (as defined by JogPair1Axis1) to jog plus in the current jog mode (as defined by the JogPair1Mode task parameter).

C.4.55. JogPair1Axis2MinusIn

This task parameter defines the virtual input number that when True (active high) will cause Axis 2 of Jog Pair 1 (as defined by JogPair1Axis2) to jog minus in the current jog mode (as defined by the JogPair1Mode task parameter).

C.4.56. JogPair1Axis2PlusIn

This task parameter defines the virtual input number that when True (active high) will cause Axis 2 of Jog Pair 1 (as defined by JogPair1Axis2) to jog plus in the current jog mode (as defined by the JogPair1Mode task parameter).

C.4.57. JogPair1EnableIn

This task parameter defines the virtual input number that when True (active high) will cause Jog Pair 1 (as defined by JogPair1Axis1) to move in the current jog mode (as defined by the JogPair1Mode task parameter). SecNum_JogPair1AxisExample

C.4.58. JogPair1Axis1

This task parameter defines the first axis that will be commanded to move by the virtual input specified by the JogPair1EnableIn task parameter. The value is specified as a bitmask. Bit 0 represents the first axis (default name X), bit 2 the second axis (default name Y), etc. See Section C.4.59.1. for more information on setting this parameter.

C.4.59. JogPair1Axis2

This task parameter defines the second axis that will be commanded to move by the virtual input specified by the JogPair1EnableIn task parameter. The value is specified as a bitmask. Bit 0 represents the first axis (default name X), bit 2 the second axis (default name Y), etc. See Section C.4.59.1. for more information on setting this parameter.

C.4.59.1. JogPair1Axis Example

For example, to assign axes Y and Z to the SlewPair1 or JogPair1Mask task parameters, you would set it to a value of 6. To see how this is done, click on each of the task axis numbers below to find its numeric value from the chart, and add them together to find the value to set the parameter to:

```
                Task Axis 2  (Y)  =      2
                Task Axis 3  (Z)  =      4
+               ____________________________
                Task Parameter    =      6          ; Y, Z axis pair
```

For the SlewPair# task parameters, this would specify that the horizontal axis of the joystick would command task axis 1 (Y) to move and the vertical axis of the joystick would command task axis 2 (Z) to move. For the JogPair#Mask task parameters this would define axis 1 as Y and axis 2 as Z. The lowest numbered task axis will be axis 1 or for the joystick, commanded by the horizontal axis of the joystick.

This order may be inverted by negating the sign of the value entered. For example;

```
                Task Parameter    =      -6         ; Z, Y axis pair
```

would specify that the horizontal axis of the joystick would command task axis 2 (Z) to move and the vertical axis of the joystick would command task axis 1 (Y) to move. This would also assign the Z axis as axis 1 and the Y axis as Axis 2.

C.4.60. JogPair1Mode

This task parameter defines the jog mode of the axes specified by the JogPair1Axis1 task parameter, as one of the following:

FreeRun = 0

The axes continue moving until the jog input is False.

Distance and Hold = 1

The axes move at the velocity and distance specified by the jog machine parameters. The motion begins when the input becomes True and will stop if it becomes False before the move completes.

Distance = 2

The axes move at the velocity and distance specified by the jog machine parameters. The motion begins when the input becomes True and does not stop unless mode 3, Halt is activated.

Halt = 3

The axes decelerate to a stop.

C.4.61. JogPair2Axis1MinusIn

This task parameter defines the virtual input number that when True (active high) will cause Axis 1 of Jog Pair 2 (as defined by JogPair2Axis1) to jog minus in the current jog mode (as defined by the JogPair2Mode task parameter).

C.4.62. JogPair2Axis1PlusIn

This task parameter defines the virtual input number that when True (active high) will cause Axis 2 of Jog Pair 2 (as defined by JogPair2Axis2) to jog minus in the current jog mode (as defined by the JogPair2Mode task parameter).

C.4.63. JogPair2Axis2MinusIn

This task parameter defines the virtual input number that when True (active high) will cause Axis 2 of Jog Pair 2 (as defined by JogPair2Axis2) to jog minus in the current jog mode (as defined by the JogPair2Mode task parameter).

C.4.64. JogPair2Axis2PlusIn

This task parameter defines the virtual input number that when True (active high) will cause Axis 2 of Jog Pair 2 (as defined by JogPair2Axis2) to jog plus in the current jog mode (as defined by the JogPair2Mode task parameter).

C.4.65. JogPair2EnableIn

This task parameter defines the virtual input number that when True (active high) will cause Jog Pair 2 (as defined by JogPair2Axis1) to move in the current jog mode (as defined by the JogPair2Mode task parameter).

C.4.66. JogPair2Axis1

This task parameter defines the first axis that will be commanded to move by the virtual input specified by the JogPair2EnableIn task parameter. The value is specified as a bitmask. Bit 0 represents the first axis (default name X), bit 2 the second axis (default name Y), etc. See Section C.4.59.1. for more information on setting this parameter.

C.4.67. JogPair2Axis2

This task parameter defines the second axis that will be commanded to move by the virtual input specified by the JogPair2EnableIn task parameter. The value is specified as a bitmask. Bit 0 represents the first axis (default name X), bit 2 the second axis (default name Y), etc. See Section C.4.59.1. for more information on setting this parameter.

C.4.68. JogPair2Mode

This task parameter defines the jog mode of the axes specified by the JogPair2Axis1 task parameter, as one of the following:

FreeRun = 0
: The axes continue moving until the jog input is False.

Distance and Hold = 1
: The axes move at the velocity and distance specified by the jog machine parameters. The motion begins when the input becomes True and will stop if it becomes False before the move completes.

Distance = 2
: The axes move at the velocity and distance specified by the jog machine parameters. The motion begins when the input becomes True and does not stop unless mode 3, Halt is activated.

Halt = 3
: The axes decelerate to a stop.

C.4.69. JoyStickPort

The JoyStickPort task parameter specifies the joystick port to be used when the joystick is activated, via the CNC Slew command or the RIAction1 task parameter. The default value is 0, for the joystick port on the UNIDEX 600 card to be used. Optionally, the joystick port may be used on the 4EN-PC cards configured as boards 1 through 3, by setting the parameter to 1 through 3, respectively.

C.4.70. LinearFeedRate

This task parameter is the programmed vectorial speed of linear axes in contoured motion (**G1**, **G2**, **G3**, **G12**, **and G13**). It is normally specified in user-units per minute. User-units are inches in **G70** mode, and millimeters in **G71** mode. See the F word documentation, for more details on the use of this parameter. However, note that the units of this parameter will be changed by the UserFeedRateMode task parameter (the **G93** /**G94**/ **G95** mode) as shown below:

UserFeedRateMode		**G93** /**G94** / **G95** mode	Units of the F word
0	DEFAULT	**G94**	user-units / minute
1		**G93**	minutes / user-unit
2		**G95**	user-units / spindle1-revolution
3		**G295**	user-units / spindle2-revolution
4		**G395**	user-units / spindle3-revolution
5		**G495**	user-units / spindle4-revolution

The vectorial speed of linear axes in contoured motion is the square root of the sum of the squares of all the speeds of the linear type axes involved in the contoured motion. This parameter value is not signed: its value must always be positive.

Note, that the LinearFeedrate task parameter value has no effect on the contoured motion of rotary axes (see the RotaryFeedRate task parameter), nor does it effect motion on linear axes that are being controlled by the rotary axes motion component (see **G98**/**G99**).

The LinearFeedrate task parameter value is a programmed value, and will retain its value whether there is currently any contoured motion. However, due to feedrate limiting and MFO, the actual vectorial velocity during a contoured move may differ from the programmed vectorial velocity. The LinearFeedRateActual task parameter indicates the actual vectorial velocity.

If you change the LinearFeedRate during a contoured move, this change will not take effect until the next contoured move. Use the MFO to change speed during a contoured move.

The CNC "F word " is equivalent to the LinearFeedRate parameter, and the value of this parameter can be observed from the MMI600 in the "F" window of the active G code section of the Run or manual screens.

C.4.71. LinearFeedRateActual

This task parameter indicates the actual vectorial feedrate of linear axes, in user units per second. This is the same as the commanded vectorial feedrate, (task parameter RotaryFeedRateActual) unless the speeds of the axes involved are limited by the MaxFeedRateIPM and MaxFeedRateRPM machine parameters, or the MFO is set to less than 1, in these cases it is less. Note, that the units of this parameter will be changed by the UserFeedRateMode task parameter. Also, this parameter only denotes the vectorial speed during contoured (**G1**, **G2**, **G3**, etc.) moves, and is zero during all other types of motion. This parameter is updated every 10 millisecond's.

C.4.72. LineNumberUser

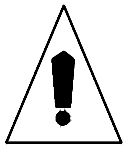
This task parameter indicates the current CNC program line number.

C.4.73. LineNumber960

This task parameter indicates the current CNC program line number on the controller. This most likely will be different than the LineNumberUser task parameter due to the fact that one user CNC program line occupies multiple lines on the controller. This parameter is for internal use only.

C.4.74. MaxCallStack

This task parameter specifies the maximum number of call stack elements available for this task. Every subroutine or program call pushes one element onto the call stack. Every return pops one element off the call stack. Each call stack has its own set of call stack parameters.



This parameter should only be set from the Setup page of the UNIDEX 600 MMI or from a C library function call, never from within a CNC program.

C.4.75. MaxLookAheadMoves

This task parameter defines the number of CNC program blocks which the CNC Block Look-Ahead process, will use to plan the (acceleration/deceleration) motion profile for contoured motion. The maximum value of this parameter is limited by the value at which motion queue starvation occurs.

It specifically serves two different purposes, depending on whether Multi-Block LookAhead (**G301**) mode is active.

If Multi block Look-Ahead mode is disabled, the controller only looks one CNC program contoured motion statement ahead in the program. However, it will look through a number of non-contoured motion statements in order to find that motion statement. This parameter is the maximum number of non-contoured motion statements it will look ahead, in order to find the "next" contoured motion statement. If the specified number of non-contoured motion statements is reached before finding a contoured-motion statement, the controller declares that there is no next motion statement.

C.4.76. MaxModeStack

This task parameter specifies the maximum number of mode stack elements available for this task. Task modes can be pushed and popped off this stack for storing and restoring modes.

C.4.77. MaxMonitorData

This task parameter specifies the maximum number of ON statements available for programs on this task.

C.4.78. MaxOnGosubData

This task parameter specifies the maximum number of OnGosub statements available for programs on this task.

C.4.79. MaxRadiusAdjust

This task parameter specifies the maximum number of degrees deviation of an arc (G2 / G3) from 360 degrees, for which the center point will not be adjusted. Normally, when an arc is specified with imprecision in the center point and target values (usually due to truncation of the target values by a "post-processor program), a new center point for the arc will be calculated (See MaxRadiusError).

However, if this imprecision is present in an arc whose angular travel is near 360 degrees (i.e. the starting and ending points are nearly identical). A small imprecision can be "leveraged" into a large difference in the circle center. In these cases we want to suppress the circle center readjustment algorithm. The user should never have to change this parameter's value, as the only time it is relevant, is when the post-processor program erroneously produces a 360 degree circle, but with unequal starting and ending point values.

C.4.80. MaxRadiusError

This task parameter specifies the maximum error that may exist between the specified radius at the starting point and ending points of a **G2** / **G3** / **G12** / **G13** command before a fault will occur. The units are in percent (%), with the default being 5%. Circular Radius Errors provides more information on this subject. All errors, less than the value of this parameter will have a new center point calculated, reducing their error to zero. Errors greater than this parameter will generate a fault.

You may also set this parameter to –1, which deactivates the Circular Radius Error Test, which requires the computation of a square root, and in the cases of high speed machining, this will speed up profile generation and avoid CNC Profile Queue Starvation Errors.

The Circular Radius Error test is only performed when specifying arcs (**G2** / **G3**) with the IJK method.

It is recommended that the Circular Radius Error test be left active until the program is completely debugged. Turning it off allows the user to write invalid (non-circular) **G2** / **G3** commands without any warning being issued to the user.

C.4.81. MFO

This task parameter sets the Manual Feedrate Override ratio for this task. This ratio can vary from 0 to infinity, where 1 is 100% of the programmed feedrate and 0% is zero feedrate. The MSO changes on the MMI600 screens are more restricted. This ratio is multiplied by the programmed feedrate for all rapid (**G0**), contoured (**G1**, **G2**, **G3**, **G12**, **G13**) and asynchronous motion (INDEX, MOVETO, etc., except async. home) in order to obtain the true feedrate of the move. G0 type moves will not exceed 100% of the feedrate, as determined by the RapidFeedRateIPM (RapidFeedRateRPM for rotary axes). This parameter will not affect camming motion (SYNC command). This parameter will not effect contoured moves (**G1**, **G2**, **G3**, **G12**, and **G13**) if the move is in the "Decel phase" (is currently decelerating to zero speed at the end of the move). This parameter is sampled at the rate indicated by the AvgPollTimeSec global parameter.

This parameter does not affect homing moves or a spindle velocity. The MSO affects the spindle. This parameter cannot be changed if the MFOLock task mode bit is active. Also, if the AnalogMFOInput task parameter is not -1, then the controller over-writes this parameter with the value obtained from the specified analog input. Additionally, this parameter reflects only the requested feedrate override, the MaxFeedRateIPM or MaxFeedRateRPM machine parameters may limit the maximum feedrate. This will not be reflected in the MFO value.

Normally, the operation of a MFO change during a contoured move will be delayed by a time interval of (UpdateNumEntries * UpdateTimeSec) seconds. By default, this will be 300 milliseconds.

The acceleration/deceleration during MFO changes is the same as the acceleration/deceleration followed at the beginning and end of the current move, which is determined by the motion type, as defined in the Accel/Decel Overview.

C.4.82. Mode1

This task parameter is a bitmask. Each bit represents a current state of the task. In general, these states are modes that are toggled by G codes. For example, the current **G70** /**G71** or English/Metric programming mode state is indicated by Bit 0. However, a few modes, such as Auto/Single Step, are not accessible via G codes.

You may view the state of most modal G codes from the MMI600 within the G-Code Display.

The CNC programmer may also toggle these bits using the Mode1 task parameter. For example, 'Mode1 = Mode1 || 0x3' would be equivalent to '**G70 G90** '. However, when setting the modes directly, you should be sure to ''logically or'' the new bits, so as to not reset all the other modes.

The user should consult the G code listed in the right column of the table below, for more information on the modal G code.

Table C-11. Mode1 Bit Descriptions

Mode 1 Task Parameter			
Bit #	**Description**	**Hex. Value**	**G Code (On/Off)**
0	English Programming Mode Active	0x1	G70 /G71
1	Absolute Programming Mode Active	0x2	G90 /G91
2	Linear Acceleration Mode Active	0x4	G64 /G63
3	Rate Based Acceleration Active	0x8	G68 /G67
4	Rotary Axis Feedrates Dominant (E word)	0x10	G98/G99
5	Motion Continuous	0x20	G108 /G109
6	Inverse Circular	0x40	G111 /G110
7	Spindle Shutdown	0x80	G101 /G100
8	Block Delete Active	0x100	G112 /G113
9	Optional Stop Active	0x200	G114 /G115
10	Breakpoint Active	0x400	N.A.
11	MFO Lock on	0x800	M48 /M49
12	MSO Lock on	0x1000	M50 /M51
13	Dry Run Mode	0x2000	G116 / G117
14	Retrace On	0x4000	Retrace
15	Auto Mode	0x8000	Auto/Single
16	Program FeedRate Min./Unit	0x10000	G93 /G94
17	Program FeedRate Units/Rev.	0x20000	Gx95 /G94
18	Program SfeedRate Surface Spindle #1	0x40000	G97 /G96
19	Program SfeedRate Surface Spindle #2	0x80000	G297 /G296
20	Program SfeedRate Surface Spindle #3	0x100000	G397 /G396
21	Program SfeedRate Surface Spindle #4	0x200000	G497 /G496
22	Block Delete2 Active	0x400000	G212 / G213
23	Run Over Mode	0x800000	Step Into/Step Over
24	Multiple CNC Program Block Look Ahead	0x1000000	G301 / G300
25	Machine Lock	0x2000000	M41 / M42
26	High Speed Machining	0x4000000	G311 / G310
27	Wait till In-Position Mode Active	0x8000000	G361 / G360
28-31	Unused		

Each of these bits indicate a modal state is active when its respective bit is true. If the bit is false it implies that the opposing modal state is active, such as:

Bit 0, False, implies Metric Programming Mode is active
Bit 1, False, implies Incremental programming mode is active
Bit 2, False, implies Sinusoidal (1-Cosine) acceleration is active,
Bit 3, False, implies Time based acceleration is active,
Bit 4, False, implies Linear Axes Feedrates are dominant,
etc..

C.4.83. MSO

This task parameter sets the Manual Spindle feedrate Override for Spindle #1 on this task. The value is a ratio varying from 0 to infinity, where 1 represents normal or unaffected motion. The MSO changes on the MMI600 screens are more restricted. This override affects only spindle type motion. This does not affect any other type of motion. This value can not be changed if the MSOLock task mode is active. Also, if the AnalogMSOInput task parameter is not -1, the specified analog input determines the MSO value and this parameter becomes read-only and will indicate its value. Additionally, this parameter reflects only the requested spindle feedrate override, if the MaxFeedRateRPM machine parameter have limited the actual feedrate. This will not be reflected in the MSO value. This task parameter is sampled at the rate indicated by the AvgPollTimeSec global parameter.

C.4.84. NormalcyToleranceDeg

This task parameter defines the absolute value of the minimum angular difference between two moves, less than which, no **G9** deceleration will be forced in-between the two moves. In addition, if the angular difference is less that this amount, then the normalcy move will be accomplished in the first UpdateTimeSec period of the second move. Therefore, to avoid "jerking" of the normalcy axis, this value should be kept low. However, if it is too low, the controller will enforce slowdowns to zero velocity, (in order to accomplish a normalcy alignment move) in-between moves, which are almost identically at the same angle. Moves that are at nearly, but, not exactly tangential are commonly output by CNC program post processors.

C.4.85. NormalcyAxis

This task parameter specifies which task axis is used for the normalcy axis. This axis must be defined as a modulo position rotary type (Type = 1) axis. When enabled the normalcy axis will maintain a normal (perpendicular) orientation to the contoured motion path. The axes used to determine the normalcy plane are specified by the NormalcyX and NormalcyY parameters. This parameter is 0 based, i.e.; the 1st axis (X) is represented as 0.

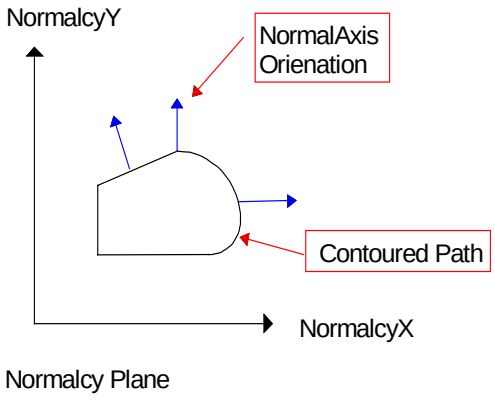


Figure C-18. Normalcy

C.4.86. NormalcyX

This task parameter specifies which task axis is used for the X axis of the normalcy plane. This axis must be a linear type. The NormalcyX and NormalcyY parameters are used to determine the normalcy plane. When enabled the normalcy axis will maintain a normal (perpendicular) orientation to the contoured motion projected onto the normalcy plane. This parameter is 0 based, i.e.; the 1st axis (X) is represented as 0.

C.4.87. NormalcyY

This task parameter specifies which task axis is used for the Y axis of the normalcy plane. This axis must be a linear type. The NormalcyX and NormalcyY parameters are used to determine the normalcy plane. When enabled the normalcy axis will maintain a normal (perpendicular) orientation to the contoured motion projected onto the normalcy plane. This parameter is 0 based, i.e.; the 2nd axis (Y) is represented as 1.

C.4.88. Number

This task parameter specifies the current task number (from 0 through 3). This value can be used by a CNC program to determine on which task it is running. This is a read-only parameter.

C.4.89. NumTaskAxisPts

This task parameter specifies the total number of task axis point variables to allocate for the task. Each task axis point uses 132 bytes of controller memory.

C.4.90. NumTaskDoubles

This task parameter specifies the total number of task double variables to allocate for the task. Each task double allocated uses up 12 bytes of controller memory

C.4.91. NumTaskStrings

This task parameter specifies the total number of task string32 variables to allocate for the task. Each task string32 allocated uses up 34 bytes of controller memory.

C.4.92. ROReq1

The ROReq1 task parameter is a pointer to a 16-bit virtual register output that is used by the UNIDEX 600 MMI to request a desired action from the controller, to be handled by the user. Request words are broken into individual bits, each bit represents a request for an action. The controller will request the appropriate action. It is the user's responsibility to respond to this request and to clear the bit from this register once acted upon. The following table shows the actions that may be requested via this parameter. Setting this parameter to –1, disables this mode. See the RoReq1Mask task parameter for specifying only those requests, which you wish to respond to.

Table C-12. ROReq1 Bit Descriptions

Bit #	Text Description	Program Define	Description
0	Cycle Start	RIO_CYCLESTART	Execute currently associated program.
1	Cycle Step	RIO_CYCLESTEP	Execute a single block of currently associated program.
2	Retrace On	RIO_CYCLERETRACE_ON	Enable retrace mode on the specified task.
3	Retrace Off	RIO_CYCLERETRACE_OFF	Disable retrace mode on the specified task.
4	Cycle Stop	RIO_CYCLESTOP	Stop at end of current executing block.
5	Cycle Reset	RIO_CYCLERESET	Reset the current program (must be active, resets program to line 0).
6	Cycle Abort	RIO_CYCLEABORT	Abort current program – All motion is aborted and program is made active.
7	Async. Motion	RIO_ASYNC_MOVE	Execute RIActionOpCode
8	Joystick Enable	RIO_SLEWSTART	Enable the Joystick
9	Joystick Disable	RIO_SLEWSTOP	Disable the Joystick
10	Unused		
11	Unused		
12	Auto Mode On	RIO_AUTOMODE_ON	Place into auto mode
13	Auto Mode Off	RIO_AUTOMODE_OFF	Place into single step mode
14	Unused		
15	Unused		

C.4.93. RIAction1

The RIAction1 task parameter is a pointer to a 16-bit virtual register input word that may be used by the user to trigger an action to occur on a task. Action words are broken into individual bits, each bit representing a different action. The UNIDEX 600 Series Controller will respond to each action specified in the Action Word and then clear the appropriate bit from the action register. The following table shows the actions that may be triggered via this parameter. This parameter is actually a pointer to an unused virtual input register that will be used to signal the desired actions shown in the chart below. The result of these actions are indicated in the virtual output register pointed to by the ROAction1 task parameter. Setting this parameter to –1, disables this action.

Table C-13. RIAction1 Bit Descriptions

Bit #	Text Description	Program Define	Description
0	Cycle Start	RIO_CYCLESTART	Execute currently associated program.
1	Cycle Step	RIO_CYCLESTEP	Execute a single block of currently associated program.
2	Retrace On	RIO_CYCLERETRACE_ON	Enable retrace mode on the specified task.
3	Retrace Off	RIO_CYCLERETRACE_OFF	Disable retrace mode on the specified task.
4	Cycle Stop	RIO_CYCLESTOP	Stop at end of current executing block.
5	Cycle Reset	RIO_CYCLERESET	Reset the current program (must be active, resets program to line 0).
6	Cycle Abort	RIO_CYCLEABORT	Abort current program – All motion is aborted and program is made active.
7	Async. Motion	RIO_ASYNC_MOVE	Execute RIActionOpCode
8	Joystick Enable	RIO_SLEWSTART	Enable the Joystick
9	Joystick Disable	RIO_SLEWSTOP	Disable the Joystick
10	Unused		
11	Unused		
12	Auto Mode On	RIO_AUTOMODE_ON	Place into auto mode
13	Auto Mode Off	RIO_AUTOMODE_OFF	Place into single step mode
14	Unused		
15	Unused		

For example, to enable and disable the joystick, you would define RIAction1 to point to an unused virtual input register,

```
RIAction1.1 = 127                    ; Assign unused virtual register input
```

You would then activate the joystick by setting the SLEW start bit,

```
$RI[RIAction1.1] = RIO_SLEWSTART     ; enable joystick
```

You would disable the joystick by setting the SLEW stop bit,

```
$RI[RIAction1.1] = RIO_SLEWSTOP      ; disable joystick
```

C.4.94. RIActionAxis

The RIActionAxis task parameter is used as a parameter for the RIActionOpCode task parameter to specify the axis used for the action. This parameter is 0 based, implying that the first axis (X) would be specified as zero.

C.4.95. RIActionParm1

The RIActionParm1 task parameter is used as the first parameter for the RIActionOpCode task parameter.

C.4.96. RIActionParm2

The RIActionParm2 task parameter is used as the second parameter for the RIActionOpCode task parameter.

C.4.97. RIActionOpCode

The RIActionOpCode task parameter is used to command motion on a task from a program running on the same or most likely another task, by setting this parameter (and the RIActionAxis, RIActionParm1 and RIActionParm2 task parameters) to one of the following values.

0 – No action RIACTION_OPCODE_NONE

Spindle OpCodes

1 – Turn Spindle CW (M3, M23, M33, M43) RIACTION_OPCODE_SPINDLE_CW
2 – Turn Spindle CCW (M4, M24, M34, M44) RIACTION_OPCODE_SPINDLE_CCW
3 – Turn Spindle Off (M5, M25, M35, M45) RIACTION_OPCODE_SPINDLE_OFF
4 – Spindle Reorient (M19, M219, M319, M419) RIACTION_OPCODE_SPINDLE_REORIENT
RIActionAxis - Not Used
RIActionParm1 - This is set to the Spindle Index (0-3)
RIActionParm2 - Not Used

Async Motion OpCodes

5 – MoveTo RIACTION_OPCODE_ASYNCTYPE_MOVETO
RIActionAxis - Axis to Move
RIActionParm1 - Target to move to
RIActionParm2 - Velocity to move at
6 – Async. Home RIACTION_OPCODE_ASYNCTYPE_HOME
RIActionAxis - Axis to Home
RIActionParm1 - Not Used
RIActionParm2 - Not Used
7 – Async. Index RIACTION_OPCODE_ASYNCTYPE_INDEX
RIActionAxis - Axis to Move
RIActionParm1 - Distance to move to
RIActionParm2 - Velocity to move at
8 – Freerun RIACTION_OPCODE_ASYNCTYPE_START
RIActionAxis - Axis to Move
RIActionParm1 - Direction to move (1 = +, -1 = -)
RIActionParm2 - Velocity to move at
9 – InFeed RIACTION_OPCODE_ASYNCTYPE_INFEED
RIActionAxis - Axis to Move
RIActionParm1 - Distance to move
RIActionParm2 - Velocity to move at
10 – Queue Index RIACTION_OPCODE_ASYNCTYPE_QINDEX
RIActionAxis - Axis to Move
RIActionParm1 - Distance to move to
RIActionParm2 - Velocity to move at

```
11 – Queue MoveTo                    RIACTION_OPCODE_ASYNCTYPE_QMOVETO
     RIActionAxis     - Axis to Move
     RIActionParm1    - Target to move to
     RIActionParm2    - Velocity to move at
12 – 14 Unused
15 – Oscillate                       RIACTION_OPCODE_ASYNCTYPE_OSCILLATE
     RIActionAxis     - Axis to Move
     RIActionParm1    - Distance to move to
     RIActionParm2    - Velocity to move at
16 – Halt Motion                     RIACTION_OPCODE_ASYNCTYPE_HALT
     RIActionAxis     - Axis to Stop
     RIActionParm1    - Not Used
     RIActionParm2    - Not Used
17 – HandWheel                       RIACTION_OPCODE_HANDWHEEL
     RIActionAxis     - Axis to handwheel
     RIActionParm1    - Channel
     RIActionParm2    - Distance

; Some example subroutines, these can be run from
; any task and will cause motion on Task 1
```

M02

```
DFS SpindleCW
   RIActionOpCode.1  =  RIACTION_OPCODE_SPINDLE_CW
   RIActionParm1.1   =  0                                  ; Spindle 1
   $RI[RIAction1.1]  =  RIO_ASYNC_MOVE
ENDDFS

DFS SpindleCCW
   RIActionOpCode.1  =  RIACTION_OPCODE_SPINDLE_CCW
   RIActionParm1.1   =  0                                  ; Spindle 1
   $RI[RIAction1.1]  =  RIO_ASYNC_MOVE
ENDDFS

DFS SpindleOff
   RIActionOpCode.1  =  RIACTION_OPCODE_SPINDLE_OFF
   RIActionParm1 .1  =  0                                  ; Spindle 1
   $RI[RIAction1.1]  =  RIO_ASYNC_MOVE
ENDDFS

DFS IndexXAxis
   RIActionOpCode.1  =  RIACTION_OPCODE_ASYNCTYPE_INDEX
   RIActionAxis .1   =  0                                  ; Axis Number
   RIActionParm1 .1  =  10                                 ; Distance
   RIActionParm2.1   =  10                                 ; Velocity
   $RI[RIAction1.1]  =  RIO_ASYNC_MOVE
ENDDFS
```

```
DFS HandXAxisOn
   RIActionOpCode.1  =  RIACTION_OPCODE_HANDWHEEL
   RIActionAxis.1    =  0                          ; Axis Number
   RIActionParm1.1   =  4                          ; Channel
   RIActionParm2.1   =  .025                       ; Distance
   $RI[RIAction1.1]  =  RIO_ASYNC_MOVE
ENDDFS

DFS HandXAxisOff
   RIActionOpCode.1  =  RIACTION_OPCODE_HANDWHEEL
   RIActionAxis.1    =  0                          ; Axis Number
   RIActionParm1.1   =  0                          ; Channel
   RIActionParm2.1   =  0                          ; Distance
   $RI[RIAction1.1]  =  RIO_ASYNC_MOVE
ENDDFS
```

C.4.98. ROAction1

The ROAction1 task parameter is a pointer to a 16-bit virtual register output word that will indicate that the action specified by the RIAction1 task parameter has occurred. Action words are broken into individual bits, each bit representing a different action. The UNIDEX 600 Series Controller will respond to each action specified in the RIAction1 register and then clear the appropriate bit from the RIAction1 register, then sets the bit in the output action word, ROAction1, in this case, indicating that the action has occurred. The following table shows the actions that may be indicated via this parameter. Setting this parameter to –1 disables this action.

Table C-14. ROAction1 Bit Descriptions

Bit #	Text Description	Program Define	Description
0	Cycle Start	RIO_CYCLESTART	Execute currently associated program.
1	Cycle Step	RIO_CYCLESTEP	Execute a single block of currently associated program.
2	Retrace On	RIO_CYCLERETRACE_ON	Enable retrace mode on the specified task.
3	Retrace Off	RIO_CYCLERETRACE_OFF	Disable retrace mode on the specified task.
4	Cycle Stop	RIO_CYCLESTOP	Stop at end of current executing block.
5	Cycle Reset	RIO_CYCLERESET	Reset the current program (must be active, resets program to line 0).
6	Cycle Abort	RIO_CYCLEABORT	Abort current program – All motion is aborted and program is made active.
7	Async. Motion	RIO_ASYNC_MOVE	Execute RIActionOpCode
8	Joystick Enable	RIO_SLEWSTART	Enable the Joystick
9	Joystick Disable	RIO_SLEWSTOP	Disable the Joystick
10	Unused		
11	Unused		
12	Auto Mode On	RIO_AUTOMODE_ON	Place into auto mode
13	Auto Mode Off	RIO_AUTOMODE_OFF	Place into single step mode
14	Unused		
15	Unused		

C.4.99. ROReq1Mask

This task parameter is a companion to the ROReq1 task parameter. The default value is 0hFFFF (65535), displayed as –1 in AerDebug. The ROReq1 task parameter requires that the user respond to all possible requests. This parameter allows you to define only those requests which you wish to respond to.

It is recommended that Cycle Start always be included in the ROReq1Mask, this will prevent a user task from executing if (this) supervisory task is not running!

For example, to respond only to Cycle Start and Cycle Stop requests:

ROReq1Mask.1 = RIO_CYCLESTART BOR RIO_CYCLESTEP

The controller will respond to all other requests and the user will be responsible for handling the CycleStart and CycleStop functions.

C.4.100. RotaryFeedRate

This task parameter is the programmed vectorial speed of rotary axes in contoured motion (**G1**, **G2**, **G3**, **G12**, and **G13**). It is normally specified in units of RPM. See the E word documentation, for more details on the use of this parameter. However note, that the units of this parameter will vary by the value of the UserFeedRateMode task parameter (the **G93** /**G94**/ **G95** mode) as shown below:

UserFeedRateMode		**G93** /**G94** /**G95** mode	Units of the F word
0	DEFAULT	**G94**	revolutions / minute
1		**G93**	minutes / revolutions
2		**G95**	revolutions/spindle1-revolution
3		**G29**	5revolutions/spindle2-revolution
4		**G395**	revolutions/spindle3-revolution
5		**G495**	revolutions/spindle4-revolution

The Vectorial velocity of rotary axes in contoured motion is the square root of the sum of the squares of all the speeds of rotary type axes involved in the contoured motion. This parameter value is not signed: its value must always be positive.

Note that the RotaryFeedrate task parameter value has no effect on contoured motion of linear axes (see the LinearFeedRate parameter), nor does it effect motion on rotary axes that are being controlled by the linear axes motion component.

The RotaryFeedrate task parameter value is a programmed value, and will retain its value whether there is currently any contoured motion. However, due to feedrate limiting and the MFO, the actual vectorial velocity during a contoured move may differ from the programmed vectorial velocity. The RotaryFeedRateActual task parameter indicates the actual vectorial velocity.

The CNC "E word " is equivalent to the RotaryFeedRate parameter, and the value of this parameter can be observed from the MMI 600 in the "E" window of the active G code section of the Run or manual screens.

If you change the RotaryFeedRate during a contoured move, this change will not take effect until the next contoured move. Use the MFO to change the speed during a contoured move.

C.4.101. RotaryFeedRateActual

This task parameter indicates the actual vectorial feedrate of rotary axes, in RPM. This is the same as the commanded vectorial feedrate, (task parameter RotaryFeedRateActual) unless the speeds of the axes involved are limited by the MaxFeedRateRPM machine parameter, or the MFO/MSO is less than 100%. Note, that this parameter only denotes the vectorial speed during contoured (**G1**, **G2**, **G3**, etc.) moves and is equal to zero during all other types of motion. This parameter is updated every 10 millisecond's.]

C.4.102. RotateAngleDeg

This task parameter specifies the amount of rotation in degrees to perform in the parts rotation plane specified by the RotateX and RotateY task parameters. A value of zero disables the parts rotation. Any non-zero value adds to the degree of rotation. The rotation is performed around the last position at which this parameter was set. Setting this parameter to 30 and then later, to 20, will cause a combined 50° rotation. Positions relative to the last point of rotation are rotated the specified number of degrees. To disable the parts rotation, set this parameter to –99,999.

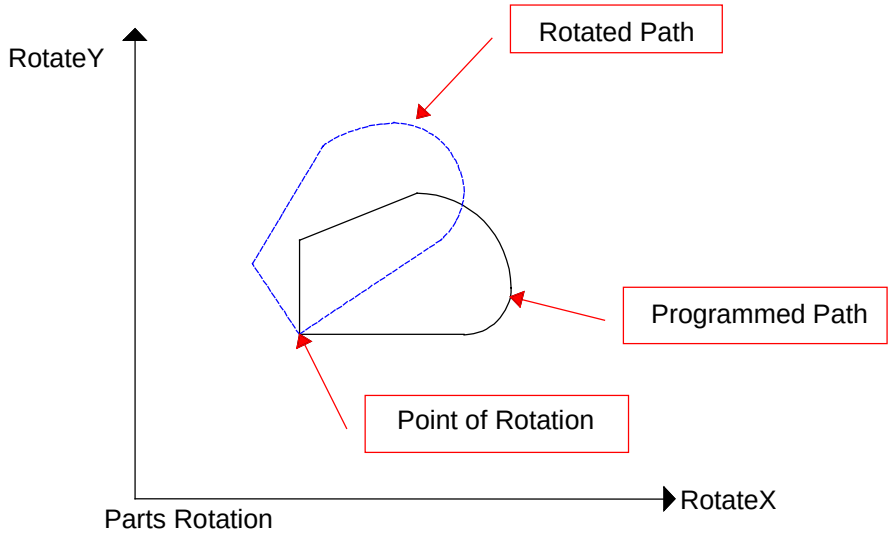


Figure C-19. Part Rotation

C.4.103. RotateX

This task parameter specifies which task axis is used for the X axis of the parts rotation plane. This axis must be a linear type. The RotateX and RotateY parameters are used to determine the parts rotation plane. When enabled the contoured and rapid motion on this plane will be rotated.

C.4.104. RotateY

This task parameter specifies which task axis is used for the Y axis of the parts rotation plane. This axis must be a linear type. The RotateX and RotateY parameters are used to determine the parts rotation plane. When enabled the contoured and rapid motion on this plane will be rotated.

C.4.105. RThetaEnabled

This task parameter enables or disables the R-Theta transformation. The RThetaX and RThetaY parameters are used to determine the perpendicular base X-Y plane. The RThetaR and RThetaT parameters are used to determine the polar R-Theta coordinate system. The RThetaT and RThetaY parameters are used to determine the cylindrical R-Theta coordinate system. The following table shows the possible R-Theta transformations and the enabling values.

Table C-15. R-Theta Transformations

RThetaEnabled Value	Transformation Type	Base Coordinate System	Transformed Coordinate System
0	None (disabled)	n/a	n/a
1	Polar (enabled)	X-Y	R-Theta
2	Cylindrical (enabled)	X-Y	Theta-Y

The polar R-Theta transformation takes the X-Y positions and transforms them into a radius and angular rotation from the origin (where the transformation was enabled).

The cylindrical transformation takes the X position and transforms it into an angular rotation based upon the starting X position (where the transformation was enabled) and the RThetaRadiusInch parameter. For the cylindrical transformation the Y axis position is unchanged.

C.4.106. RThetaR

This task parameter specifies which task axis is used for the R (Radius) axis of the R-Theta transformation. This axis must be a linear type. The RThetaX and RThetaY parameters are used to determine the perpendicular base X-Y plane. The RThetaR and RThetaT parameters are used to determine the polar R-Theta coordinate system. The RThetaT and RThetaY parameters are used to determine the cylindrical R-Theta coordinate system.

C.4.107. RThetaRadius

This task parameter will indicate or may be used to specify the radius used by the R-Theta transformation in the current user units.

C.4.108. RThetaRadiusInch

This task parameter specifies the radius for cylindrical R-Theta transformations. The appropriate angular position is calculated from the cylindrical radius and the X axis position.

C.4.109. RThetaT

This task parameter specifies which task axis is used for the T (Theta) axis of the R-Theta transformation. This axis must be a rotary type. The RThetaX and RThetaY parameters are used to determine the perpendicular base X-Y plane. The RThetaR and RThetaT parameters are used to determine the polar R-Theta coordinate system. The RThetaT and RThetaY parameters are used to determine the cylindrical R-Theta coordinate system.

C.4.110. RThetaX

This task parameter specifies which task axis is used for the X (base) axis of the R-Theta transformation. This axis must be a linear type. The RThetaX and RThetaY parameters are used to determine the perpendicular base X-Y plane. The RThetaR and RThetaT parameters are used to determine the polar R-Theta coordinate system. The RThetaT and RThetaY parameters are used to determine the cylindrical R-Theta coordinate system.

C.4.111. RThetaY

This task parameter specifies which task axis is used for the Y (base) axis of the R-Theta transformation. This axis must be a linear type. The RThetaX and RThetaY parameters are used to determine the perpendicular base X-Y plane. The RThetaR and RThetaT parameters are used to determine the polar R-Theta coordinate system. The RThetaT and RThetaY parameters are used to determine the cylindrical R-Theta coordinate system.

C.4.112. S1_Index

This task parameter specifies which task axis is used for the first spindle axis. This axis must be a rotary type. The numbers are zero based (i.e., S1_Index = 0 indicates the first axis). This parameter may not be changed while the spindle is in motion.

C.4.113. S1_RPM

This task parameter defines the feedrate of spindle number one. By default, the units are revolutions per minute. The G codes listed below will change the units of this parameter.

C.4.114. S1_SpindleRadius

The S1_SpindleRadius task parameter specifies the spindle radius, which is used along with the S1_RPM (the S Word) task parameter to compute the actual spindle RPM. S1_SpindleRadius operates for task 1, use the other parameters for tasks 2, 3 and 4. (e.g., S2_SpindleRadius for task 2). This parameter can equivalently be set with the F parameter of the **G97** command.

If the Spindle Radius is zero, then the units of the S word are assumed to be RPM. However, if the spindle radius is positive, then the S word (S1_RPM for task 1) is assumed to be in distance units, and the actual spindle RPM is computed as

$$RPM = v / (2\pi R)$$

Where RPM is the result spindle rotation speed, v is the surface speed (the S word value), and R is the spindle radius provided in the F parameter of the **G97** command. Note that the units of the S word and the Spindle Radius are assumed to be the same.

C.4.115. S2_AnalogMSOInput

This task parameter specifies which analog input channel is used for the analog MSO for the second spindle, as defined by S2_Index. Otherwise it is exactly the same as the AnalogMSOInput task parameter.

You must have the ExecuteNumSpindles task parameter set properly to utilize more than one spindle.

C.4.116. S2_Index

This task parameter specifies which task axis is used for the second spindle axis. This axis must be a rotary type. The numbers are zero based (i.e., S2_Index = 0 indicates the first axis). This parameter may not be changed while the spindle is in motion.

C.4.117. S2_MSO

This task parameter sets the Manual Spindle feedrate Override for Spindle #2 on this task. The value is a ratio varying from 0 to infinity, where 1 represents normal or unaffected motion. The MSO changes on the MMI600 screens are more restricted. This override affects only spindle type motion. This does not affect any other type of motion. This value can not be changed if the MSOLock task mode is active. Also, if the S2_AnalogMSOInput task parameter is not -1, the specified analog input determines the MSO value and this parameter becomes read-only and will indicate its value. Likewise, the MFO slider bar on the Run and Manual (MDI) screens of the MMI600 can only be used when this parameter is set to –1, other values will enable external control, causing the slider bar to display the set value.

You must have the ExecuteNumSpindles task parameter set properly to utilize more than one spindle.

C.4.118. S2_RPM

This task parameter defines the feedrate of spindle number two. By default, the units are revolutions per minute. The G codes listed below will change the units of this parameter.

C.4.119. S2_SpindleRadius

The S2_SpindleRadius task parameter specifies the spindle radius, which is used along with the S2_RPM (the S Word) task parameter to compute the actual spindle RPM. S3_SpindleRadius operates for task 3, use the other parameters for tasks 1, 2 and 4. (e.g., S1_SpindleRadius for task 1). This parameter can equivalently be set with the F parameter of the **G97** command.

If the Spindle Radius is zero, then the units of the S word are assumed to be RPM. However, if the spindle radius is positive, then the S word (S2_RPM for task 2) is assumed to be in distance units, and the actual spindle RPM is computed as

$$RPM = v / (2\pi R)$$

Where RPM is the result spindle rotation speed, v is the surface speed (the S word value), and R is the spindle radius provided in the F parameter of the **G97** command. Note that the units of the S word and the Spindle Radius are assumed to be the same.

C.4.120. S3_AnalogMSOInput

This task parameter specifies which analog input channel is used for the analog MSO for the third spindle, as defined by S3_Index. Otherwise it is exactly the same as the AnalogMSOInput task parameter.

You must have the ExecuteNumSpindles task parameter set properly to utilize more than one spindle.

C.4.121. S3_Index

This task parameter specifies which task axis is used for the third spindle axis. This axis must be a rotary type. The numbers are zero based (i.e., S3_Index = 0 indicates the first axis). This parameter may not be changed while the spindle is in motion.

C.4.122. S3_MSO

This task parameter sets the Manual Spindle feedrate Override for Spindle #3 on this task. The value is a ratio varying from 0 to infinity, where 1 represents normal or unaffected motion. The MSO changes on the MMI600 screens are more restricted. This override affects only spindle type motion. This does not affect any other type of motion. This value can not be changed if the MSOLock task mode is active. Also, if the S3_AnalogMSOInput task parameter is not -1, the specified analog input determines the MSO value and this parameter becomes read-only and will indicate its value. Likewise, the MFO slider bar on the Run and Manual (MDI) screens of the MMI600 can only be used when this parameter is set to –1, other values will enable external control, causing the slider bar to display the set value.

You must have the ExecuteNumSpindles task parameter set properly to utilize more than one spindle.

C.4.123. S3_RPM

This task parameter defines the feedrate of spindle number three. By default, the units are revolutions per minute. The G codes listed below will change the units of this parameter.

C.4.124. S3_SpindleRadius

The S3_SpindleRadius task parameter specifies the spindle radius, which is used along with the S3_RPM (the S Word) task parameter to compute the actual spindle RPM. S3_SpindleRadius operates for task 3, use the other parameters for tasks 1, 2 and 4. (e.g., S2_SpindleRadius for task 2). This parameter can equivalently be set with the F parameter of the **G97** command.

If the Spindle Radius is zero, then the units of the S word are assumed to be RPM. However, if the spindle radius is positive, then the S word (S3_RPM for task 3) is assumed to be in distance units, and the actual spindle RPM is computed as

$$RPM = v / (2\pi R)$$

Where RPM is the result spindle rotation speed, v is the surface speed (the S word value), and R is the spindle radius provided in the F parameter of the **G97** command. Note that the units of the S word and the Spindle Radius are assumed to be the same.

C.4.125. S4_AnalogMSOInput

This task parameter specifies which analog input channel is used for the analog MSO for the fourth spindle, as defined by S4_Index. Otherwise it is exactly the same as the AnalogMSOInput task parameter.

You must have the ExecuteNumSpindles task parameter set properly to utilize more than one spindle.

C.4.126. S4_Index

This task parameter specifies which task axis is used for the fourth spindle axis. This axis must be a rotary type. The numbers are zero based (i.e., S4_Index = 0 indicates the first axis). This parameter may not be changed while the spindle is in motion.

C.4.127. S4_MSO

This task parameter sets the Manual Spindle feedrate Override for Spindle #4 on this task. The value is a ratio varying from 0 to infinity, where 1 represents normal or unaffected motion. The MSO changes on the MMI600 screens are more restricted. This override affects only spindle type motion. This does not affect any other type of motion. This value can not be changed if the MSOLock task mode is active. Also, if the S4_AnalogMSOInput task parameter is not -1, the specified analog input determines the

MSO value and this parameter becomes read-only and will indicate its value. Likewise, the MFO slider bar on the Run and Manual (MDI) screens of the MMI600 can only be used when this parameter is set to –1, other values will enable external control, causing the slider bar to display the set value.

You must have the ExecuteNumSpindles task parameter set properly to utilize more than one spindle.

C.4.128. S4_RPM

This task parameter defines the feedrate of spindle number four. By default, the units are revolutions per minute. The G codes listed below will change the units of this parameter.

C.4.129. S4_SpindleRadius

The S4_SpindleRadius task parameter specifies the spindle radius, which is used along with the S4_RPM (the S Word) task parameter to compute the actual spindle RPM. S4_SpindleRadius operates for task 4, use the other parameters for tasks 1, 2 and 3. (e.g., S3_SpindleRadius for task 3). This parameter can equivalently be set with the F parameter of the **G97** command.

If the Spindle Radius is zero, then the units of the S word are assumed to be RPM. However, if the spindle radius is positive, then the S word (S4_RPM for task 4) is assumed to be in distance units, and the actual spindle RPM is computed as

$$RPM = v / (2\pi R)$$

Where RPM is the result spindle rotation speed, v is the surface speed (the S word value), and R is the spindle radius provided in the F parameter of the **G97** command. Note that the units of the S word and the Spindle Radius are assumed to be the same.

C.4.130. SlewPair1

The SlewPair1 task parameter defines the 2 task axes that comprise the first axes pair, that the joystick will command motion on. This parameter may not be zero, there must be at least one axis pair defined. SlewPair2 through SlewPair8 may be undefined (zero).

This parameter is set to a value representing the summation of the two task axes numeric values, assigned when the axis is configured within the axis configuration wizard. See Section C.4.138. for an example.

C.4.131. SlewPair2

The SlewPair2 task parameter defines the 2 task axes that comprise the second pair, that the joystick will command motion on. This parameter may be zero, causing the axes defined by SlewPair1 to be the only axes pair.

This parameter is set to a value representing the summation of the two task axes numeric values, assigned when the axis is configured within the axis configuration wizard. See Section C.4.138. for an example.

C.4.132. SlewPair3

The SlewPair3 task parameter defines the 2 task axes that comprise the third axes pair, that the joystick will command motion on. This parameter may be zero, causing the axes defined by SlewPair1 to be selected after SlewPair2.

This parameter is set to a value representing the summation of the two task axes numeric values, assigned when the axis is configured within the axis configuration wizard. See Section C.4.138. for an example.

C.4.133. SlewPair4

The SlewPair4 task parameter defines the 2 task axes that comprise the fourth axes pair, that the joystick will command motion on. This parameter may be zero, causing the axes defined by SlewPair1 to be selected after SlewPair3.

This parameter is set to a value representing the summation of the two task axes numeric values, assigned when the axis is configured within the axis configuration wizard. See Section C.4.138. for an example.

C.4.134. SlewPair5

The SlewPair5 task parameter defines the 2 task axes that comprise the fifth axes pair, that the joystick will command motion on. This parameter may be zero, causing the axes defined by SlewPair1 to be selected after SlewPair4.

This parameter is set to a value representing the summation of the two task axes numeric values, assigned when the axis is configured within the axis configuration wizard. See Section C.4.138. for an example.

C.4.135. SlewPair6

The SlewPair6 task parameter defines the 2 task axes that comprise the sixth axes pair, that the joystick will command motion on. This parameter may be zero, causing the axes defined by SlewPair1 to be selected after SlewPair5.

This parameter is set to a value representing the summation of the two task axes numeric values, assigned when the axis is configured within the axis configuration wizard. See Section C.4.138. for an example.

C.4.136. SlewPair7

The SlewPair7 task parameter defines the 2 task axes that comprise the seventh axes pair, that the joystick will command motion on. This parameter may be zero, causing the axes defined by SlewPair1 to be selected after SlewPair6.

This parameter is set to a value representing the summation of the two task axes numeric values, assigned when the axis is configured within the axis configuration wizard. See Section C.4.138. for an example.

C.4.137. SlewPair8

The SlewPair8 task parameter defines the 2 task axes that comprise the last (eight) axis pairs, that the joystick will command motion on. This parameter may be zero, causing the axes defined by SlewPair1 to be selected after SlewPair7.

This parameter is set to a value representing the summation of the two task axes numeric values, assigned when the axis is configured within the axis configuration wizard. See Section C.4.138. for an example.

C.4.138. SlewPair# Example

For example, to assign axes Y and Z to the SlewPair1 or JogPair1Mask task parameters, you would set it to a value of 6. To see how this is done, click on each of the task axis numbers below to find its numeric value from the chart, and add them together to find the value to set the parameter to:

```
                Task Axis 2  (Y) =      2
                Task Axis 3  (Z) =      4
          +     ____________________________
                Task Parameter   =      6          ; Y, Z axis pair
```

For the SlewPair# task parameters, this would specify that the horizontal axis of the joystick would command task axis 1 (Y) to move and the vertical axis of the joystick would command task axis 2 (Z) to move. For the JogPair#Mask task parameters this would define axis 1 as Y and axis 2 as Z. The lowest numbered task axis will be axis 1 or for the joystick, commanded by the horizontal axis of the joystick.

This order may be inverted by negating the sign of the value entered. For example;

```
                Task Parameter   =      -6         ; Z, Y axis pair
```

would specify that the horizontal axis of the joystick would command task axis 2 (Z) to move and the vertical axis of the joystick would command task axis 1 (Y) to move. This would also assign the Z axis as axis 1 and the Y axis as Axis 2.

C.4.139. Task Modes

The modal state of the 4 tasks on the controller is indicated by the task parameters.

Status1 – CNC program state, FeedHold and Emergency Stop

Status2 – Spindle #1 - #4 and MSO states

Status3 – Modal G code states

Additional information is provided in the axis parameters.

ALT_STATUS – Misc. Information

STATUS – Axis instantaneous state

MOTIONSTATUS – More axis states

SERVOSTATUS – Axis I/O states (Drive, Aux, Limits, Hall inputs, Marker etc.)

C.4.140. Status1

This task parameter indicates various program states of the task when the respective bit is true. Also, be aware that some motion conditions may also be reported in the STATUS, Status2, Status3, MOTIONSTATUS, and SERVOSTATUS parameters. There are pre-defined definitions of the bits in this task parameter within AerStat.Pgm. These definitions take the form of "TASKSTATUS1_xxx" where xxx is the name as listed below. For example;

If (Status1.2 BAND TASKSTATUS1_ProgramExecuting)

could be used to test for a program executing on task 2.

The first 7 bits in the chart below are the most important, they indicate the status of a CNC program on the task.

Table C-16. Status1 Bit Descriptions

Status1 Task Parameter		
Bit #	**Description**	**Hexadecimal Value**
0	Program Associated	0h1
1	Program Active	0h2
2	Program Executing	0h4
3	Immediate Code Executing	0h8
4	Return Motion Executing	0h10
5	Program Aborted	0h20
6	Single Step Into (U600 MMI Single Step Mode)	0h40
7	Single Step Over (through a subroutine, etc.)	0h80
8	Interrupt Fault Pending	0h100
9	Interrupt Callback Pending	0h200
10	Emergency Stop Active	0h400
11	Feed Hold Active	0h800
12	Callback Hold Active	0h1000
13	Callback Responding	0h2000
14	Program Cleanup	0h4000
15	Program Code Cleanup	0h8000
16	OnGosub Command Pending	0h10000
17	Feed Hold Input Latch	0h20000
18	Probe Cycle Active	0h40000
19	Retrace Mode Active	0h80000
20	Insert Link Move	0h100000
21	Interrupt Active	0h200000
22	Slew Active	0h400000
23	Corner Rounding	0h800000
24	ROReq1 Active	0h1000000
25	Canned Function Pending	0h2000000
26	Canned Function Active	0h4000000
27	Canned Function Executing	0h8000000
28	Program Reset	0h10000000

C.4.140.1. CNC Program Active

A Program is "active" anytime after the first cycle start has occurred, and the program has not yet ended. But, a CNC program is "executing" only when it is actually running a CNC program line.

For example a program that is stopped on a M0 command, or is in step mode waiting for the Cycle Start key to advance to the next line, is "active", but is not "executing".

C.4.140.2. CNC Program Executing

CNC program is "executing" only when it is actually running a CNC program line.

For example a program that is stopped on a M0 command, or is in step mode waiting for the Cycle Start key to advance to the next line, is "active", but is not "executing".

C.4.140.3. CNC Program Aborted

A program is "aborted" if the user pressed the Abort key from the UNIDEX 600 MMI, or if a fault stopped the CNC program execution. The "aborted" bit will then stay on until the "executing" bit comes on again. The "aborted" bit will not come on after a CNC program terminates normally.

C.4.141. Status2

This task parameter indicates various program states of the task when the respective bit is true. Also, be aware that some motion conditions may also be reported in the STATUS, Status1, Status3, MOTIONSTATUS, and SERVOSTATUS parameters. There are pre-defined definitions of the bits in this task parameter within AerStat.Pgm. These definitions take the form of "TASKSTATUS2_xxx" where xxx is the name as listed below. For example;

If (Status2.2 BAND TASKSTATUS2_ProgramExecuting)

could be used to test for a program executing on task 2.

Table C-17. Status2 Bit Descriptions

Status2 Task Parameter		
Bit #	**Description**	**Hexadecimal Value**
0	Spindle #1 Active	0h1
1	Spindle #2 Active	0h2
2	Spindle #3 Active	0h4
3	Spindle #4 Active	0h8
4	MSO Change	0h10
5	Spindle FeedHold Active	0h20
6	Asynchronous FeedHold Active	0h40
7	Cutter Enabling	0h80
8	Cutter Disabling	0h100
9	Cutter Offsets Enabling	0h200
10	Cutter Offsets Disabling	0h400

C.4.141.1. Spindle FeedHold Active

FeedHold has been seen by spindle motion (spindle is either stopped or decelerating).

C.4.141.2. Asynchronous FeedHold Active

FeedHold has been seen by asynchronous motion (motion is either stopped or decelerating).

C.4.142. Status3

ICRC, normalcy mode, R Theta transformations, fixture offsets, etc. Also, be aware that some motion conditions may also be reported in the STATUS, Status1, Status2, MOTIONSTATUS, and SERVOSTATUS parameters. There are pre-defined definitions of the bits in this task parameter within AerStat.Pgm. These definitions take the form of "TASKSTATUS3_xxx" where xxx is the name as listed below. For example;

If (Status3.2 BAND TASKSTATUS3_ProgramExecuting)

could be used to test for a program executing on task 2.

Table C-18. Status3 Bit Descriptions

Status3 Task Parameter		
Bit #	**Description**	**Hexadecimal Value**
0	Axes Rotation Active	0h1
1	R Theta Polar Transformation Active	0h2
2	R Theta Cylindrical Transformation Active	0h4
3	Offset Preset Active	0h8
4	Offset Fixture Active	0h10
5	Offset Manual Active	0h20
6	G0 Motion active	0h40
7	G1/G2/G3/G12/G13 Contoured Motion active	0h80
8	Unused	0h100
9	Motion Continuous	0h200
10	MFO Change	0h400
11	MotionFeedHold Active	0h800
12	Positive Cutter Compensation Offset Active	0h1000
13	Cutter Offset Compensation (ICRC) Left Active	0h2000
14	Cutter Offset Compensation (ICRC) Right Active	0h4000
15	Negative Cutter Compensation Offset Active	0h8000
16	Left Normalcy Mode Active	0h10000
17	Right Normalcy Mode Active	0h20000
18	Normalcy Mode Alignment Active	0h40000
19	Program FeedRate in Minutes/Unit	0h80000
20	Program FeedRate in Units/Revolution	0h100000
21	Limit FeedRate Active	0h200000
22	Limit MFO Active	0h400000
23	Coord1Plane1	0h800000
24	Coord1Plane2	0h1000000

Table C-18. Status3 Bit Descriptions (Continued)

Status3 Task Parameter		
Bit #	**Description**	**Hexadecimal Value**
25	Coord1Plane3	0h2000000
26	Coord2Plane1	0h4000000
27	Coord2Plane2	0h8000000
28	Coord2Plane3	0h10000000
29	Motion No Acceleration	0h20000000
30	Mirroring Mode Active	0h40000000

C.4.142.1. Motion FeedHold Active

FeedHold has been seen by a synchronous motion statement (motion is either stopped or decelerating).

C.4.142.2. Motion Continuous Bit

This bit only has meaning while the controller is executing **G0**, **G1**, **G2** or **G3** motion commands.

If this bit is OFF, then the controller will decelerate to zero speed at the end of the current move, and wait until the current motion is done, before declaring the CNC line completed.

If the bit is ON, then the controller will not decelerate, or will decelerate to a non-zero speed, at the end of the current move. Additionally, the controller will not wait until motion is "done", before continuing to the next CNC command. When there is no contoured motion executing (MOTIONSTATUS, PROFILING bit is on) the bit is set to the previous state.

C.4.143. TaskFault

Task faults are indicated by this task parameter when an error in the execution of a CNC program occurs. For example, dividing by zero or trying to set a non-existent parameter from a CNC program causes a task fault. Task Faults stop the CNC program executing on the task, and also stop motion running on the task (see below for details).

There are many conditions that can cause task faults (for application programmers, any error prefixed by "AER960RET_" in the file, \U600\Include\AerCode.H can potentially be returned as a task fault). The user can also trigger a task fault manually, by setting the TaskFault task parameter non-zero, although only certain values (mentioned above in the 'AER960RET_' series constants) will yield a recognizable description.

A special task fault, "physical axis fault" is generated when any axis contained in the HaltTaskOnAxisFault Task parameter encounters an axis fault. This allows the user to stop CNC programs in response to axis faults. By default, all axis faults stop all CNC programs.

When a task fault occurs, the U600 MMI reports an explanation of the task fault in the lower right-hand corner of the Run or Manual page. The user can view the task fault description from AerDebug by typing "TK x", (where x is the task number) and then

"TSKI." A description of the current task fault, if any, appears in the "fault" line of the display lines shown.

Task faults stay active until cleared, by setting the TaskFault task parameter to zero (or pressing the fault acknowledge button from the UNIDEX 600 MMI manual or run page).

Clearing a task fault does not clear any axis faults that may be generated as a result of the task fault (see below). If the user sets this parameter to zero, it clears the task fault, but not any related axis faults that were generated as a result. However, the fault acknowledge button on the UNIDEX 600 MMI clears both task and axis faults.

Task faults have three possible actions: Stopping the CNC program running on the task, generating an axis fault, and generating an interrupt back to the PC. Keep in mind that unlike axis faults, the user cannot set different actions based on the type of task fault. Each of these is discussed separately below.

C.4.143.1. TaskWarning

Task warnings indicate questionable situations that have occurred on the controller. These are situations in which the CNC program can continue without ambiguity, but nevertheless are probably undesirable. An example is the CNC Motion Queue Starvation condition. Unlike task faults, task warnings have no effect on program execution, nor can they generate interrupts, or cause axis faults. They are just messages produced for the user. Task warning messages appear in the same places as the Task Fault messages (in the lower right-hand corner of UNIDEX 600 MMI, or they may be read via the TSKI command under the AerDebug.exe utility). However, in the UNIDEX 600 MMI, task warnings will be highlighted in yellow.

You can deduce which CNC line is causing a Task Warning by temporarily setting the ThrowWarningsAsTaskFaults global parameter to one. This will stop the CNC program on the offending line, when the warning message occurs.

C.4.143.2. Stopping the CNC program in Response to a TaskFault

Any Task Fault always stops the CNC program running on the task. If the task is currently running a program that is executing a contoured motion (**G0**, **G1**, **G2**, **G12** or **G13**), then that motion will halt (decelerate to a stop). However, any asynchronous motion initiated by that program (including spindle motion) will not normally be halted when a task fault occurs. There is one exception to this:

1. Spindle motion halts (decelerates to a stop) if the **G101**, Spindle Shutdown mode is enabled.

C.4.143.3. Generating an Axis Fault in Response to a TaskFault

A task fault may also generate a special axis fault, known as: "Task Fault" under certain circumstances. This is done to allow the programmer to stop, halt or abort axis motion due to a task fault. The behavior is as follows:

1. An ESTOP task fault, (generated by either a global or task ESTOP) will generate an ESTOP axis fault.

2. All other Task faults, except "Physical Axis Fault", generate the "TaskFault" Axis fault.

The exception in item 2 is necessary to prevent "infinite loop" faulting: Since the "physical axis fault" task fault is created from an axis fault, it would not make sense to generate another axis fault in response to it.

For example, suppose axes 2 and 3 are bound to task 1, and task 1 generates a task fault. Also, suppose that axis 2 has its **FAULTMASK** axis parameter set to 0, but axis 3 has its **FAULTMASK** axis parameter set to 0x10000 (the Task Fault bit is set). Then the axis fault: "task fault" is generated for axis 3, but not for axis 2. Furthermore, suppose that axis 3 has its **DISABLEMASK** axis parameter set to 0x10000; then axis 3 will be disabled when a task fault occurs.

The above example should illustrate that the occurrence of an axis fault on an axis does not by itself define any axis action. What happens to an axis due to an axis fault, is defined by the axis mask parameters (**FAULTMASK**, **DISABLEMASK**, etc.) of that axis. Therefore, what happens due to an axis fault generated by a task fault (ESTOP or "TaskFault" as described above) depends on the ESTOP and TaskFault bits (bits 8 and 16 respectively) of the axis mask parameters (**FAULTMASK**, **DISABLEMASK**, etc.)

By default, ESTOP and "TaskFault" axis faults halt all axes that are bound to the task.

C.4.143.4. Generating a PC Interrupt in Response to a TaskFault

A task fault generates an interrupt to the PC, if, and only if, the task is not already faulted and the 'task fault' bit is set in the INTMASK axis parameter. The application programmer can use the AerEventxxx( ) functions to act upon the interrupt, and then query the TaskFault axis parameter to determine the nature of the fault.

C.4.144. UpdateNumEntries

This task parameter controls the number of points in the profile queue generated by the CNC profiler. This parameter applies only to contoured motion (**G1**, **G2**, **G3**, **G12**, **and G13**). Increasing or decreasing this parameter's value has the same effect as increasing or decreasing the UpdateTimeSec parameter value.

C.4.145. UpdateTimeSec

This task parameter specifies the time between calculated profile points generated by the CNC profiler. This parameter is set by the **G62** command. This parameter applies only to contoured motion (**G1**, **G2**, **G3**, **G12**, **and G13**). Every contoured move is broken into profile points, separated by fixed time intervals (see diagram below). The axis position and velocity commands are then determined by splining between these profile points. The profile points are generated once every polling cycle by the CNC. Therefore, the UpdateTimeSec parameter should not be less than the AvgPollTimeSec value. The more complicated the motion (number of axes, transformations, cutter compensation, and normalcy, etc.) the more time that is required to calculate the profile points. By monitoring the AvgPollTimeSec parameter the minimum amount of time required between points can be determined. This parameter should only be set by those having a full understanding of the details of the UNIDEX 600 Controller and its operation. The resolution of the profile time is 0.001 and its value must be greater than zero. This parameter has no effect on camming motion.

This data is then placed into the Motion Queue, which will feed the commands to the servo loop. The motion queue interpolates data between the points, by using a splining algorithm (see Figure C-20).

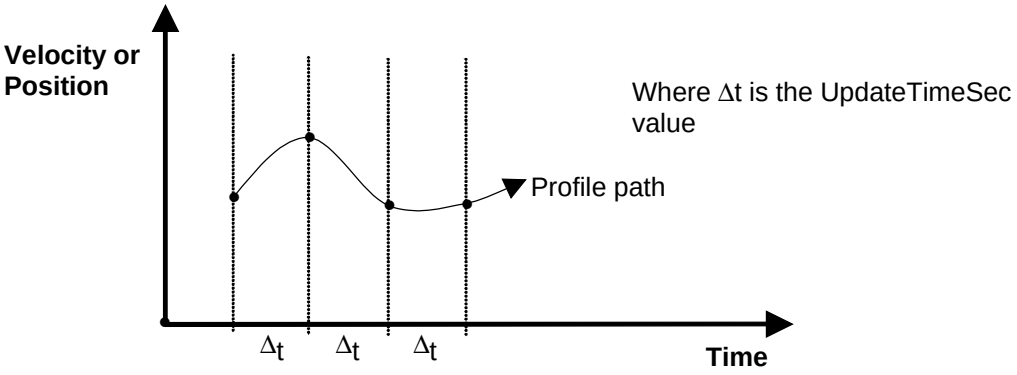


Figure C-20. UpdateTimeSec Diagram

C.4.145.1. Effects of Decreasing the UpdateTimeSec Task Parameter

Decreasing the UpdateTimeSec task parameter will result in more points being calculated within a given time period. However, this is usually not relevant, since the splining algorithm will approximate the programmed path very well with few points. A linear move (**G1**) requires only two points for a perfect fit, and an arc (**G2**, **G3**) requires only three points. All moves will always have at least two points (start and end of move), more points has no effect on **G1** moves. However, for short/fast **G2**, **G3** moves you can end up with only two calculated profile points on an arc. In this case, if the error is unacceptable (see the ChordicalToleranceInch parameter) then you can lower this parameter to prevent two-point arcs. Keep in mind also, that an UpdateTimeSec value that is too low will cause Motion Queue Starvation during contoured moves. But if you do need a low UpdateTimeSec to prevent two point arcs, you can raise the UpdateNumEntries parameter to help prevent starvation. Since the profile points are generated once every polling cycle by the CNC profiler, a minimum UpdateTimeSec parameter can be established as the AvgPollTimeSec value that is observed during the contoured motion.

C.4.145.2. Effects of Increasing the UpdateTimeSec Task Parameter

More typical is the need to increase the UpdateTimeSec task parameter; this may be necessary to prevent Motion Queue Starvation. However, if the starvation occurs as a result of contouring through moves whose duration is shorter than the UpdateTimeSec task parameter, then this will not help eliminate Motion Queue Starvation.

For example, if UpdateTimeSec is 10 (milliseconds), but a particular move lasts only 6 milliseconds, then for that move, the two points generated will be 6 milliseconds apart, not 10 milliseconds. A minor side effect of increasing the value for the UpdateTimeSec, is a potential time delay of "Response Time" (see below) between the generation of motion and its execution. This delay is seen when aborting or changing the speed of a contoured move. (i.e. using the MFO, Jog/Interrupt or feedhold). Once a target position is passed to the motion queue, that motion will occur as calculated, regardless of changes to the MFO,

feedhold, or aborting the move. The requested action will not take place until the next calculation. Therefore the controller will, in general, not respond to the MFO, feedhold, interrupt, or abort a move any faster than:

Response Time = UpdateNumEntries * UpdateTimeSec

This comment applies also to the end of the program, i.e. when the program is ended, the motion may continue for up to "Response Time" seconds after the actual end is seen.

C.4.146. UserFeedRateMode

This task parameter determines the interpretation of the specified linear and rotary feedrates (the F keyword, or LinearFeedRate and the E keyword, or RotaryFeedRate) as shown below (It is equivalent to using the **G93**/**G94**/**G95** codes):

UserFeedRateMode	**G93** /**G94** /**G95** mode	Units of the F word
0	**G94**	user-units / minute
1	**G93**	minutes /user-unit
2	**G95**	user-units / spindle1-revolution
3	**G29**	5user-units / spindle2-revolution
4	**G395**	user-units / spindle3-revolution
5	**G495**	user-units / spindle4-revolution

Note that the value of this parameter will change the units of the LinearFeedRate task parameter (the F keyword) and the units of the RotaryFeedRate task parameter (the E keyword).

C.5. Global Parameters

Global parameters are only used by the CNC interface. Unless CNC motion is used these parameters can be ignored. These values are used to specify information relevant to all axes and all tasks. The case of the global parameters is significant, as defined in the table below.

Table C-19. Global Parameters

Name	Parameter #	Access	Minimum	Maximum	Default
AvgPollTimeSec	0	RU	<NA>	<NA>	0.000429688
BuildNumber	9	R	<NA>	<NA>	23.0
CallBackTimeoutSec	6	RW	0.0	60.0	1.0
CompatibilityMode	15	RW	0	1	0
Enable1KhzServo	8	RW	0	1	0
Enable2DCalibration	13	RW	01	1	0
EStopEnabled	5	RW	0	1	0
Interrupt2TimeSec	7	RW	0.0	60.0	0.0
MeasurementMode	17	RWU	0		0
NumCannedFunctions	14	RW	0	100	10
NumDecimalsCompare	16	RW	0	14	0
NumGlobalAxisPts	4	RW	0	1,000	10
NumGlobalDoubles	2	RW	0	1,000	10
NumGlobalStrings	3	RW	0	1,000	10
ThrowTaskWarningsAsFaults	11	RW	-1	1	0
UserMode	10	RW	-1	7	-1
Version	1	R	<NA>	<NA>	5.11

Task, Machine, and Global parameters apply only to CNC directed motion.

C.5.1. AvgPollTimeSec

This Global parameter indicates the average amount of time in seconds it takes for the controller to complete one polling cycle. Each polling cycle the controller can respond to one library command per axis. Each task is updated every cycle where no more than one CNC command can be executed per cycle. However, some CNC commands (like a **G1**) can take multiple cycles to execute. Therefore, this time represents the average minimum amount of time required for the controller to respond to a library call. This is a read-only parameter and is updated continuously by the controller. All actions not executed off of

the internal interrupt are executed within a polling cycle, sometimes called "the forever loop". A summary of its operation is shown below. The most important functions of the polling loop are to run CNC programs and respond to library functions.

```
DO FOREVER
    Get new I/O values
    Check watchdog
    Check Pendent
    FOR EACH AXIS (16)
        Look for and perform a library call command
    END FOR
    FOR EACH TASK (4)
        Throw interrupt back to PC (if needed)
        Test ESTOP, MFO, MSO, feedhold bits and react to them if necessary
        Monitor spindle and async. motion (if any)
        Test ONGOSUBS and ON monitors, do the indicated action if needed
        Execute immediate command (if any requested)
        Execute the next CNC statement (if that CNC is running a program), or if
        statement was still running (like a G1, or G4) then see if its finished.
    END FOR
ENDDO
```

The AvgPollTimeSec parameter simply represents the time it takes to execute one pass of the "DO FOREVER" loop shown above. The "DO FOREVER" loop runs continually, but is secondary in priority to actions executed off of the Internal Interrupt (mainly servo loop closure). Therefore, when servo loop operation is taking up a lot of time, the Polling Cycle will take correspondingly longer to complete, and the AvgPollTimeSec parameter will be higher. Use this parameter to monitor how much the servo loop motion is utilizing available time. You can view this parameter from the AerDebug.exe utility (after loading the firmware and closing the UNIDEX 600 MMI, AerStat.exe and any other UNIDEX 600 applications) to get an idea of the baseline, or fastest that the polling loop is able to execute. A value of .005 seconds or greater is indicative of an over-loaded controller.

C.5.2. BuildNumber

This Global parameter indicates the version number of the UNIDEX 600 software. This number is a monotonically increasing number, indicating the index of the software build. This number also shows up as the last digit in the version string, (see the title bar on UNIDEX 600 MMI). This number does not necessarily indicate compatibility, (i.e., BuildNumber 20 files may or may not be compatible with BuildNumber 19 files). Compatibility is represented by the version numbers, which are the first numbers in the version string.

C.5.3. CallBackTimeoutSec

This Global parameter specifies the amount of time in seconds that the controller will wait before generating a task fault if there is no response to a callback from the Frontend.

C.5.4. CompatibilityMode

This global parameter, allows the programmer to change the behavior of the UNIDEX 600 back to the "original" or legacy behavior, when such is required. This parameter is provided to allow users to run CNC programs written for older versions of the UNIDEX 600 MMI.

The parameter is a bit-mapped value, where each bit toggles the behavior from the current, or default behavior, to a previous behavior that was defined in older versions. Bit #0 is the least significant bit. The version number in the chart indicates the first version of the software that had this change in behavior.

Table C-20. Compatibility Chart

Bit #	Hexadecimal Value	Data Changed	Version	Old Behavior that is Activated
0	0x1	1/15/99	5.90	**G2** / **G3** are not modal
1	0x2	5/6/99	6.98	Old Style Contouring
2	0x4	8/1/99	6.103	Handling of "within tolerance" radius errors by a linear jerk to start point, using the radius specified from the endpoint.
3	0x8	8/1/99	6.103	Non-averaging end-of-move non-integer time portions, into proceeding move slices.
4	0x10	9/1/99	6.106	Convert **G43**/**G47**/**G65**/**G66** to User Units
5	0x20			Reserved
6	0x40	12/21/99	6.112	Old style Link Moves for Normalcy Motion

C.5.4.1. Move Calculation Averaging

Setting this bit to 1, True, defines non-averaging end-of-move non-integer time portions, into proceeding move slices. Otherwise, the fractional move distance is averaged over the entire move.

C.5.4.2. Radius Error Bit 2

Handling of "within tolerance" radius errors by a linear jerk to start point, using the radius specified from the endpoint. Setting this bit to one restores this mode, otherwise, a new center point is calculated, for errors less than the MaxRadiusError task parameter.

C.5.4.3. Convert G43/G47/G65/G66 to User Units

Software versions prior to 6.106, assumed that that the distance units for the parameters to the **G43** / **G47** / **G65** / **G66** commands were in inches. Setting this bit to 0, defines this state. Otherwise, if the bit is 1, the parameters are assumed to be in the state defined by the **G70** / **G71** modal commands.

This bit will default to one, True, on new software installations. On updates of older systems this bit will be remain zero, to maintain compatibility.

C.5.4.4. Non Modal G2 / G3 Commands

The example CNC program below illustrates the modal/non-modal nature of the **G2** / **G3** commands.

```
G91 G1
G2 I1                ; Generate circular motion
X1                   ; Current versions generate the error: "No Offsets in G2/G3
CompatibiltyMode = CompatibiltyMode BOR 0x1        ; set old style active
G2 I1                ; Generate circular motion
X1                   ; Old style executes this CNC block as a G1 command
```

C.5.4.5. Old Style Contouring

The old style contouring mode refers to small changes made in the manner in which G1, G2 and G3 contours are generated. Both styles are virtually identical. The old style is maintained only for compatibility with internal Aerotech testing. The user should never use the old style. Old style is:

1. "Slice lookahead" during decel phase changed from 3 to 2
2. "Slice lookahead" in constant phase suppressed for arcs

C.5.5. Enable1KHzServo

This Global parameter defines the servo-loop update rate for all axes, as either 1 kHz or 4 kHz. Setting this parameter to 1, activates the 1 kHz mode (1 millisecond) while a 0 activates the 4 kHz mode(4 millisecond). This setting reduces the AvgPollTimeSec. This setting also has no effect on activities run off the interrupt unrelated to the servo loop. This mode may also be toggled by the **G130** / **G131** G codes.

Normally, a 4KHz servo-loop update rate is desirable, to give the fastest possible rate of position error correction, resulting in the highest degree of accuracy to the commanded motion. This is almost a necessity for systems with one or more axes having a system resolution of 1 micron or less.

However, if the controller is over-loaded and having difficulty in completing tasks other than closing the servo loop, then a 1KHz servo-loop interrupt should be considered. This will be obvious if "CNC Profile Queue Starvation" error message is displayed or motion is discontinuous. The trade-off is that a 1Khz loop will result in a lower rate of servo loop update. The AvgPollTimeSec task parameter is an excellent indicator of how much time is left for operations other than the servo loop.

Conditions that may indicate difficulty in completing tasks other than closing the servo loop (indicating the need for a 1KHz interrupt) include:

1. Motion Queue Starvation
2. Motion is discontinuous.
3. "Locking up" or slow response of the UNIDEX 600 MMI during motion.

Another use for the 1KHz setting is to reduce noise in low-resolution encoder systems. When the servo-loop is updated at 4 kHz, and the feedback is of low resolution (greater than 1 micron step size), the resulting corrections may contain a audible high frequency oscillation. Selecting the 1 kHz update rate slows done the rate of correction, thus removing this oscillation.

C.5.6. Enable2Dcalibration

This global parameter enables the 2D axis calibration file, specified on the setup page of the UNIDEX 600 MMI. Setting this parameter to 1 enables 2D Compensation, 0 disables 2D Compensation.

C.5.7. EStopEnabled

This Global parameter specifies whether or not the opto-isolated hardware E-Stop input should be used. A 1 indicates the E-Stop input will be used. A 0 will cause the software to ignore the hardware E-Stop input. Refer to the UNIDEX 600 Series Hardware Manual, P/N EDU154 for more details. This input is active low.

Be sure to set the ESTOP bit in the **FAULTMASK** axis parameter to enable the detection of this fault, then set the bit in the appropriate mask parameter (**DISABLEMASK**, **HALTMASK**, **AUXMASK**, **ABORTMASK**, **INTMASK** and **BRAKEMASK**) for the action to occur on this fault.

C.5.8. Interrupt2TimeSec

This Global parameter specifies the interrupt frequency of the user definable interrupt. The valid range is from 0 - 60 seconds.

C.5.9. Measurement Mode

Setting this global parameter to 1 or 2 activates the measurement mode. In this mode, no motion occurs, but, the controller gathers statistical data on the motion. After you execute statements in measurement mode, you can execute the PRGM command in the AerDebug utility, to examine a detailed analysis of the motion, including the path length, maximum speed, and total elapsed time of the motion (assuming it was not in measurement mode). Note that in measurement mode, the program will execute at very high speed, since no dwells or motion is executed, as defined below.

- **G0**, **G1**, **G2**/**G3**, G4, **G12**/**G13**, **M3**, **M4**, **M5** not executed, just collects statistics on the move
- Binary output ($BO) and register output ($RO) commands are not executed.
- WAIT statements are always declared true.
- Asynchronous motion (jogging) is ignored, and not measured.

You may set the MeasurementMode parameter to 0 at any time, to disable the measurement mode.

C.5.10. NumCannedFunctions

This Global Parameter defines the maximum number of canned functions allowed. Ten is the default.

C.5.11. NumDecimalsCompare

This global parameter is used by the floating point operators to determine the number of fractional decimal places to use in the operation.

C.5.12. NumGlobalAxisPts

This Global parameter specifies the total number of global axis point variables available. Each global axis point allocated uses 132 bytes of controller memory.

C.5.13. NumGlobalDoubles

This Global parameter specifies the total number of global double variables available. Each global double allocated uses 12 bytes of controller memory.

C.5.14. NumGlobalStrings

This Global parameter specifies the total number of global string variables available. Each global string allocated uses 34 bytes of controller memory.

This Global parameter specifies the number of tasks that will be serviced by the controller. The default is 4.

Set this task parameter to one to detect a Motion Queue Starvation Condition. If its non-zero and motion queue starvation occurs, a descriptive task fault will be generated and motion will stop.

C.5.15. ThrowTaskWarningsAsFaults

If this global parameter is positive, the controller will handle task warnings as task faults. If it is zero (the default), task warnings behave as described by the TaskWarning task parameter. If it is negative, task warnings will be ignored.

C.5.16. UserMode

This Global parameter specifies the active page of the UNIDEX 600 MMI, if it is running. If the Manual (MDI) or Jog Page is entered via the Run Page, the Run Page will still be indicated as active.

Table C-21. U600 UserMode Meanings

Value	Meaning
2	Manual or Jog Page is active
1	Run Page is active
0	None of the above

C.5.17. Version

This parameter specifies the current version of (major.minor) the software running on the axis processor. This is a read only parameter.

∇ ∇ ∇

APPENDIX D: WARRANTY AND FIELD SERVICE

In This Section:

Aerotech, Inc. warrants its products to be free from defects caused by faulty materials or poor workmanship for a minimum period of one year from date of shipment from Aerotech. Aerotech's liability is limited to replacing, repairing or issuing credit, at its option, for any products which are returned by the original purchaser during the warranty period. Aerotech makes no warranty that its products are fit for the use or purpose to which they may be put by the buyer, where or not such use or purpose has been disclosed to Aerotech in specifications or drawings previously or subsequently provided, or whether or not Aerotech's products are specifically designed and/or manufactured for buyer's use or purpose. Aerotech's liability or any claim for loss or damage arising out of the sale, resale or use of any of its products shall in no event exceed the selling price of the unit.

Laser Products

Aerotech, Inc. warrants its laser products to the original purchaser for a minimum period of one year from date of shipment. This warranty covers defects in workmanship and material and is voided for all laser power supplies, plasma tubes and laser systems subject to electrical or physical abuse, tampering (such as opening the housing or removal of the serial tag) or improper operation as determined by Aerotech. This warranty is also voided for failure to comply with Aerotech's return procedures.

Return Procedure

Claims for shipment damage (evident or concealed) must be filed with the carrier by the buyer. Aerotech must be notified within (30) days of shipment of incorrect materials. No product may be returned, whether in warranty or out of warranty, without first obtaining approval from Aerotech. No credit will be given nor repairs made for products returned without such approval. Any returned product(s) must be accompanied by a return authorization number. The return authorization number may be obtained by calling an Aerotech service center. Products must be returned, prepaid, to an Aerotech service center (no C.O.D. or Collect Freight accepted). The status of any product returned later than (30) days after the issuance of a return authorization number will be subject to review.

Returned Product Warranty Determination

After Aerotech's examination, warranty or out-of-warranty status will be determined. If upon Aerotech's examination a warranted defect exists, then the product(s) will be repaired at no charge and shipped, prepaid, back to the buyer. If the buyer desires an air freight return, the product(s) will be shipped collect. Warranty repairs do not extend the original warranty period.

Returned Product Non-warranty Determination

After Aerotech's examination, the buyer shall be notified of the repair cost. At such time the buyer must issue a valid purchase order to cover the cost of the repair and freight, or authorize the product(s) to be shipped back as is, at the buyer's expense. Failure to obtain a purchase order number or approval within (30) days of notification will result in the product(s) being returned as is, at the buyer's expense. Repair work is warranted for (90) days from date of shipment. Replacement components are warranted for one year from date of shipment.

Rush Service

At times, the buyer may desire to expedite a repair. Regardless of warranty or out-of-warranty status, the buyer must issue a valid purchase order to cover the added rush service cost. Rush service is subject to Aerotech's approval.

On-site Warranty Repair

If an Aerotech product cannot be made functional by telephone assistance or by sending and having the customer install replacement parts, and cannot be returned to the Aerotech service center for repair, and if Aerotech determines the problem could be warranty-related, then the following policy applies:

Aerotech will provide an on-site field service representative in a reasonable amount of time, provided that the customer issues a valid purchase order to Aerotech covering all transportation and subsistence costs. For warranty field repairs, the customer will not be charged for the cost of labor and material. If service is rendered at times other than normal work periods, then special service rates apply.

If during the on-site repair it is determined the problem is not warranty related, then the terms and conditions stated in the following "On-Site Non-Warranty Repair" section apply.

On-site Non-warranty Repair

If any Aerotech product cannot be made functional by telephone assistance or purchased replacement parts, and cannot be returned to the Aerotech service center for repair, then the following field service policy applies:

Aerotech will provide an on-site field service representative in a reasonable amount of time, provided that the customer issues a valid purchase order to Aerotech covering all transportation and subsistence costs and the prevailing labor cost, including travel time, necessary to complete the repair.

Company Address

Aerotech, Inc.
101 Zeta Drive
Pittsburgh, PA 15238-2897
USA

Phone: (412) 963-7470
Fax: (412) 963-7459

∇ ∇ ∇

Symbol

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

Z

READER'S COMMENTS

U600 User's Guide
P/N EDU 157, December 2000

Please answer the questions below and add any suggestions for improving this document. Is the information:

	Yes	No
Adequate to the subject?	____	____
Well organized?	____	____
Clearly presented?	____	____
Well illustrated?	____	____
Would you like to see more illustrations?	____	____
Would you like to see more text?	____	____

How do you use this document in your job? Does it meet your needs?
What improvements, if any, would you like to see? Please be specific or cite examples.

_____

_____

_____

_____

_____

_____

Your name _____
Your title _____
Company name _____
Address _____
_____

Remove this page from the document and fax or mail your comments to the technical writing department of Aerotech.

AEROTECH, INC.
Technical Writing Department
101 Zeta Drive
Pittsburgh, PA. 15238-2897 U.S.A.
Fax number (412) 967-6870

AEROTECH®