
**THE UNIDEX® 600 SERIES CNC
PROGRAMMING, WIN NT/95
MANUAL**

P/N: EDU158 (V1.1)



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

The UNIDEX 600 is a product of Aerotech, Inc.
Windows 95 and Windows NT are registered trademarks of Microsoft Corporation.
Visual Parse++ is a registered trademark of Sandstone

The UNIDEX 600 Series CNC Programming, Win NT/95 Manual Revision History:

Rev 1.0	July, 1997
Rev 1.1	June 6, 2000

TABLE OF CONTENTS

CHAPTER 1: Introduction and Overview	1-1
1.1. Introduction	1-1
1.2. Purpose of This Manual.....	1-1
1.3. Prerequisites	1-2
1.4. Related Documentation	1-2
1.4.1. Hardware Manuals	1-2
1.4.2. Programming Manuals	1-3
1.4.3. MMI Interface Manual	1-3
1.5. CNC Design Philosophy.....	1-3
1.6. CNC Functionality Summary.....	1-4
1.7. Syntactic Descriptions	1-4
1.7.1. Elements of the Syntactic Description.....	1-5
1.7.2. Keyword Case Sensitivity	1-6
1.7.3. White space	1-6
1.7.4. Syntax Description Examples.....	1-7
1.7.5. Limitations and Bounds.....	1-7
CHAPTER 2: Commands	2-1
2.1. Description	2-1
2.1.1. Command Sets.....	2-1
2.1.2. Conformance to Standards	2-1
2.2. Programs.....	2-2
2.2.1. White space	2-2
2.2.2. Line Terminators.....	2-2
2.2.3. Characters.....	2-2
2.2.4. Comments.....	2-3
2.2.5. Lines.....	2-3
2.2.6. Block Delete.....	2-3
CHAPTER 3: Expressions	3-1
3.1. Description	3-1
3.2. Expressions.....	3-1
3.2.1. Expression Elements	3-1
3.2.2. Expression Types	3-2
3.2.3. Expression Components	3-2
3.2.4. Expression Examples	3-2
3.3. CNC and Axis Letters.....	3-3
3.4. CNC Masks (Axis Masks)	3-4
3.5. CNC Words	3-4
3.6. CNC Block Expressions	3-5
3.6.1. CNC Block Constants	3-5
3.6.1.1. CNC G-code Blocks.....	3-6
3.6.1.2. Axis Points	3-6
3.6.1.3. Argument Lists.....	3-6
3.6.2. APT Variables.....	3-7
3.7. Floating-Point Expressions	3-7
3.7.1. Floating-Point Constants	3-7
3.7.2. Floating Point Variables.....	3-8
3.7.3. Floating Point Operators	3-8

3.7.4.	Floating Point Functions.....	3-9
3.7.5.	Floating Point Computation Precedence	3-10
3.8.	Integer Expressions.....	3-12
3.8.1.	Integer Constants.....	3-12
3.8.1.1.	Hexadecimal Numbers	3-12
3.8.2.	Integer Operators.....	3-13
3.9.	String 32 Expressions	3-14
3.9.1.	String32 Constants	3-15
3.9.2.	String32 Variables.....	3-15
3.9.3.	String 32 Operators	3-15
3.10.	Labels	3-15
3.11.	Variants.....	3-16
3.11.1.	Variant Types	3-16
3.11.2.	Variant Names	3-16
3.11.3.	Assignments to Variants	3-17
3.11.4.	Variables	3-18
3.11.4.1.	Global Variables.....	3-18
3.11.4.2.	Task Variables.....	3-20
3.11.4.3.	Program Variables.....	3-20
3.11.4.4.	Program Array Variables.....	3-21
3.11.5.	Parameters	3-22
3.11.5.1.	Aliases	3-22
3.11.5.2.	Global Parameters	3-23
3.11.5.3.	Task Parameters	3-24
3.11.5.4.	Axis Parameters.....	3-25
3.11.5.5.	Machine Parameters	3-26
3.11.5.6.	Modifying Parameters from within a CNC Program	3-26
3.11.6.	Virtual I/O	3-28
3.11.6.1.	Binary I/O Bits	3-28
3.11.6.2.	Virtual I/O Registers	3-29
3.11.6.3.	Analog Inputs	3-29
3.11.7.	Call Arguments.....	3-30
3.11.7.1.	Call Argument Existence Testing	3-31
CHAPTER 4: Compiler Directive Commands.....		4-1
4.1.	Overview	4-1
4.1.1.	Compiler Directives Syntax.....	4-2
4.2.	Define Statements	4-2
4.2.1.	The Target Word	4-3
4.2.2.	Recognition of the Target Word.....	4-3
4.2.3.	The Replacement String	4-4
4.2.4.	Replacement with Multiple Lines.....	4-5
4.2.5.	Replacement within Replacement Strings	4-5
4.3.	Include Statement	4-6
4.3.1.	Filenames	4-6
4.3.2.	Standard Include Files	4-6
4.4.	AxisNames Statement.....	4-7

CHAPTER 5: G-code Commands	5-1
5.1. Introduction	5-2
5.1.1. Motion Types Available	5-2
5.1.2. Motion Commands Available	5-3
5.1.3. Prerequisites for Initiating Motion from the CNC	5-4
5.1.4. Command .vs. Actual	5-4
5.1.5. Target Positions	5-5
5.1.6. Simultaneous Movement of Multiple Axes	5-5
5.1.7. Velocity	5-6
5.1.8. Acceleration/Deceleration	5-7
5.1.9. Further Information	5-7
5.1.10. Modal	5-7
5.1.11. Default	5-8
5.2. CNC Block Syntax	5-13
5.2.1. CNC Blocks	5-13
5.2.2. N Words	5-14
5.2.3. Motion Blocks	5-14
5.2.3.1. Simple Mode Words	5-15
5.2.3.2. F, E and S Codes (Rate Words)	5-16
5.2.3.3. Motion Modifier Words	5-21
5.2.3.4. Motion Type Words	5-21
5.2.3.5. Offset Words	5-21
5.2.4. Stand-Alone Blocks	5-21
5.2.5. Parameter Setting Blocks	5-22
5.2.5.1. F-code Parameter Blocks	5-22
5.2.5.2. Mask Parameter Blocks	5-22
5.2.5.3. Point Parameter Blocks	5-22
5.3. Non-Contoured Motion (G0)	5-23
5.3.1. Point-to-Point Positioning at a Rapid Feedrate (Motion) G0	5-23
5.4. Contoured Motion (G1, G2, G3)	5-24
5.4.1. Linear Interpolation (Motion) G1	5-24
5.4.2. Circular Interpolation CW on Coordinate System #1 (Motion) G2	5-25
5.4.3. Circular Interpolation CCW on Plane #1 (Motion) G3	5-34
5.5. Dwell (G4)	5-35
5.5.1. Dwell G4	5-35
5.5.2. Asynchronous Dwells	5-35
5.6. Velocity Blending (G8, G9, G108, G109)	5-36
5.6.1. Instantaneous Acceleration G8	5-39
5.6.2. Force Deceleration G9	5-40
5.7. Contoured Motion on Coordinate System # 2 (G12, G13)	5-41
5.7.1. Circular Interpolation CW on Coordinate System #2 (Motion) G12	5-41
5.7.2. Circular Interpolation CCW on Coordinate System #2 G13	5-42
5.8. Coordinate System #1 Configuration (G16 – G19)	5-43
5.8.1. Assign Coordinate System #1 Axes G16	5-43
5.8.2. Plane Selection Codes Set # 1 G17/G18/G19	5-44
5.9. Normalcy Motion Overview (G20, G21, G22)	5-45
5.9.1. Disable Normalcy Mode G20	5-48

5.9.2.	Activate Normalcy Mode Left G21	5-48
5.9.3.	Activate Normalcy Mode Right G22	5-49
5.10.	Corner Rounding (G23, G24) G23	5-50
5.10.1.	Disable Corner Rounding Mode G24	5-50
5.11.	Coordinate System #2 Configuration (G26 – G29)	5-51
5.11.1.	Assign Coordinate System #2 Axes G26	5-51
5.11.2.	Plane Selection Codes for Coordinate System #2 G27/G28/G29	5-51
5.12.	Software Limits Overview	5-52
5.12.1.	Configuring Software Limits	5-52
5.13.	Safe Zones (G34, G35, G36, G37)	5-52
5.13.1.	Set Safe Zone Minimum Values G34	5-54
5.13.2.	Set Safe Zone Maximum Values G35	5-54
5.13.3.	Enable Safe Zones G36	5-54
5.13.4.	Disable Safe Zones G37	5-55
5.13.5.	Safe Zone Activation	5-56
5.13.6.	Configuring and Using Safe Zones	5-56
5.14.	Backlash Compensation (G38, G39)	5-57
5.14.1.	Enable Backlash Compensation G38	5-57
5.14.2.	Disable Backlash Compensation G39	5-57
5.15.	Cutter Radius Compensation (G40, G41, G42, G43, G45)	5-58
5.15.1.	CNC Block Look-Ahead Requirements in Cutter Compensation Mode	5-59
5.15.2.	Cutter Radius Compensation Lead-On and Lead-Off Moves	5-60
5.15.3.	Interaction of Mirroring and Cutter Compensation Commands	5-60
5.15.4.	Cutter Compensation Limitations within Inside Corners	5-60
5.15.5.	Cutter Compensation within Outside Corners	5-62
5.15.6.	Deactivate Cutter Compensation (ICRC) G40	5-63
5.15.7.	Activate ICRC Left G41	5-64
5.15.8.	Activate ICRC Right G42	5-65
5.15.9.	Set Cutter Compensation Radius G43	5-66
5.15.10.	Set Cutter Compensation Axes G44	5-67
5.16.	Polar/Cylindrical Transformations (G45, G46, G47)	5-68
5.16.1.	Disable Polar or Cylindrical Coordinate Transformation G45	5-68
5.16.2.	Enable Polar Coordinate Transformation G46	5-68
5.16.3.	Enable Cylindrical Coordinate Transformation G47	5-71
5.16.4.	Monitor Touch Probe G51	5-73
5.16.5.	Define Polar/Cylindrical Transformation Axes G52	5-73
5.17.	Fixture Offsets (G53 – G59)	5-74
5.17.1.	Cancel Fixture Offset G53	5-74
5.17.2.	Set Fixture Offset #1 G54	5-74
5.17.3.	Set Fixture Offset #2 G55	5-76
5.17.4.	Set Fixture Offset #3 G56	5-76
5.17.5.	Set Fixture Offset #4 G57	5-77
5.17.6.	Set Fixture Offset #5 G58	5-78
5.17.7.	Set Fixture Offset #6 G59	5-79
5.18.	Contoured Accel/Decel Overview (G60, G61)	5-81

5.18.1.	Explicit Feedrates and Automatic Acceleration	5-82
5.18.2.	Set Acceleration Time G60	5-83
5.18.3.	Set Deceleration Time G61	5-83
5.19.	Profile Resolution Time (G62)	5-84
5.19.1.	Set Profile Time G62	5-84
5.20.	Accel/Decel Rates and Modes (G63 -> G68)	5-84
5.20.1.	Sinusoidal (1-Cosine) Accel/Decel Mode G63	5-84
5.20.2.	Linear Accel/Decel Mode G64.....	5-86
5.20.3.	Set Acceleration Rate (for linear type axes) G65	5-86
5.20.4.	Set Deceleration Rate (for linear type axes) G66	5-87
5.20.5.	Time Based Acceleration/Deceleration G67	5-87
5.20.6.	Rate Based Acceleration/Deceleration G68	5-88
5.21.	Metric/English Units (G70, G71)	5-89
5.21.1.	Inch Dimension Programming Mode (Units) G70	5-89
5.21.2.	Metric Dimension Programming Mode (Units) G71.....	5-90
5.22.	Restore Preset Position Registers G82	5-90
5.23.	Transformation Overview (G83, G84)	5-91
5.23.1.	Mirror Image G83	5-91
5.23.2.	Parts Rotation G84	5-93
5.24.	Positioning Modes (G90, G91).....	5-95
5.24.1.	Absolute Dimension Programming Mode (Distance) G90.....	5-95
5.24.2.	Incremental Position Programming (Distance) G91.....	5-96
5.25.	Preset Positions (G92)	5-97
5.25.1.	Software Home (Set Preset Positions) G92	5-97
5.26.	Feedrate Modes (G93, G94, G95).....	5-99
5.26.1.	Inverse Time Feedrate Programming (FeedrateMode) G93.....	5-99
5.26.2.	Feed Per Minute Feedrate Programming (FeedrateMode) G94	5-100
5.26.3.	Feed Per Spindle Revolution Feedrate Programming G95.....	5-101
5.26.4.	Surface Speed Spindle Feedrate Programming G96	5-102
5.26.5.	RPM Spindle Feedrate Programming G97	5-104
5.27.	Dominant Feedrate Overview (G98, G99).....	5-105
5.27.1.	Rotary Feedrate Dominant G98.....	5-106
5.27.2.	Linear Feedrate Dominant G99	5-107
5.28.	Spindle Shutdown Modes (G100, G101).....	5-108
5.28.1.	Disable Spindle Shutdown Mode G100	5-108
5.28.2.	Enable Spindle Shutdown Mode G101	5-108
5.29.	Modal Velocity Profiling (G108, G109)	5-109
5.29.1.	No Deceleration to Zero Velocity Between Moves G108.....	5-109
5.29.2.	Force Deceleration to Zero Velocity Between Moves G109.....	5-109
5.30.	Circular Direction Codes (G110, G111).....	5-110
5.30.1.	Normal Circular Interpolation G110	5-110
5.30.2.	Inverse Circular Interpolation G111.....	5-111
5.31.	Block Delete Mode (G112, G113)	5-112
5.31.1.	Set Block Delete Mode G112	5-112
5.31.2.	Clear Block Delete Mode G113	5-112

5.32.	Optional Stop Mode (G114, G115)	5-113
5.32.1.	Set Optional Stop Mode G114	5-113
5.32.2.	Clear Optional Stop Mode G115.....	5-113
5.33.	Dry Run Mode (G116, G117).....	5-113
5.33.1.	Dry Run Mode Enabled G116.....	5-113
5.33.2.	Dry Run Mode Disabled G117.....	5-113
5.34.	Servo Update Rate (G130, G131).....	5-114
5.34.1.	4 Kilohertz Servo Update Rate G130.....	5-114
5.34.2.	1 Kilohertz Servo Update Rate G131	5-114
5.35.	Cutter Tool Offset Compensation Overview (G143, G144, G149).....	5-115
5.35.1.	Activate Positive Cutter (Tool) Offsets G143	5-115
5.35.2.	Activate Negative Cutter (Tool) Offsets G144.....	5-116
5.35.3.	Deactivate Cutter (Tool) Offsets G149	5-116
5.36.	Scale Factor (G150, G151).....	5-117
5.36.1.	Clear Scale Factor G150	5-117
5.36.2.	Set Scale Factor G151	5-117
5.36.3.	The Scaling Center	5-118
5.37.	Suspend All Fixture Offsets G153	5-119
5.38.	Rotary Axis Acceleration Rates (G165, G166)	5-120
5.38.1.	Set Acceleration Rate (for Rotary Type Axes) G165	5-120
5.38.2.	Set Deceleration Rate (for Rotary Type Axes) G166.....	5-120
5.39.	Block Delete2 Mode (G212, G213)	5-121
5.39.1.	Set Block Delete2 Mode G212.....	5-121
5.39.2.	Clear Block Delete2 Mode G213	5-121
5.40.	CNC Block Look-Ahead (G300, G301)	5-122
5.40.1.	Disable Multi-Block Look-Ahead G300	5-123
5.40.2.	Enable Multi-Block Look-Ahead G301	5-123
5.40.3.	CNC Block Look-Ahead Conditions that Force (G9) Deceleration	5-123
5.40.4.	CNC Block Look-Ahead Failures	5-124
5.41.	High Speed Machining (G310, G311).....	5-126
5.41.1.	Disable High Speed Machining G310	5-126
5.41.2.	Enable High Speed Machining G311	5-126
5.41.3.	High Speed Machining Limitations	5-126
5.41.4.	Continue when Velocity command is Zero G360.....	5-127
5.41.5.	Wait till In-Position G361	5-127
5.42.	M-codes	5-128
5.42.1.	Program Stop M0	5-128
5.42.2.	Optional Stop M1	5-128
5.42.3.	End of Program M2.....	5-128
5.42.4.	Spindle On Clockwise M3, M23, M33, M43	5-129
5.42.5.	Spindle On Counterclockwise M4, M24, M34, M44	5-129
5.42.6.	Spindle Off M5, M25, M35, M45	5-130
5.42.7.	Spindle Off/Reorient M19, M219, M319, M419	5-130
5.42.8.	Restart Program Execution and Wait for Cycle Start M30	5-130
5.42.9.	Machine Lock Mode	5-131
5.42.10.	Machine Lock Enabled M41	5-131
5.42.11.	Machine Lock Disabled M42	5-131
5.42.12.	Restart Program Execution M47	5-131

5.42.13. Feedrate Override Lock M48	5-132
5.42.14. Feedrate Override Unlock M49.....	5-132
5.42.15. Spindle Feedrate Override Lock M50	5-132
5.42.16. Spindle Feedrate Override Unlock M51	5-132
5.42.17. Loop over Near Call to Subroutine M97.....	5-133
5.42.18. Loop over Far Call to Subroutine M98	5-133
5.42.19. Spindle On Clockwise Asynchronously M103, M123, M133, M143	5-134
5.42.20. Spindle On Counter-Clockwise Asynchronously M104, M124, M134, M144	5-134
CHAPTER 6: Extended Commands	6-1
6.1. Introduction	6-1
6.2. Motion with Extended Commands	6-2
6.3. Host vs. Axis Processor Based Commands	6-2
6.3.1. Axis Processor Based Extended Commands	6-2
6.3.2. Host Based Extended Commands	6-2
6.3.2.1. Time-outs	6-3
6.3.2.2. Error Returns from the CallBack Commands.....	6-3
6.3.2.3. Return Values from the Callback Commands.....	6-4
6.3.2.4. Parameters to a Callback Command.....	6-4
6.4. RS-447 Extended Commands.....	6-5
6.4.1. Auto Focus AFCO.....	6-9
6.4.2. ALIGN Command.....	6-10
6.4.3. BIND Axis Command BIND	6-12
6.4.4. Call Subroutine Command CALL / CLS	6-13
6.4.5. CallDLL Command	6-14
6.4.6. Capture Axis.....	6-14
6.4.7. CFGMASTER Command (Configure Master Axis)	6-15
6.4.8. Change Axis Configuration from within a CNC Program.....	6-15
6.4.9. COMMITIT	6-17
6.4.10. COMMSETTIMEOUT	6-18
6.4.11. Data Acquisition Start DATASTART.....	6-19
6.4.12. Data Acquisition Stop DATASTOP.....	6-25
6.4.13. Define Subroutine	6-25
6.4.14. DISABLE Axes Command	6-26
6.4.15. Displaying Text in the CDW Window DISPLAY.....	6-27
6.4.16. Define Program Variable or Array DVAR.....	6-27
6.4.16.1. Define Program Variable	6-28
6.4.16.2. Define Program Array	6-28
6.4.17. ENABLE Command.....	6-29
6.4.18. End motion (Asynchronous) ENDM	6-30
6.4.19. Execute DOS or Windows Program EXE	6-30
6.4.20. EXECCANNEDFUNCTION Command	6-32
6.4.21. Execute DOS or Windows Program and Wait for Completion.....	6-32
6.4.22. FARCALL FARCALL / PGM / PRG	6-33
6.4.23. Jump to program FARGOTO / FARJUMP	6-34

6.4.24.	FEDM Command	6-35
6.4.25.	File and Serial Port Command Overview	6-35
6.4.25.1.	File Close Command FILECLOSE	6-36
6.4.25.2.	File Existence Testing Command.....	6-36
6.4.25.3.	File Open Command FILEOPEN.....	6-36
6.4.25.4.	FILEREAD Command FILEREAD	6-37
6.4.25.5.	FILEREADINI Command.....	6-39
6.4.25.6.	File Write Command FILEWRITE	6-40
6.4.25.7.	FILEWRITEINI Command.....	6-42
6.4.26.	Free axes FREE	6-43
6.4.27.	FREECAMTABLE Command	6-43
6.4.28.	Goto to a CNC block GOTO / JUMP.....	6-44
6.4.29.	HANDWHEEL Command HAND / HANDWHEEL	6-45
6.4.30.	Home Command HOME / REF	6-46
6.4.31.	HOMEASYNC Command HOMEASYNC	6-47
6.4.32.	IF Command IF ... THEN ... ELSE ... ENDIF.....	6-47
6.4.32.1.	IF ... GOTO command	6-48
6.4.32.2.	IF ... THEN Command	6-48
6.4.33.	INDEX Command INDEX.....	6-50
6.4.34.	IsAvail, Axes Available Command	6-50
6.4.35.	Camming Motion Overview	6-51
6.4.35.1.	Axis Parameters Affecting Camming	6-53
6.4.35.2.	Axis Parameters Used To Monitor Camming Motion	6-53
6.4.35.3.	Camming Performance Tip	6-53
6.4.35.4.	Master Axis Selection	6-53
6.4.35.5.	Synchronizing Multiple Axes	6-54
6.4.35.6.	Camming Motion from a File	6-54
6.4.35.7.	Infeeding Overview	6-56
6.4.35.8.	Asynchronous Motion Commands	6-56
6.4.35.9.	Camming Example Program.....	6-58
6.4.35.10.	Cam Table Format.....	6-58
6.4.35.11.	Cam Table Format Example.....	6-60
6.4.36.	LOADCAMTABLE Command.....	6-62
6.4.37.	#MAKENCODESLABELS	6-63
6.4.38.	MAP Command MAP	6-64
6.4.39.	MaskToDouble Command	6-64
6.4.40.	MOVETO (Asynchronous Absolute Move) Command	6-65
6.4.41.	MSET Command.....	6-66
6.4.42.	MSGxxx Commands Overview	6-66
6.4.42.1.	MSGBOX Command	6-67
6.4.42.2.	MSGCLEAR Command	6-70
6.4.42.3.	MSGDISPLAY Command	6-71
6.4.42.4.	MSGHIDE Command	6-72
6.4.42.5.	MSGINPUT Command.....	6-72
6.4.42.6.	MSGLAMP# Command.....	6-74
6.4.42.7.	MSGMENU Command.....	6-75
6.4.42.8.	MSGSHOW Command.....	6-76
6.4.42.9.	MSGTASK Command	6-76
6.4.43.	ON command ON.....	6-77

6.4.44.	Conditional ONGOSUB Command ONGOSUB	6-79
6.4.45.	Oscillate Move Command OSC	6-88
6.4.46.	POPMODES Command	6-89
6.4.47.	PUSHMODES Command	6-89
6.4.48.	Initialize Touch Probe PROBE	6-90
6.4.49.	PROGRAMDOWNLOADFILE Command	6-91
6.4.50.	PROGRAMEXECUTE Command.....	6-92
6.4.51.	PROGRAMEXECUTEFILE Command	6-92
6.4.52.	PROGRAMTASKRESET Command	6-93
6.4.53.	PROGRAMUNLOAD Command	6-94
6.4.54.	PSO Card Based Commands.....	6-95
6.4.54.1.	Configuring the PSO-PC Card to Fire a Laser.....	6-96
6.4.55.	Position Synchronized Output Firing Distance Entry PSOD	6-97
6.4.55.1.	Mode Argument for PSOD Command	6-97
6.4.55.2.	Pulse Output at an Incremental Distance PSOD 0	6-97
6.4.55.3.	Fire Equidistantly PSOD 7	6-99
6.4.55.4.	Offset Firing Pulse PSOD 8	6-99
6.4.56	Enable/Disable Position Synchronized Output Firing PSOF	6-100
6.4.56.1.	Mode Arguments for PSOF.....	6-100
6.4.56.2.	Disable Laser Output Pulse PSOF 0.....	6-100
6.4.56.3.	Laser Output Fires Continuously PSOF1	6-100
6.4.56.4.	Fire Laser a Specified Number of Times PSOF 2.....	6-101
6.4.56.5	Laser Output Synchronized with Position PSOF 3	6-101
6.4.57.	Position Synchronized Output Pulse Configuration PSOP	6-102
6.4.57.1.	Mode Arguments for PSOP.....	6-102
6.4.57.2.	Simple Single Pulse PSOP 0	6-102
6.4.57.3.	Single Pulse with Lead, Width and Trail PSOP 1	6-103
6.4.57.4.	Level based Laser Control.....	6-103
6.4.57.5.	Simple One-shot Pulse PSOP 4	6-104
6.4.58.	Position Synchronized Output Scaling PSOS.....	6-105
6.4.58.1.	Disabling Scaling PSOS 0	6-105
6.4.58.2.	Enable Scaling PSOS 1	6-105
6.4.58.3.	Define PSO Axes Scaling PSOS 2	6-105
6.4.59	Digital/Analog Output Command PSOT	6-106
6.4.59.1.	MODE Argument for PSOT	6-106
6.4.59.2.	Set Individual Output State PSOT 0	6-106
6.4.59.3.	Set Analog Outputs to Discrete Values PSOT 2.....	6-107
6.4.59.4.	Velocity Ramping PSOT 4.....	6-108
6.4.59.5.	Position Ramping PSOT 6	6-109
6.4.59.6.	PSOT 4 <i>velocity</i> Argument	6-111
6.4.59.7.	PSOT 6 <i>position</i> Argument.....	6-111
6.4.60.	Release Command.....	6-112

6.4.61.	Repeat Loop REPEAT / RPT	6-112
6.4.62.	Canned Function Overview	6-113
6.4.62.1.	SETCANNEDFUNCTION Command.....	6-113
6.4.62.2.	Disabling Canned Functions.....	6-114
6.4.63.	Return from Subroutine/Program RETURN	6-117
6.4.63.1.	RETURN from an ONGOSUB Command.....	6-117
6.4.64.	SetParm Command.....	6-119
6.4.65.	Slew Command SLEW	6-119
6.4.66.	Start Motion (STRM) Command STRM.....	6-120
6.4.67.	String Functions	6-121
6.4.67.1.	STRLEN.....	6-121
6.4.67.2.	STRCMP.....	6-122
6.4.67.3.	STRFIND	6-122
6.4.67.4.	STRCHAR	6-123
6.4.67.5.	STRTODBL.....	6-123
6.4.67.6.	STRTOASCII.....	6-124
6.4.67.7.	STRUPR.....	6-124
6.4.67.8.	STRLWR	6-124
6.4.67.9.	DBLTOSTR.....	6-125
6.4.67.10.	STRMID.....	6-125
6.4.68.	SYNC Command SYNC	6-126
6.4.69.	Track Command	6-128
6.4.70.	VOLCOMP Command	6-131
6.4.71.	Wait Command WAIT	6-132
6.4.72.	Conditional Looping WHILE / WHL.....	6-132
CHAPTER 7: Custom Commands 7-1		
7.1.	Introduction	7-1
7.2.	Custom M-codes (Using Defines).....	7-2
7.2.1.	Custom M-Code Tips	7-2
7.3.	Custom G-codes (Using Calls)	7-3
7.3.1.	Custom G-Code Tips.....	7-5
7.4.	Custom Commands (Using Callback Commands)	7-5
APPENDIX A: Glossary of Terms..... A-1		
A.1.	Introduction	A-1
APPENDIX B: WARRANTY AND FIELD SERVICE B-1		
INDEX		
∇ ∇ ∇		

LIST OF FIGURES

Figure 4-1.	Flow of Execution of Compiler Directives.....	4-1
Figure 5-1.	CW Circular Interpolation.....	5-25
Figure 5-2.	Orientation of a G2, in various planes in Coord. System #1	5-26
Figure 5-3.	Orientation of a G3, in various planes in Coord. System #1	5-26
Figure 5-4.	PQ Method Example.....	5-27
Figure 5-5.	“R” Method Example.....	5-28
Figure 5-6.	Circular Radius Example	5-30
Figure 5-7.	Arc Center Change.....	5-32
Figure 5-8.	CCW Circular Interpolation.....	5-34
Figure 5-9.	G8 and G9 Velocity Profile.....	5-36
Figure 5-10.	Velocity Profile with G8	5-39
Figure 5-11.	Velocity Profile Without G9	5-40
Figure 5-12.	Velocity Profile with G9	5-41
Figure 5-13.	Coordinate System 1 (Clockwise or G2 motion).....	5-44
Figure 5-14.	Tool Orientation.....	5-45
Figure 5-15.	Normalcy Left	5-49
Figure 5-16.	Normalcy Right.....	5-49
Figure 5-17.	Coordinate System 2 Orientation (Clockwise or G2 Motion)	5-51
Figure 5-18.	Unrestricted Safe Zones	5-55
Figure 5-19.	Cutter Radius Compensation Path.....	5-59
Figure 5-20.	Cutter Compensation with Intervening Statements	5-59
Figure 5-21.	Cutter Radius Compensation Lead-On Moves	5-60
Figure 5-22.	Inside Corner.....	5-61
Figure 5-23.	Outside Corner (Diagram A).....	5-63
Figure 5-24.	Lead Off Moves	5-64
Figure 5-25.	Path Compensation Left	5-65
Figure 5-26.	Path Compensation Right.....	5-66
Figure 5-27.	Polar/Cylindrical Transformations Diagram	5-69
Figure 5-28.	X, Y, Rotational and Optional Infeed Axis	5-72
Figure 5-29.	Feedrate Changes	5-82
Figure 5-30.	UpdateTimeSec Diagram	5-84
Figure 5-31.	Constant vs. Cosine Acceleration.....	5-85
Figure 5-32.	G83 Mirror Image Example 1	5-92
Figure 5-33.	G83 Mirror Image Example 2	5-92
Figure 5-34.	G84 Parts Rotation Example.....	5-94
Figure 5-35.	Absolute Mode Programming	5-95
Figure 5-36.	Incremental Mode Programming.....	5-96
Figure 5-37.	Scale Factor Example.....	5-117
Figure 5-38.	Scaling Center Illustration.....	5-118
Figure 5-39.	Scaling Center Illustration 2.....	5-119
Figure 6-1.	Align Command Function Illustration.....	6-11
Figure 6-2.	Master/Slave Profile.....	6-55
Figure 6-3.	MSGBOX Pop-up Message Example	6-67
Figure 6-4.	The CDW Display List Window	6-71
Figure 6-5.	MSGINPUT Command Message Box Display	6-72
Figure 6-6.	MSGMENU Command Display.....	6-75
Figure 6-7.	Trigger Pulse Fired at Constant Increments	6-97

Figure 6-8.	Single Pulse Generated on Firing Condition	6-102
Figure 6-9.	Single Pulse Output with Lead, Width, and Trail.....	6-103
Figure 6-10.	Single One-shot Pulse Output	6-104
Figure 6-11.	User-Specified Analog Voltage.....	6-107
Figure 6-12.	Velocity Ramping.....	6-109
Figure 6-13.	Position Ramping	6-111
Figure 6-14.	Track Command Δ Diagram	6-129

▽ ▽ ▽

LIST OF TABLES

Table 1-1.	UNIDEX 600 Series Interface Manuals.....	1-2
Table 1-2.	Syntactic Description Language Components.....	1-6
Table 3-1.	Expression Examples	3-3
Table 3-2.	Summary of Floating-Point Operators Available (Where α and β are Arguments)	3-9
Table 3-3.	Summary of Floating-Point Functions Available (Where α is the Argument)	3-10
Table 3-4.	Operator Precedence Indexes.....	3-11
Table 3-5.	Summary of Integer Operators Available (Where α and β are Arguments).....	3-13
Table 3-6.	Summary of Bitwise Operations.....	3-14
Table 3-7.	Variant Names.....	3-17
Table 5-1.	Where to Find Details	5-2
Table 5-2.	CNC Move Options	5-3
Table 5-3.	G-code and M-code Summary	5-8
Table 5-4.	Required Order of Axes in a G44, when no G16 has been Executed.....	5-67
Table 5-5.	Transformation from an X/Y Cartesian Plane to a Polar Coordinate System	5-68
Table 5-6.	Transformation from an X/Y Cartesian Plane to a Cylindrical Coordinate System	5-71
Table 5-7.	Fixture Offset Example	5-80
Table 5-8.	Accel/Decel G-codes Summary	5-81
Table 5-9.	G-Codes to Change Axes Used for Circular Interpolation.....	5-110
Table 5-10.	Relationship of Arc Direction, Plane, & Circle Center point	5-111
Table 5-11.	The Five Look-Ahead Cases	5-122
Table 6-1.	Where to Find Details	6-1
Table 6-2.	Extended Command Categories	6-2
Table 6-3.	Extended Command Summary	6-5
Table 6-4.	Data Available for Collection	6-23
Table 6-5.	Mode Parameter Values	6-38
Table 6-6.	Configuring Camming Motion	6-52
Table 6-7.	Configuring Camming Motion Cleanup.....	6-52
Table 6-8.	Button Specifiers.....	6-68
Table 6-9.	Input Window Specifiers (* = DEFAULT).....	6-73
Table 6-10.	Button Specifiers (* = DEFAULT)	6-73
Table 6-11.	Distance Calculations for Multiple Axes Using the PSOD Command	6-98

▽ ▽ ▽

PREFACE

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

CHAPTER 1: OVERVIEW

This chapter introduces the Aerotech CNC language.

CHAPTER 2: COMMANDS

This chapter describes the general syntax of CNC programs and lines. It explains the different sets of commands available. This chapter does not address the functionality or syntax of any specific commands; this is done in chapters 4,5, and 6.

CHAPTER 3: EXPRESSIONS

This chapter describes the syntax and functionality of all the elements of the Aerotech CNC language except for those that specifically relate to specific commands. For example, this chapter describes how to construct variables and expressions, but does not explain how to construct a **G1** or an **IF** command.

CHAPTER 4: COMPILER DIRECTIVE COMMANDS

This chapter describes the syntax and functionality of the Compiler Directive commands.

CHAPTER 5: G-CODE COMMANDS

Chapter 5 describes the syntax and functionality of the G-code commands.

CHAPTER 6: EXTENDED COMMANDS

Chapter 6 describes the syntax and functionality of the Extended commands.

CHAPTER 7: CUSTOM COMMANDS

This chapter describes how the user can define their own G or M-codes to be the execution of a custom CNC program. This is especially useful in the context of defining complex I/O functionality.

APPENDIX A: GLOSSARY OF TERMS

APPENDIX B: WARRANTY AND FIELD SERVICE

Appendix B 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. Locator page numbers appear in one style: standard serif font (e.g., 3-1).

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. Readers 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 modal symbol (see left) appears in the outer margin next to commands that are modal for quick reference
- Each section in the remaining chapters of this manual describes one or more language elements, where a language element is defined as a command or an expression. Each section details *both* the semantic and syntactic features of the language element(s) it addresses
- The callback symbol (see left) appears in the outer margin next to commands that are callback commands for quick reference
- Many sections will use language elements defined in other sections or chapters, and therefore the user may have to reference several sections to understand a single language element. However to make things easier, the language elements within a chapter (aside from the introduction section) are organized in alphabetic order, keyed on the language element name. As a further aid, "page tabs" are shown in the outside margins, naming the language elements described within that page
- Note symbols (see left) appear in the outer margins next to notes following sections or paragraphs
- The section title will usually match the name of the language element described within, aside from minor formatting differences. For example, the language element "<axisPoint>" is described under the section titled "Axis Points." However some sections will describe more than one language object. For example the section on "Floating Point Numbers" contains descriptions of the two language elements: "<floatingPointConstants>" and "<floatingPointVariables>". Therefore, the description of a language object may not always be found under a section title equivalent to its name.
- MFO is an acronym for Manual Feed Override
- 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:	Page
• Introduction.....	1-1
• Purpose of This Manual	1-1
• Prerequisites.....	1-2
• Related Documentation	1-2
• CNC Design Philosophy	1-3
• CNC Functionality Summary	1-4
• Syntactic Descriptions.....	1-4

1.1. Introduction

The UNIDEX 600 Series Computer Numerical Control (CNC) programming language is a large and powerful set of commands that permit the user to write simple or complex motion control programs in a large variety of formats.

This chapter summarizes the major features of the UNIDEX 600 Series CNC language.

1.2. Purpose of This Manual

This manual is for programmers writing CNC (Computer Numerically Controlled) programs on a UNIDEX 600 Series Motion controller.

This manual does not require prior knowledge of the CNC language. It covers all aspects of the U600 Series CNC language syntax and functionality, serving as both a reference and tutorial on the subject. Also, it is the only manual addressing the Aerotech CNC language of the U600 Series controllers.

Running a CNC program involves at least the following steps:

1. Writing correct CNC program text.
2. Executing the CNC program including the following steps
 - 2a. Running the CNC compiler to produce a binary object file from the program text.
 - 2b. Downloading the binary object file to the U600 Motion Controller.
 - 2c. Executing and monitoring the program on the U600 Motion Controller.

This manual only addresses step 1. Please see the documentation relative to the interface being used for information on how to run the CNC compiler, download programs, and run programs on the Motion Controller card. Table 1-1 lists the appropriate manuals.

Table 1-1. UNIDEX 600 Series Interface Manuals

Interface	Manual
C language and/or Visual Basic	EDU156 U600 Series Library Reference, Win NT/95 Manual
C++ Language	SDK Online Help File
MMI	MMI Online Help File

1.3. Prerequisites

Prior knowledge of CNC language standards and related industry accepted definitions of M-codes and G-codes is not required.

Prior knowledge of programming in general is not strictly required. However, it must be expected that any reader who is inexperienced in writing programs will not be able to easily construct correct Aerotech U600 CNC programs due to the numerous pitfalls and obstacles inherent in constructing any working computer program.

Knowledge of motion control concepts such as feedback loops and contoured motion are not strictly required. However, it must be expected that the reader who is inexperienced in motion control concepts will not be able to easily construct the more sophisticated CNC programs initially.

The reader must read and understand the CNC programming sections in the programming chapter of the *U600 Series User's Guide*, P/N EDU157, to take full advantage of the CNC interface and avoid wasting time writing inefficient or poorly operating CNC programs.

1.4. Related Documentation

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.

1.4.1. Hardware Manuals

For a description of the controller's hardware, reference the hardware manual for 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).

1.4.2. Programming Manuals

This manual covers the CNC G-code programming language. For Visual Basic and C programmers, reference the UNIDEX 600 Series Library Reference Manual (P/N EDU156). For OLE Custom Control programming, reference the Software Development Kit Online Help File.

1.4.3. MMI Interface Manual

For information on using the MMI600-NT CNC Application, reference the MMI Online Help File).

1.5. CNC Design Philosophy

The U600 CNC language is specifically designed to achieve the following goals: *flexibility, reliability, and power*. The following major design decisions influenced the U600 Series CNC Compiler construction.

- All capabilities available to the U600 Series Motion Controller must be available to the U600 Series CNC compiler.
- An externally provided parser was used within the compiler: Visual Parse++™ by SandStone. Syntax parsing is a complex subject, only mastered over the decades and is best left to the experts and specialists in that field.
- Bounds and limits were avoided as much as possible by taking advantage of dynamic allocation. For example, there is no inherent limit to the number of lines in a program, characters in a variable, program depth, and WHILE block nesting.
- Whenever possible, C language constructs and conventions are followed. For example, the conditional and bit operators are identical to those used in the C programming language.
- As much as possible, variant forms of equivalent syntax were accommodated. For example, both “WHILE” and “whl” are allowed.
- Functionality is matched by single or multiple test cases maintained in a regression library continuously tested and re-tested. This guarantees a minimum level of functionality regardless of new features added. Aerotech developed an “approximation differencing” capability to compare test results that come from actual motion on the controller. In other words, we can test functionality even in the presence of random variations caused by inter-processor timing or physical environment variation such as vibration.
- M code requirements vary widely based on the specific application. For example, when setting an output bit, the programmer needs the CNC to wait until one or more input bits are set, verifying the setting of the I/O before continuing to the next step in the CNC Program. Aerotech offers maximum flexibility by allowing M-codes to be CNC programs where the user can define any sort of behavior. Refer to Chapter 7: Custom Commands for details.

1.6. CNC Functionality Summary

The Aerotech CNC language provides all the functionality specified by the RS-274D and RS-447 documents, plus a great deal more. Major motion and I/O capabilities that Aerotech has added to the Recommended Standards include:

- Simultaneous contoured moves along a circular arc and circular or linear paths.
- Coordinate system rotation and mirroring.
- Up to sixteen spindles (4 per task).
- Analog I/O.
- Simultaneous rotational and linear movement.
- Cutter Compensation.
- Normalcy.

In addition to new motion capabilities, the U600 CNC has added significant program control related features:

- User defined G-codes or M-codes.
- Simultaneous execution of up to four CNC programs.
- Multiple program storage capability.
- Automatic program compilation, and dependency detection.
- Block “if-endif” type constructs.
- Subroutines with parameters, returns and stack capabilities.

1.7. Syntactic Descriptions

The syntactic description is especially important in a language as diverse as the U600 CNC. If there were only a few options for constructing a language element, then the explanation text could describe these syntax options. However, since the U600 CNC language’s flexibility and power allow the programmer to express language elements in so many different forms, it makes it incredibly tedious to describe all the syntax options in the text. The syntactic description language described in the next section permits us to provide a necessary and sufficient description of all valid syntax options in a concise fashion (usually one line). However, the examples and explanation text do emphasize and clarify the most important and least obvious features of the syntax.

Each language object description always consists of three parts, presented in the order shown below:

1. “Syntactic description(s)”
2. Explanation text
3. Example(s)

The format of the presentation of these sections is shown on the next page.

SYNTAX:

...

EXAMPLES:

....

SEE ALSO:

...

A syntax description follows on the same line as the SYNTAX: text, and in some cases occupies more than one line. A section may contain multiple syntax descriptions. The ellipses (...) between the syntax and examples represent the explanation text; sometimes presented in multiple paragraphs. Each example occupies its own line, and there may be multiple examples. Inline comments often accompany examples.

The semantics or meaning of a language element is in the explanation text and comments attached to the examples. The explanation and the examples also provide some information relating to the syntax (the format) of the language element, but only the “Syntactic description” provides a concise and complete expression of the required syntax.

1.7.1. Elements of the Syntactic Description

The three elements of the syntactic description language are names, keywords, and specifiers. Names refer to a syntactic object and keywords are alphanumeric text that must be typed exactly as shown. Specifiers provide additional information on how keywords and names can be combined.

Table 1-2 specifies all the elements of the syntactic description language along with examples of their use.

Notice the use of *italics*, **bold**, underlining and subscripts (_{subscripts}) to differentiate the various elements.



Please refer to the examples in the following sections for more clarification on the meaning of various elements.

Table 1-2. Syntactic Description Language Components

Element	Example	Meaning
Name	< <i>fConst</i> >	Names a basic syntactic element in the language (see Chapter 3).
Keyword	\$GLOB	Must be typed exactly as shown; not case sensitive.
<u>is</u>	< <i>dog</i> > <u>is</u> SPOT	Specifies that a definition of a name follows on the same line.
<u>or</u>	< <i>dog</i> > <u>or</u> < <i>cat</i> >	This specifies an alternative between two choices.
<u>except</u>	< <i>digit</i> > <u>except</u> 0	Sometimes certain forms must be excluded from the general case.
[]	[[<u>option</u>]]	The object between the double brackets is optional.
₁	< <i>digit</i> > ₁	The preceding object can be repeated one or more times.
₀	< <i>space</i> > ₀	The preceding object can be repeated zero or more times.
()	PART(A or B)	Sometimes parentheses are needed to indicate grouping. The example shown is equivalent to: PARTA or PARTB .
~	~	Indicates the optional use of white space.
...	A or B or C ... Z	In some cases ellipses will be used to abbreviate an obvious progression.

1.7.2. Keyword Case Sensitivity

Keywords are alphanumeric text the user must type exactly as shown, except for case. The user can enter the text in either upper or lowercase even though all keywords in this manual are shown in uppercase.

However, text in a keyword can never consist of mixed case characters. For example, “MAP” and “map” are legal and equivalent keywords, but “Map” and “mAp” are not keywords.

Parameters, however, have a defined case, as shown in AerParam.Pgm. Axis parameters are always entirely uppercase. Global, task, and machine parameters have mixed case with the first letter of each word within the parameter capitalized, i.e. CntsPerInch.

1.7.3. White space

White space is defined as any non-zero number of consecutive spaces, tabs, commas, backspaces, or deletes (ASCII codes 32, 9, 44, 8 and 127, respectively). The contents of white space is always ignored in the U600 language. In other words, all forms of white space are syntactically and semantically equivalent.

White space is sometimes required in the U600 CNC language and is sometimes optional. Spaces within a syntactic definition mean that some white space is required. A “~” in a syntactic description means that white space is optional. In general, white space is required in Compiler Directive and Extended commands, but optional in G-code commands.

The example that follows illustrates how the MAP command must be separated from its parameter (an axis point) by white space. Therefore, “MAPX4” is illegal, while “MAP X4” and “MAP,,X4” are legal and equivalent. The second example illustrates that the white space is optional in a “G0”, so “G0 X4” and “G0X4” and “G0,,X4” are all legal and equivalent.

```
<mapCommand> is MAP <axispoint>
<g0Command>  is G0~<axispoint>
```

However, spaces appearing in a syntactic definition on either side of a textual separator (an is or or) are purely for readability and not part of the definition syntax. For example, the spaces separating the “is” from the “MAP” above are not relevant.

1.7.4. Syntax Description Examples

Here are some examples of syntactic language definitions along with text describing the meaning.

SYNTAX: <fValue> is <fConstant> or <fVariable>
 <fVariable> is \$GLOB <integer>

This first example defines a “*fValue*” object as being a “*fConstant*” or a “*fVariable*” object. The second example defines a “*fVariable*” as the letters “\$GLOB” followed by an integer.

Most syntax definitions rely on other syntactic definitions. For example, the <integer> and <fConstant> elements used in the above definitions must be defined elsewhere in order for the above definitions to be complete. The index and table of contents can be used to quickly locate syntax definitions that are not immediately visible.



A more complex example shown below defines a floating-point constant.

SYNTAX: <fConstant> is [[- or +]](<digit>1,<digit>0 or <digit>0,<digit>1))

The above syntax description describes the syntax for real numbers like -.8 or 5. or 6.9. It's probably the most complicated syntax description in this manual.

1.7.5. Limitations and Bounds

A conscious effort was made to avoid all bounds and limits when constructing the compiler by taking advantage of dynamic allocation. Of course, the available memory of the system is always a limitation, but for the purposes of this discussion that is ignored.

For example, there is no limit to the number of lines in a program or number of letters in a variable or label. Also, there are no limits to program depth, WHILE or SUBROUTINE block nesting, or INCLUDE file nesting.

However, there are some cases where limitations were forced by the motion controller or general programming considerations. For example, the motion controller dictates that

“string32” type constants cannot be more than 32 characters long. Another example is that the length of the line is limited to 1028 characters.

If no limitation or bound is included in a description, then it is inferred that no such limitation exists. If a limit does exist, it will be defined in the documentation.

▽ ▽ ▽

CHAPTER 2: COMMANDS

In This Section:	Page
• Description	2-1
• Programs	2-2

2.1. Description

This chapter introduces the Aerotech U600 Series CNC language commands, and the general rules on how commands are constructed. The information contained in this chapter is relevant to all U600 Series CNC commands. Information specific to any of the three command sets is found in Chapters 4, 5 and 6.

2.1.1. Command Sets

The U600 Series CNC Programming language syntax really consists of three language syntax joined together.

1. A “Compiler Directive” command set (based on ANSI C syntax)
2. A “G-code” command set (based on RS-274D syntax)
3. An “Extended” command set (based on RS-447 syntax)

Any Aerotech CNC line may be written in any of the three syntax, but syntax cannot be mixed on the same line. For example “G91” is a legitimate G-code command, and “MAP X1” is a legitimate extended command, but “G91 MAP X1” is not a legitimate U600 CNC command.

To users familiar with CNC code, the term G-code used in this document refers to more than just CNC words starting with a “G” (such as G90). The term G-code includes all the syntax defined in RS-274D that includes G-codes, M-codes, F-codes, etc. For example, the line N07 G2 X5 Y6 I5 J7 F30 is legal G-code syntax.

2.1.2. Conformance to Standards

The “Compiler Directive” syntax is loosely based on a subset of the ANSI C standard Compiler Directive language, but no guarantee is given of its compatibility to the ANSI C or any other standard.

The G-code and extended syntax were designed to follow the RS-274D and RS-447 standards. However, in some cases the Aerotech U600 syntax may vary slightly from the described standards. These differences are usually trivial and were adopted to add extra flexibility and power. For example, the RS-447 standard requires that any extended CNC command must be surrounded by parentheses, but in Aerotech’s language the user can optionally omit these parentheses.

2.2. Programs

This section contains the first usage of the syntactical description language. For notes on how to interpret the Syntactic Description Language, reference Chapter 3 within this manual.

SYNTAX: *<CNCPromgram>* is (*[<CNCLine>]*)~(*[<Comment>]*)~*<lineTerminator>*)₀

The above description states that a CNC program consists of a series of CNC lines and/or comments.

Terminate each line with a line terminator.



The comment, line, and terminator may be separated by white space.

2.2.1. White space

The “~” character in the syntactic description language indicates that white space may be included (refer to Table 1-2 in Chapter 1). White space is defined as any non-zero number of consecutive spaces, tabs, backspaces or deletes (ASCII codes 32, 9, 44, 8 and 127, respectively). The content of white space is always ignored; in other words, all forms of white space are syntactically and semantically equivalent.

2.2.2. Line Terminators

SYNTAX : *<lineTerminator>* is CRLF

A *line terminator* is the pair of characters, carriage return and linefeed. These are ASCII codes 13 and 10, respectively, and cannot be separated by white space.

2.2.3. Characters

SYNTAX : <i><ASCIIcharacter></i>	<u>is</u> ASCII codes: (0 <u>or</u> 1 <u>or</u> ... <u>or</u> 127)
<i><printableCharacter></i>	<u>is</u> ASCII codes: (32 <u>or</u> 33 <u>or</u> ... <u>or</u> 126)
<i><digit></i>	<u>is</u> 0 <u>or</u> 1 <u>or</u> 2 ... <u>or</u> 9
<i><letter></i>	<u>is</u> A <u>or</u> B ... <u>or</u> Z
<i><underscore></i>	<u>is</u> _
<i><period></i>	<u>is</u> .
<i><space></i>	<u>is</u> ASCII code 32 <u>or</u> ASCII code 9 (TAB)
<i><alphanumeric></i>	<u>is</u> (<digit> <u>or</u> <letter> <u>or</u> <underscore>) ₁

Some simple definitions of character types are necessary before proceeding. This chapter and the following chapters contain those definitions.

In general, program text can be any series of printable characters. The ASCII characters outside of the printable range contained in a valid CNC line are backspace, tab, carriage return, line feed and delete (ASCII codes 8, 9, 13, 10 and 127, respectively). These non-printable characters are used as white space and line terminator characters (see Sections 2.2.1. and 2.2.2.).

2.2.4. Comments

SYNTAX: <comment> is ;<ASCIIcharacter>₁ <lineTerminator>

The semicolon character “;” is the start of a comment. The compiler ignores all text following this character (on the same line) when executing the program. The line terminator terminates this comment. There is no multi-line comment operator in the CNC language; each comment line *must* begin with a semicolon.

2.2.5. Lines

SYNTAX: <CNCLine> is ~(<CompilerDirectiveLine> or <GCodeLine> or <ExtendedLine>)

The above description states that any CNC line can be in one of the three syntax, but not a combination of all three.

Also, any of the three types of lines may be preceded by white space.



A line should not be more than 1028 characters. If a line is too long, the compiler returns an error. However, in some cases the compiler accepts a line that is between 1028 and 2056 characters long. Nevertheless, to guarantee acceptance of a line, it should be less than 1028 characters.

If a preprocessor line exceeds the line length limit (usually in the context of a user defined M-code), then the programmer can break the line into multiple lines with the “\” preprocessor command character.

2.2.6. Block Delete

The “/” character, if it is the first non-white-space character on the line, is the block delete character. Block delete permits omission of certain program lines during program execution. When the block delete setting is on, all lines prefixed by the “/” character will be treated as comments (not executed). When the block delete setting is off, lines designated as block delete lines are treated no differently than normal lines.

The block delete feature cannot be used for compiler directives.



▽ ▽ ▽

CHAPTER 3: EXPRESSIONS

In This Section:	Page
• Description	3-1
• Expressions	3-1
• CNC and Axis Letters	3-3
• CNC Masks	3-4
• CNC Words.....	3-4
• CNC Block Expressions.....	3-5
• Floating-Point Expressions.....	3-7
• Integer Expressions	3-12
• String 32 Expressions.....	3-14
• Labels.....	3-15
• Variants	3-16

3.1. Description

This chapter describes the syntax for expressions in the UNIDEX 600 Series CNC language. Expressions are used as arguments to both G-code and Extended commands, but cannot be used in Compiler Directive commands.

This chapter also describes the use of all symbols (non-alphabetic character strings) except for those few covered in Chapter 1 and Chapter 2 (such as the comment character) pertaining to Compile Directive Commands. This includes the assignment operator, comparators, and all arithmetic operators. It also describes the use of all constants, and all variant quantities (variables, parameters, and I/O elements).

This chapter contains exhaustive descriptions, so it may be tedious to read through as a tutorial. We recommend that the user find the commands they want to use in Chapter 5 and Chapter 6, and reference back to here for definitions of the expressions used by that command.

3.2. Expressions

This section defines what an expression is, while simultaneously defining the terms used in the remainder of this chapter.

3.2.1. Expression Elements

SYNTAX: `<expression> is <element><element><element>...`

EXAMPLE: `8*(6+$GLOB0)` ; elements are: 8,*,and (6+\$GLOB0),
; where (6+\$GLOB) is in itself an expression

Expressions are composed of a series of expression elements. Expression elements are the basic building blocks of the UNIDEX 600 Series CNC command line. The most familiar form of an expression is a mathematical operation, like the example above.



Elements of an expression can be expressions themselves.

Expression elements are classified in two ways: by “type” or by “component.” Together these classifications describe what the element is and its use.

3.2.2. Expression Types

SYNTAX: $\langle\text{expression}\rangle \text{ is } \langle\text{CNCLetter}\rangle \text{ or } \langle\text{NCMask}\rangle \text{ or } \langle\text{NCWord}\rangle \text{ or } \langle\text{NCBlock}\rangle \text{ or } \langle f\text{Expression}\rangle \text{ or } \langle\text{integer}\rangle \text{ or } \langle s32\text{Expression}\rangle \text{ or } \langle\text{label}\rangle \text{ or } \langle\text{conditionalExpression}\rangle$

EXAMPLE: $8*(6+\$GLOB0)$

Expression types describe *what* is stored in the expression element.

The example illustrates a $\langle f\text{Expression}\rangle$ type expression. All of the objects on the left side of the “is” are different types of expressions.

When the CNC line containing an expression executes, its elements are combined, evaluated, or reduced by the compiler and/or controller into a single element of constant type. In order for that reduction to take place, an expression must be composed of elements of the same type. For example, an expression cannot contain a floating point element and CNC mask element.

3.2.3. Expression Components

SYNTAX: $\langle\text{expression}\rangle \text{ is } \langle\text{constant}\rangle \text{ or } \langle\text{variant}\rangle \text{ or } \langle\text{operator}\rangle \text{ or } \langle\text{function}\rangle \text{ or } \langle\text{variant}\rangle \text{ is } \langle\text{variable}\rangle \text{ or } \langle\text{parameter}\rangle \text{ or } \langle\text{I/O}\rangle$

Expression components describe *how* the expression is stored and manipulated.

For example, the floating point expression “7 +9” consists of two floating-point constants and a floating-point operator.

Variants is the name used to describe the collection of objects whose contents can change during program execution. Variant values may change because the programmer changes them (e.g., variables) or because the controller changes them (e.g., the CLOCK parameter). Variants include all variables, parameters, and I/O elements. For more details on variants, see Section 3.11.

3.2.4. Expression Examples

Each expression has a specific “type” and a specific “component.” Table 3-1 illustrates examples of each component of each expression type. This chart is not comprehensive, since a number of variant components are not listed here (for more details on variants, see Section 3.11).

All expression types have a constant form, but many types only have the constant form available.



Table 3-1. Expression Examples

		Component					
	Constants	Variants		Operators	Functions		
Type		global var.	global parm.	I/O			
CNCLetter	X
CNCMask	X Y
CNCWord	X5.2
CNCBlock	X5 Y1.7	\$APTGLOB1
Float	6.7	\$GLOB1	\$PLOB1	...	+	SQRT	
Integer	6	\$BI2
Labels	startHere
String32	"string"	\$STRGLOB1	+

3.3. CNC and Axis Letters

SYNTAX: *<commandLetter>* is [[G or M or g or m]]
<argLetter> is [[O or P or Q or R or o or p or q or r]]
<taskLetter> is [[F or E or S or I or J or K or T or
 f or e or s or i or j or k or t]]
<axisLetter> is [[X or Y or Z or U or A or B or C or D
 or x or y or z or u or a or b or c or d]]

EXAMPLE: X

This expression type is a set of reserved letters. There are no variants, operators, or functions available for this expression type, only constants. Normally, the case of the letter is unimportant; lowercase and uppercase receive identical treatment. The exception to this rule applies to the axis letters; each case of a letter specifies a different axis.

An axis letter indicates a particular axis. Since there are 16 axes on the UNIDEX 600 Series controller, there are sixteen axis letters available: X, Y, Z, U, A, B, C, D, x, y, z, u, a, b, c, d. By default, these axis letters are associated with channels 1 through 16, respectively, although the MAP extended command can easily change this mapping.



The lower and uppercase letters indicate different axes. For example, **C** is axis 7 and **c** is axis 15, and neither one is related to the other.

CNC letters, rarely used by themselves, are important only as parts of CNC words and CNC masks. The exception to this is the asynchronous motion commands that take axis letters as arguments.

The programmer should be aware that they can use longer names in place of axis letters to represent axes. Refer to the “#axisnames” documentation in Chapter 4 for instructions.

3.4. CNC Masks (Axis Masks)

SYNTAX: $\langle CNC\ Mask \rangle \text{ is } \langle axisLetter \rangle_1$

EXAMPLE: X Y Z

CNC Masks (Axis Masks) specify one or more axes. There are no CNC Mask variants, operators, or functions for this expression type, only constants are available.

CNC masks consist of a series of one or more axis letters, where each letter *must* be separated by white space, i.e.; X Y Z is legal, as is X, Y, Z, but XYZ is not legal. Axes masks are only used in the axis contention (e.g., **FREE**, **BIND**) and asynchronous motion (e.g., **INDEX**, **STRM**) extended commands and in the **MASKTODBL** command.

3.5. CNC Words

SYNTAX: $\langle axisWord \rangle \text{ is } \langle axisLetter \rangle \sim \langle fExpression \rangle$

EXAMPLE: X5.5

SYNTAX: $\langle commandWord \rangle \text{ is } \langle commandLetter \rangle \sim \langle integer \rangle$

EXAMPLE: G2

SYNTAX: $\langle taskWord \rangle \text{ is } \langle taskLetter \rangle \sim \langle fExpression \rangle$

EXAMPLE: F100

SYNTAX: $\langle argumentWord \rangle \text{ is } (\langle axisLetter \rangle \text{ or } \langle argLetter \rangle) \sim \langle fExpression \rangle$

EXAMPLE: p9.9

CNC words are the foundation of all G-code commands and expressions; also used in some extended commands as Axis Points and Argument Points. There are four types of CNC words shown above.

Each CNC word consists of two parts: a “letter” followed by an “expression.” These two parts are optionally separated by white space. There are various restrictions on both parts, depending on the type of CNC word, see the syntax above, and notes below.

The Axis Word accepts axis names in the “expression.” These can be more than one letter, please see the #axisnames statement.



The Command Word accepts only integer constants in the “expression.”



The argument word is slightly different in syntax than the others, because it can use either argument letters or axis letters. In other words, axis letters do double duty, serving in two different capacities depending on the context.



The Argument Word accepts parameter names in the “expression.” These can be more than one letter, please see the #parmnames statement.



3.6. CNC Block Expressions

SYNTAX: *<CNCBlock> is <CNCBlockConstant> or <APTVariable>*

CNC blocks are either constants or variables. There are no operators or functions available.

3.6.1. CNC Block Constants

SYNTAX: *<CNCBlockConstant> is (<CNCWord>),
<CNCBlockConstant> is <GCodeBlock> or <axisPoint> or <argumentList>*

EXAMPLE: X5 Y\$GLOB6 Z5.5

CNC expressions are a collection of one or more CNC words, optionally separated by white space. However, there are many types of CNC block constants (see descriptions in the following sections).

3.6.1.1. CNC G-code Blocks

SYNTAX: `<GCodeBlock> is (<commandWord> or <axisPoint> or <taskWord>~)`

EXAMPLE: X5 Y\$GLOB6 Z5.5

G-code blocks are used exclusively in G-code commands. The syntax shown above is a general description and is not very useful. There are many other restrictions on the order of the words within a G-code block. Refer to Chapter 5: G-codes, for more details on G-code block syntax and meaning.

3.6.1.2. Axis Points

SYNTAX: `<axisPoint> is (<axisWord>~) or <APTVariable>`

except : may not contain more than one word that uses the same CNC letter

EXAMPLE:

X5	; a constant axis point
X5 Y6	; a constant axis point
X5 Y=6	; a constant axis point
X(\$GLOB5) Y6	; a constant axis point, with a variable value for X
X5 Y\$GLOB6 Z5.5	; a constant axis point, with a variable value for Y
\$APTGLOB7	; an axis point variable

A CNC axis point is a specific type of CNC block that specifies a point, plane, or position in space, where the space spans the axes in the system. If it is a constant, it is a series of CNC words, any of which can be separated by white space, or not separated at all.

For example, in a three-axis system, “X0.0,Y0.0,Z0.0” represents the origin point in the coordinate system whose axes are X, Y and Z.

In G-code commands, the CNC Axis Point specifies a point in space to move to. They are also used in **MAP** extended commands.

3.6.1.3. Argument Lists

SYNTAX: `<argumentList> is (<argumentWord>~)`

except : may not contain more than one word that uses the same CNC letter

EXAMPLE: P5 Y\$GLOB6 q5.5

A CNC argument list specifies a set of arguments to pass to a subroutine or program. See the **CALL** extended command for more details on program and subroutine calling.

There are twenty arguments that may be passed, including the 16 axis names (X, Y, Z, U, A, B, C, D, x, y, z, u, a, b, c, d), and O, P, Q and R. The O, P, Q, R arguments have no corresponding lowercase argument, so they are case insensitive, allowing o, p, q and r to be used interchangeably. These parameter names may be re-assigned.

3.6.2. APT Variables

APT Variables are variants that can hold either Axis Points or argument lists. An axis point variable is actually an array that contains one value for each axis. This allows each of the axis point variables to maintain the coordinates for a 16 dimensional point in space. See Section 3.11 for examples of use and details on variable scope, initialization, and usage.

EXAMPLES:

```
$APTGLOB0 = X5 Y7 Z2
```

```
G1 $APTGLOB0
```

3.7. Floating-Point Expressions

SYNTAX: *<fExpression> is [[()]] <fConstant> or <fVariant> or <fOperatorExpression> or <fFunctionExpression> or <iExpression> [()]*

EXAMPLE: 8*(\$GLOB7+\$GLOB[9+2])

Floating-point expressions do the bulk of the work in calculations. Simply put, they are arithmetic combinations of constants and variants. Constants, variants, operators, and functions are all available for the floating-point type (see Table 3-2). However, to properly construct floating-point expressions from operators and functions, the programmer must understand the precedence rules described in Section 3.7.5.

3.7.1. Floating-Point Constants

SYNTAX: *<fConstant> is [[- or +]](<digit>₁.<digit>₀ or <digit>₀.<digit>₁) or <integer>*

EXAMPLE: 8.9982

A floating-point constant is a signed decimal value.

The user cannot express a floating-point constant in scientific notation.



There can be no white space within the constant.



Any integer constant is also a valid floating-point constant.



There is no limit to the number of digits in a floating-point constant, but there is limited precision beyond which truncation of digits occurs. The UNIDEX 600 Series motion controller supports an 8 byte (64 bit) value, where floating point values are stored as 52 bit binary numbers (15 decimal digits), with the remaining 11 bits used to store the mantissa. In other words, the value “123456789.012345” is recognized as a different value compared with the value “123456789.012344”, but is not recognized any different than the value “123456789.0123451”.

Also, there is a limit to the range of entering a floating-point constant. The maximum absolute value allowed is 1.7 E308; the minimum absolute value is 1.7 E-308. If a constant is defined outside this range, the compiler delivers a warning message and the value is set to the appropriate limit.

3.7.2. Floating Point Variables

Please see Section 3.11 for examples of use and details on variable scope, initialization, and usage.

3.7.3. Floating Point Operators

SYNTAX: *<fOperatorExpression>* is *<fOperator>~<fExpression>* or
<fExpression>~<fOperator>~<fExpression>

EXAMPLE: 6+9.9

The floating-point expressions in Operator Expressions are called arguments. Operators are either unary (require one argument) or binary (require two arguments). The unary operators follow the syntax defined on the first syntax line definition above, the binary operators follow the syntax definition defined on the second line, above. The precision of the floating-point operators may be limited via the NumDecimalsCompare task parameter.

SYNTAX: *<Foperator>* is + or - or * or / or % or **
or OR or AND or NOT
or || or && or !
or EQ or NE or GT or LT or GE or LE
or == or != or > or < or >= or <=

EXAMPLE: GT

The first line of operators in the floating-point operator syntax definition above is just the standard mathematical operations available on a calculator. The second line comprises logical operators, whose result is always 0.0 or 1.0. The third line is the ANSI C equivalents to the verbal form lying directly above it on the second line. For example, “||” is equivalent to “OR”.

The fourth line of operators shown are comparators. Normally these are used only in an “IF” statement or other conditional statement, although they can be used anywhere other floating-point operators are used. These operators evaluate to one if the condition is true, and evaluate to zero if the condition is false. For example, “\$GLOBAL0=3 EQ 9” would assign zero to global variable 0.

The fifth and last line in the syntax description is the ANSI equivalent form of the verbal comparators directly above them in the fourth line. For example “==“ is equivalent to “EQ”.

All the symbolic forms shown in the syntax description above (lines one, three and five) are compatible with ANSI C or C++.

The operator expression is evaluated, or converted into, a floating-point constant when the controller executes the line containing that operator expression.

In some cases certain arguments are illegal, like division by zero (refer to Table 3-2). If an illegal argument is a constant, the compiler generates an error. However, if the argument is not a constant and is evaluated to an illegal argument when the line executes, the motion controller generates an appropriate fault when that line executes.

Table 3-2 shows the operators, their result, and any restrictions on their input arguments. The symbol α denotes the first argument and the symbol β denotes the second argument, if any.

Table 3-2. Summary of Floating-Point Operators Available (Where α and β are Arguments)

Operator	Args	Restrictions	Description of Result
+	2	...	α plus β
-	2	...	α minus β
*	2	...	product of α and β
/	2	$\beta \neq 0$	α divided by β
%	2	$\beta \neq 0$	α mod β (remainder of integer division)
**	2	not: $\alpha < 0$, β not integer	α raised to the β power
OR	2	...	Logical or of α and β (1.0 if either $\alpha \neq 0$ or $\beta \neq 0$)
AND	2	...	Logical and of α and β (1.0 if both α and $\beta \neq 0$)
NOT	1	...	1.0 if α is zero, 0.0 otherwise
EQ	2		1 when α equals β , otherwise 0
NE	2		1 when α not equal to β , otherwise 0
GT	2		1 when α greater than β , otherwise 0
LT	2		1 when α less than β , otherwise 0
GE	2		1 when α greater than or equal to β , otherwise 0
LE	2		1 when α less than or equal to β , otherwise 0

3.7.4. Floating Point Functions

SYNTAX: $<fFunctionExpression>$ is $<fFunction>$ ($<fExpression>$) $<fFunction>$
 is INT or ABS or SQRT or FRAC or EXP or SIN or
 COS or TAN or ASIN or ACOS or ATAN

EXAMPLE: SQRT(6.7)

Floating-point functions are unary, where they take only one expression (called an argument) within parentheses. The function is evaluated or converted into a floating-point constant when the axis processor card executes the line containing that function.

In some cases certain arguments are illegal, like the square root of a negative number (refer to Table 3-3). If an illegal argument is a constant, then the compiler delivers an error. However, if the argument is not a constant and is evaluated to an illegal argument when the line executes, the motion controller generates an appropriate fault when that line executes.

Table 3-3 shows the functions, their result, and any restrictions on their input arguments. The symbol α denotes the single argument.

Table 3-3. Summary of Floating-Point Functions Available (Where α is the Argument)

Function	Restrictions	Description of Result
ABS	...	Absolute value of α
FRAC	...	Fractional part (subtracts the integer portion) of α
EXP	...	e to the power of α
INT	...	Integer value (truncation of fraction) of α
SQRT	$\alpha \geq 0$	Square root of α
SIN	α in RADIANS	Trigonometric sine of α
COS	α in RADIANS	Trigonometric cosine of α
TAN	α in RADIANS	Trigonometric tangent of α
ASIN	$-1 \leq \alpha \leq 1$	Trigonometric inverse sign of α (result in RADIANS)
ACOS	$-1 \leq \alpha \leq 1$	Trigonometric inverse cosine of α (result in RADIANS)
ATAN	...	Trigonometric inverse tangent of α (result in RADIANS)

The FRAC and INT functions truncate the fractional and integer portions of the input expression, regardless of the sign. In other words, $\alpha = \text{INT}(\alpha) + \text{FRAC}(\alpha)$, for all values of α .

3.7.5. Floating Point Computation Precedence

If a floating point expression contains a succession of more than one binary operator or binary function, then it is not obvious what order the functions or operators are to be executed. For example, in the expression “6 + 8 % 3”, it is not obvious which computation is first; the addition or the modulus. The UNIDEX 600 Series controller CNC language uses the conventional precedence order of conventions during evaluation, briefly summarized below.

1. An expression in parentheses is evaluated by itself until everything within those parentheses fully reduces to a single constant.
2. All functions have precedence over operators.
3. All unary operators have precedence over binary operators.
4. When an operand is compressed between two binary operators, the operator with the higher precedence index computes first. However, if the operators are of the same precedence index, then the left operator is computed first.

Table 3-4 shows the operator precedence indexes.

Table 3-4. Operator Precedence Indexes

Binary Operator	Precedence
==	1 (lowest)
!=	1 (lowest)
<	1 (lowest)
>	1 (lowest)
<=	1 (lowest)
>=	1 (lowest)
+ (as in 8+8)	2
- (as in 8-8)	2
*	3
/	3
%	3
**	4
OR	2
AND	3
NOT	4
+ (as in +8)	5 (highest)
- (as in -8)	5 (highest)

EXAMPLES:

```
$GLOBO = 2 + 3 * 4      ; this is 14
$GLOBO = (2 + 3) * 4    ; this is 20
$GLOBO = -3**2          ; this is 9
```

3.8. Integer Expressions

SYNTAX: *<iExpression> is <integer> or <iOperatorExpression> or <fExpression>*
except : the *fExpression* must evaluate to an integral value (that is $\text{FRAC}(fExpression) = 0.0$)

EXAMPLE: **6+8**9**

Integer expressions are a subset of floating point expressions; used only in situations where a decimal value would be irrational or illegal.

3.8.1. Integer Constants

SYNTAX: *<integer> is [[- or +]]<digit>1*

EXAMPLE: **62**

0h3E ; 0h3E (Hex) = 62 decimal

0H3E ; 0h3E (Hex) = 62 decimal

An integer constant is a signed integer value. There can be no white space within the constant.

There is no limit to the number of digits in an integer, but there is a limited precision beyond which truncation of values occurs. The UNIDEX 600 Series motion controller supports a 4-byte (32-bit) value, where the values may range from +2,147,483,648 to -2,147,483,648. If a constant is entered outside this range, then the compiler delivers a warning message and the value is set to the appropriate limit.

3.8.1.1. Hexadecimal Numbers

Syntax: **0H<hexadecimal digit>1**

Example: **0Habsdef**

0habcdef

Hexadecimal numbers may be expressed by entering them with a leading 0H (zero H) before the hexadecimal number. For example, 0Habcde is a valid number in hexadecimal format.

3.8.2. Integer Operators

SYNTAX: $<iOperatorExpression> \text{ is } <iExpression>\sim<fOperator>\sim<iExpression>$

$<iOperator>$	is BOR or BAND or BNOT or BXOR
	or BNAND or BNOR or or &
	or ~ or ^ or >> or <<

EXAMPLE: **62 >> 2**

These are bitwise operators that work much like floating point binary operators. The operator expression is evaluated or converted into an integer constant when the controller executes the line containing that operator expression. Note that the arguments must be integer values.

The second line of symbols above are the ANSI C equivalents to the verbal equivalent directly above it. For example, “BOR” is equivalent to “|” (note that there are no equivalent verbal forms for the bit shift operators). Therefore, these operators (not the verbal forms) are fully compatible with ANSI C.

In some cases certain arguments are illegal. If an illegal argument is a constant, then the compiler delivers an error. However, if the argument is not a constant and is evaluated to an illegal argument when the line executes, the motion controller generates an appropriate fault while attempting to execute that line.

Table 3-5 shows the operators, their result, and any restrictions on their input arguments. The symbol α denotes the first argument and the symbol β denotes the second argument, if any.

Table 3-5. Summary of Integer Operators Available (Where α and β are Arguments)

Operator	Restrictions	Description of result
BOR	...	Bitwise or of α and β
BAND	...	Bitwise and of α and β
BNOT	...	Bitwise not (inversion) of α
BXOR	...	Bitwise exclusive or of α and β
BNAND	...	Logical NOT of a bitwise AND of α and β
BNOR	...	Logical NOT of a bitwise OR of α and β
$<<$	$\beta \geq 0$	Left shift α by β bits
$>>$	$\beta \geq 0$	Right shift α by β bits

The bitwise operators perform 32 independent operations, one on each corresponding pair of bits in the arguments. Table 3-6 summarizes the results of three common bit operations.

Table 3-6. Summary of Bitwise Operations

α Argument	β Argument	OR	AND	XOR	BNAND	BNOR
0	0	0	0	0	1	1
0	1	1	0	1	1	0
1	0	1	0	1	1	1
1	1	1	1	0	0	0

The examples below show legal syntax using the above operators, (0h prefixes a hexadecimal constant) and summarize their results in binary: (the comment following the code shows the binary description of the operation).

EXAMPLE:

<u>Hexadecimal Expression</u>	<u>Binary Expression</u>	<u>Result</u>
0hC BOR 0hA	= 1100 BOR 1010	= 1110 (0hE)
0hC BAND 0hA	= 1100 BAND 1010	= 1000 (0h8)
BNOT 0hA	= BNOT 1010	= 0101 (0h5)
0hC BXOR 0hA	= 1100 BXOR 1010	= 0110 (0h6)
0hC << 1	= LEFT SHIFT 01100	= 11000 (0h18)
0hC << 1	= RIGHT SHIFT 01100	= 00110 (0h6)

Note, that the BNAND and BNOR operators are combinations of logical and binary operators, for example:

BNAND \$glob0 is equivalent to NOT (BAND (\$glob0))

BNOR \$glob0 is equivalent to NOT (BOR (\$glob0))

3.9. String 32 Expressions

SYNTAX:

<s32Expression> is *<s32Constant>* or *<s32Variable>* or *<s32OperatorExpression>*

String 32 expressions are strings whose maximum length are 32 characters. Constants, variants, and operators are all available for this type (see Table 3-1).

3.9.1. String32 Constants

SYNTAX: *<sCharacter> is <printableCharacter> or <space> or <CR> or <LF>*
except “
except ‘
except -
<s32Const> is <sCharacter>1

String32 constants may contain up to 32 characters and nearly every type of printable character, plus spaces, carriage returns, and line feeds.

EXAMPLE: “this is a string”

3.9.2. String32 Variables

See Section 3.11 for examples of use and details on variable scope, initialization, and usage.

3.9.3. String 32 Operators

SYNTAX: *<s32OperatorExpression> is <s32Expression>~+~<s32Expression>*

The only string operator available is the concatenation operator, specified as the plus operator.

The user can use string constants and string variables interchangeably.



EXAMPLE: \$STRGLOB1 = “this”+“is”+“one” + \$STRGLOB0 + “string”

3.10. Labels

SYNTAX: *:<label> is <letter><letter>(<alphanumeric> or <period>)1*

EXAMPLE: :jumpToHere

Labels specify program locations. They must be at least three characters long and the first two characters must be letters. Underscores are legal after the second character.

Labels are used in **GOTO**, **DFS**, **FARJUMP**, **CLS**, and **FARCALL** extended commands.

3.11. Variants

Variants describe the collection of objects whose value may change during program execution. Variant values may change because the programmer changes them (i.e., variables) or because the controller changes them (i.e., the CLOCK parameter). Variants consist of variables, parameters, I/O elements, and call arguments.



The \$ symbol must precede all variant names, except for program variable aliases (see section 3.11.5.1).

3.11.1. Variant Types

SYNTAX: <variant> is <fVariant> or <s32Variant> or <APTVariant> or <integer> or <parameterVariant>

As discussed in Section 3.2.2., there are a number of expression types. However, in Table 3-7, only three of the expression types have variants available.

3.11.2. Variant Names

See Table 3-7 for a list of all variant names available and their expression type.



Most expression types do not have variant forms available.



The program variable does not have an explicit variant name; instead it is the expression element <pvName>. This is because the programmer assigns a name to program variables (see subsection 3.11.6).

It is important to add that the \U600\Programs\AerParam.Pgm include file provides more descriptive and easier to use forms of the parameter variants than those in Table 3-7. These forms are called “aliases” and Aerotech recommends using them instead of the parameter forms listed in Table 3-7. See Section 3.11.5 for more details on aliases.

Table 3-7. Variant Names

Variant Type									
Variables	CNCLetter	CNCMask	CNCWord	CNCBlock	Float	Integer	Labels	String32	
Global	APTGLOB	GLOB	STRGLOB	
Task	APTTASK	TASK	STRTASK	
Program	<pvName>	
Parameters	CNCLetter	CNCMask	CNCWord	CNCBlock	Float	Integer	Labels	String32	
Axis	PAXIS	
Task	PTASK	
Machine	PMACH	
Global	PGLOB	
I/O	CNCLetter	CNCMask	CNCWord	CNCBlock	Float	Integer	Labels	String32	
Binary In	BI	
Binary Out	BO	
Register In	RI	
Register Out	RO	
Analog In	AI	
Call Arguments	CNCLetter	CNCMask	CNCWord	CNCBlock	Float	Integer	Labels	String32	
Value	<argLetter>	
Existence	<argLetter> .DEFINED	

3.11.3. Assignments to Variants

SYNTAX: *<fAssignmentCommand> is <fVariant>~ =~<fExpression>*
<s32AssignmentCommand> is <s32Variant>~ =~<s32Expression>
<APTAAssignmentCommand> is <APTVariant>~ =~<APTEExpression>

EXAMPLES: \$GLOB7 = (6+\$GLOB0*2)
\$STRGLOB7 = "this is a string"+STRGLOB0
\$APTGLOB7 = X9.9 Y7 Z(\$GLOB0)

The assignment commands assign values to variants. The variant and the expression must be of the same type (i.e., floating point or string).

Certain variants are read only. All analog inputs, call argument existence variants, and certain parameters are read only (see *EDU157 UNIDEX 600 Series Users Guide*, under Parameters, to see which parameters are read only). If the user tries assigning values to

any of the read only parameters, a compiler error occurs. If the user tries assigning values to a read only parameter, the controller generates the appropriate fault.

3.11.4. Variables

SYNTAX: <variable> is <fVariable> or <s32Variable> or <APTVariable>

Variables are variants whose value is completely controlled by the user. Other than system initialization, the assignment statement is the only way to change these values.

3.11.4.1. Global Variables

SYNTAX: <fGlobVariable> is \$GLOB<integer> or \$GLOB[~<fExpression>~]
<s32Variable> is \$STRGLOB<integer> or \$STRGLOB[~<fExpression>~]
<fAPTVariable> is \$APTGLOB<integer> or \$APTGLOB[~<fExpression>~]

EXAMPLES: \$GLOB4 = 5

\$STRGLOB[4] = “dog”

\$APTGLOB[4+9] = X4 Y6.8

Global variables have universal scope, meaning they are accessible from any program on any task, and always refer to the same physical memory location.

The number of global variables available for use is defined by the *NumGlobalDoubles*, *NumGlobalStrings*, and *NumGlobalAxisPts* global parameters. If the user increases these numbers, then there is an allocation of new global variables.

The controller reset cycle initializes global floating-point variables. Initialized to zero are the floating-point global variables. Initialized to “” or the null string are String32 global variables. Finally, initialized to the NULL point are APT global variables.

The expression within the square brackets, if it exists, must be a non-negative integer. If it is not a positive constant integer, then the compiler generates an error. If it is an expression that does not reduce to a non-negative integer, then the controller generates a fault when it tries to execute the line containing the variable.



Global variables should not be used as target positions or feedrates in a contoured move, if the global variables are to be changed at other than at the program’s start, while CNC block look-ahead is active. CNC block look-ahead does not track global variables. Use program variables for this purpose.

3.11.4.1.1. Saving Global Variables to a File

The following subroutine may be used in your CNC program to save the values of global variables so that they may be quickly restored the next time the U600 MMI starts. The values are restored by executing a CNC program which sets the values of the variables.

```
DVAR $var

DFS Save_Vars ; save global vars 0-113 to ini file

$fHandle = FILEOPEN "Glob.Pgm", 0 ; over-write mode
$var = 0
REPEAT 114
    FILEWRITE $fHandle, "$GLOB[ \"$var \"] = ", $GLOB[ $var ]
    $var = $var + 1
ENDREPEAT
FILEWRITE $fHandle, "RETURN" ; TERMINATE PROGRAM (FOR
                             ; FARCALL)
FILECLOSE $fHandle

ENDDFS
```

3.11.4.1.2. Restoring Global Variables from a File

The following CNC program lines will restore the values of global variables previously written to a file. The variables values are restored by executing a CNC program which sets the values of the variables.

```
PROGRAMDOWNLOADFILE "C:\U600\Programs\Glob.Pgm"
FARCALL "Glob.Pgm" ; init. global vars
PROGRAMUNLOAD "Glob.Pgm" ; clean up memory
```

3.11.4.2. Task Variables

SYNTAX:

```
<fTaskVariable>    is $TASK<integer>      or $TASK[~<fExpression>~]  
<s32TaskVariable> is $STRTASK<integer>or $STRTASK[~<fExpression>~]  
<APTTTaskVariable> is $APPTTASK<integer> or $APPTTASK[~<fExpression>~]
```

EXAMPLES: \$TASK4 = 5

\$STRTASK[4] = “dog”
\$APPTASK[4+9] = X4 Y6.8

Task variables have task scope, meaning they are accessible from any program running on a given task. Task variables on different tasks are independent and have different values for each task. However, the user can access and set task variables running on another task.

The number of task variables available for use is defined by the *NumTaskDoubles*, *NumTaskStrings*, and *NumTaskAxisPts* task parameters. If the user increases these numbers, then there is an allocation of new global variables.

The number of task variables available for use is defined by the *NumTaskDoubles* global parameters. If the user increases these numbers, then there is an allocation of new global variables.

The controller reset cycle initializes global floating-point variables. Initialized to zero are the floating-point global variables. Initialized to “” or the null string are String32 global variables. Finally, initialized to the NULL point are APT global variables.

The expression within the square brackets, if it exists, must be a non-negative integer. If it is not a positive integer, then the compiler generates an error. If it is an expression that does not reduce to a non-negative integer, then the controller generates a fault when it tries to execute the line containing the variable.



Task variables should not be used as target positions or feedrates in a contoured move, if the task variables are to be changed at other than at the program’s start, while CNC block look-ahead is active. CNC block look-ahead does not track task variables. Use program variables for this purpose.

3.11.4.3. Program Variables

SYNTAX: <pvName> is \$<letter><letter><alphanumeric>₁**EXAMPLES:** \$myVariable[9]
\$myVariable[\$GLOB0+6]

The first two characters in a program variable name must be letters. The remaining characters can be letters, numbers or underscores. There must be at least three characters

in the name. In program variable names, like aliases, the case of the name is sensitive. For example, “\$myVariable” is a different variable than “\$MYVARIABLE.”

Program variables have program scope, meaning they are accessible from only the declaring program. They cannot be accessed from programs called by the declaring program. Program variables declared in the same program that is running on two different tasks, reference different variables. If a program calls itself, then the variables from the two instances of the program are independent and separate.

The user must declare program variables in a program. All program variable declarations must appear in the program before any non-declaration CNC lines. Program variables are declared with the **DVAR** extended command.

Declaration of program variables initializes them to zero.

3.11.4.4. Program Array Variables

SYNTAX: <programArrayVar> **is** <pvName>~[~<fExpression>~]

EXAMPLES: \$myVariable[9]
\$myVariable[\$GLOB0+6]

If the user uses an index expression with a program variable, then they are accessing an array element. To define arrays, use the **DVAR** extended command. The user can access any program variable with an index, even if it was not declared as an array, but the results may be unexpected. See the **DVAR** command for details on program variable indexing.

The expression within the brackets must be a non-negative integer. If it is not a positive constant integer, then the compiler generates an error. If it is an expression that does not reduce to a non-negative integer, then the controller generates a fault when it tries to execute the line containing the variable.

3.11.5. Parameters

SYNTAX: <parameter> is <axisParameter> or <taskParameter> or
<machineParameter> or <globalParameter>

Parameters are variants that directly affect the controller operation, or whose value is a direct result of the controller operation. In general, the user or the controller can change parameters. However, this depends on the specific parameter in question. Some parameters are only accessible by the programmer, others only accessible by the controller, and still others are accessible to both the programmer and the controller.

Virtually all controller activity is affected by parameters and an understanding of them is absolutely necessary. A detailed description of all parameters is in the *UNIDEX 600 Series Users Guide, P/N EDU157*.

3.11.5.1. Aliases

All parameter names are “aliased” for convenience and these aliases are available by including the standard include file \U600\Programs\AerParam.Pgm (see Chapter 4 for use of the include command). For example, “\$PAXIS[16]” can be alternately referred to as “KI”. The file \U600\Programs\AerParam.Pgm is always automatically included in user CNC programs by the program automation page of the UNIDEX U600 MMI, so normally the user would never use the parameter variant names listed below; instead they would use these provided aliases.

There are two major differences to be noted when using an alias as opposed to parameter names:

1. Unlike all other keywords, the case matters. The user must enter the alias in exactly the case shown in \U600\Programs\AerParam.Pgm.
2. Unlike all other variants, the user does not enter the dollar sign preceding the alias names.

3.11.5.2. Global Parameters

SYNTAX: <globalParameter> is \$PGLOB[~<fExpression>~]

EXAMPLES: \$PGLOB[2]

NumGlobalDoubles ; Note this is using an alias, see 3.11.5.1

Global parameters hold floating-point values used in the overall operation of the controller. For example, the servo loop time (1 or 4 kHz) and the emergency stop enabled state are global parameters. In some cases (such as emergency stop enable) only integer values are meaningful, but in general a global parameter is considered a floating-point value.

The square brackets are required, not optional as in variables. The expression within the square brackets must be a non-negative integer. If it is not a positive constant integer, the compiler generates an error. If it is an expression that does not reduce to a non-negative integer, then the controller generates a fault when it tries to execute the line containing the variable.



Global parameters have universal scope, meaning they are accessible from any program on any task.

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 parameter is significant, as defined in the User's Guide (EDU157) and the MMI online help file.

3.11.5.3. Task Parameters

SYNTAX: <taskNum> is 1 or 2 or 3 or 4
<taskParameter> is \$PTASK[~<fExpression>~)][.[<taskNum>]]

EXAMPLES: \$PAXIS[5].1
NumTaskDoubles.1 ; Note this is using an alias, see 3.11.5.1

Task parameters hold floating-point values used in the overall operation of a single task on the controller. For example, the MFO and Task Fault number are task parameters. In some cases (such as Task Fault Number) only integer values are meaningful, but in general a task parameter is considered a floating-point value.



The square brackets are required, not optional as in variables. The expression within the square brackets must be a non-negative integer. If it is not a positive constant integer, the compiler generates an error. If it is an expression that does not reduce to a non-negative integer, then the controller generates a fault when it tries to execute the line containing the variable.

The syntax for a task parameter allows a task number to be optionally included. If a task number is included, the parameter for that task is referenced (the controller may execute up to four simultaneous tasks). If a task number is not included, then the task parameter for the current task is referenced. Task parameters have universal scope, meaning they are accessible from any program on any task.

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. See the User's Guide (EDU157) and the MMI online help file.

3.11.5.4. Axis Parameters

SYNTAX: <axisParameter> is \$PAXIS[~<fExpression>~].<axisLetter>

EXAMPLES: \$PAXIS[16].X ; set X axis KI axis parameter
KI.X ; Note this is using an alias, see 3.11.5.1

Axis parameters hold integer values used in the overall operation of a single axis. For example, the Proportional Gain and Axis Position (in counts) are axis parameters. Axis parameters only hold integer values. Axis related parameters that are floating point are called machine parameters.

The programmer must supply an axis letter to indicate the axis that the parameter references.



The square brackets are required, not optional as in variables. The expression within the square brackets must be a non-negative integer. If it is not a positive constant integer, the compiler generates an error. If it is an expression that does not reduce to a non-negative integer, then the controller generates a fault when it tries to execute the line containing the variable.



Axis parameters have universal scope, meaning they are accessible from any program on any task.

Axis parameters are on a per axis basis, meaning, that each axis has its own independent set of parameters (as defined in the User's Guide (EDU157) and the MMI online help file). 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.

3.11.5.5. Machine Parameters

SYNTAX: <machineParameter> is \$PMACH[~<fExpression>~].<axisLetter>

EXAMPLES: \$PMACH[7].X

HOMETYPE.X ; Note this is using an alias, see 3.11.5.1

Machine parameters hold floating-point values used in the overall operation of a single axis. For example, the Maximum Feedrate and the Home Offset are machine parameters. In some cases (like Home Type) only integer values are meaningful, but in general a machine parameter is considered a floating-point value.



The programmer must supply an axis letter to indicate the axis that the axis parameter references.



The square brackets are required, not optional as in variables. The expression within the square brackets must be a non-negative integer. If it is not a positive constant integer, the compiler generates an error. If it is an expression that does not reduce to a non-negative integer, then the controller generates a fault when it tries to execute the line containing the variable.

Machine parameters have universal scope, meaning they are accessible from any program on any task.

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 (as defined in the User's Guide (EDU157) and the MMI online help file).

3.11.5.6. Modifying Parameters from within a CNC Program

3.11.5.6.1. Axis Parameters

An 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 the User's Guide (EDU157) and the MMI online help file.

The axis name is the name 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.

3.11.5.6.2. Machine Parameters

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 User's Guide (EDU157) and the MMI online help file.

The axis name is the name 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 = MaxFeedrateRPM.Y

3.11.5.6.3. Task Parameters

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 User's Guide (EDU157) and the MMI online help file. 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.

3.11.6. Virtual I/O

Virtual I/O (virtual input and output) variants report the state of virtual inputs and/or virtual outputs. If these virtual locations are tied or mapped to physical hardware locations (see *EDU154 UNIDEX 600 Series Hardware Manual*), then the corresponding virtual I/O reflects (or alters in the case of outputs) the state of the hardware mapped to them. Virtual outputs and inputs that are not mapped to hardware behave just like global variables; the user can set or read them at will. The exception being analog inputs, which maintain -10 volts.

In general, both inputs and outputs can be assigned or viewed, although assigning to a virtual input mapped to a hardware device is meaningless (the hardware device immediately overwrites the setting). Assigning values to virtual inputs not mapped to hardware can be done and is useful in some cases.

In I/O assignments (see Section 3.11.3) it is guaranteed that the preceding motion statement (excluding asynchronous motion) is completed before the assignment is done.

I/O has universal scope, meaning they are accessible from any program on any task.

3.11.6.1. Binary I/O Bits

SYNTAX: `<I/O> is $BI<integer> or $BI[~<fExpression>~] or
$BO<integer> or $BO[~<fExpression>~]`

EXAMPLES: `$BO4`

`$BI[8]`

`$BI[8+$GLOB0]`

Binaries are 1-bit integer values that range from 0 to 1. If the user tries to assign a value outside of this range to a binary I/O variant, the value is set to 0.

The expression within the square brackets must be a positive integer. If it is not a positive constant integer, the compiler generates an error. If it is an expression that does not reduce to a positive integer, then the controller generates a fault when it tries to execute the line containing the variable.

The “BI” prefix indicates a binary input; the “BO” prefix indicates a binary output. There are 512 virtual binary inputs and 512 virtual binary outputs available, so the index expression for a binary input or output must evaluate to an integer between 0 and 511. If the index expression evaluates to a value outside of this range, or to a non-integer, then the controller generates a fault.

By default, the first 16 binary inputs and outputs are mapped to the UNIDEX 600 card inputs and outputs. See the *EDU154 UNIDEX 600 Hardware Manual* for details. By default 40 output and 40 input bits are mapped to each encoder expansion (4EN-PC) card, if they are present in the user’s system.

The user can write or read binary inputs or outputs, but if a binary input is mapped to hardware, then it is overwritten immediately by the state of the physical hardware.

3.11.6.2. Virtual I/O Registers

SYNTAX: <I/ORegister> is \$RI<integer> or \$RI[~<fExpression>~] or
\$RO<integer> or \$RO[~<fExpression>~]

EXAMPLES: \$RO4

\$RI[8]

\$RI[8+\$GLOB0]

Registers contain unsigned 16-bit integer values that can range from 0 to 65535. If the user tries to assign a register a value larger than 65535, the register is set to a value of mod(α ,65536). If the user tries to assign to a register a value less than 0, the register is set to a value of mod(65536+ α ,65536). If the user tries to assign a non-integer value to a register, it truncates the value to the next lowest integer.

The expression within the square brackets must be a non-negative integer. If it is not a positive constant integer, the compiler generates an error. If it is an expression that does not reduce to a non-negative integer, then the controller generates a fault when it tries to execute the line containing the variable.

The “RI” prefix indicates a register input; the “RO” prefix indicates a register output. There are 128 register inputs and 128 register outputs available, so the index expression for a binary input or output must evaluate to an integer between 0 and 127. If the index expression evaluates to a value outside of this range, or a non-integer, then the controller generates a fault.

By default, no virtual registers are mapped to any hardware.

The user can write or read register inputs or outputs, but if a register input is mapped to hardware, then it would be overwritten immediately by the state of the physical hardware, if present.

All binary and register assignment commands will wait until all axes in previous motion commands are “in-position”.



3.11.6.3. Analog Inputs

SYNTAX: <I/OAnalog> is \$AI<integer> or \$AI[~<fExpression>~]

EXAMPLES: \$AI[8]

\$AI[8+\$GLOB0]

Analog inputs are returned as signed floating-point values that can range from -10 to 10 volts. They are read-only, meaning the user cannot assign to an analog value, they can only read it.

The expression within the square brackets must be a non-negative integer. If it is not a positive constant integer, the compiler generates an error. If it is an expression that does

not reduce to a non-negative integer, then the controller generates a fault when it tries to execute the line containing the variable.

Potentially, there are 16 available analog inputs (4 are standard on a UNIDEX 600 card, with 4 more available on each additional 4EN-PC expansion card), so the index expression for a binary input or output must evaluate to an integer between 0 and 15. If the index expression evaluates to a value outside of this range, the controller generates a fault. If the index expression evaluates to a non-integer value, then the controller generates a fault.



The user only has four analog inputs in the UNIDEX 600 Series axis processor card and four analog inputs on each encoder expansion card (4EN-PC) in the system. If the user tries to read an analog input that does not exist in the system, analog input 15 for example, when there is only one encoder expansion card, then the value returned is -10 (volts).

3.11.7. Call Arguments

SYNTAX: *<callArgument>* is \$<argLetter> [<fExpression>]

EXAMPLE: \$p4.0
 \$X[8+\$GLOB0]
 \$x

This section details passing and handling parameters in relation to subroutines and programs. The details concerning parameter passing apply equally to the **FARCALL** statement and the **CALL** statement. However, you cannot use call parameters in an **ONGOSUB** statement.

A subroutine may have up to 20 parameter values passed to it; each identified by the sixteen axis names and four additional letters (O, P, Q, and R). These parameter names may be reassigned for unused axes via the axis configuration wizard within the MMI 600. See the MMI help file for more information. Unlike the axis name parameters, these last four names are not case sensitive (i.e., x is a different parameter than X, but p and P are the same parameter). Using the syntax example: "X78 Y\$GLOB8", the X parameter value is 78, and the Y parameter is the value of GLOBAL variable 8, at the time the subroutine is called.

Call arguments may be assigned floating point values. However, parameters are passed by value (the value is not passed back to the program that called the program assigning to the call argument).

The value of a parameter within a called subroutine is referred to by a dollar sign followed by the parameter name. Each of the 20 parameters are optional, meaning, the same subroutine may be called with no arguments, all 20 arguments, or any number in between. If there is no argument specified when the subroutine is called, then its value as referenced within the called subroutine is 0.

The user should test each parameter to see if a value for that parameter was passed into the subroutine. The DEFINED keyword serves this purpose. The example clarifies this.

3.11.7.1. Call Argument Existence Testing

SYNTAX: *<callArgumentExist> is \$<argLetter>.DEFINED*

EXAMPLE: **IF \$X.DEFINED == 1 THEN**

\$GLOB1 = \$X

ENDIF

Any number of parameter values may be passed into a subroutine or program. The .DEFINED keyword is used to determine whether a particular parameter was assigned a value when the subroutine or program was called. The .DEFINED keyword returns one, if the specified call argument was assigned a value from the program/subroutine call argument list when the program/subroutine was called. Otherwise, it returns zero. See the Example Program for clarification.

Call argument existence variants cannot be assigned to (the syntax “\$X.DEFINED=1” is illegal).

If a parameter is referenced that was not passed into a program or subroutine, then a "Call Stack parameter not passed" error Message is generated.

EXAMPLE PROGRAM

```

CALL PRODUCT          ; Will set $GLOBAL0 to 1
CALL PRODUCT X3       ; Will set $GLOBAL0 to 18

CALL PRODUCT X3 p2 Y6 ; Will set $GLOBAL0 to 36
M02

DFS PRODUCT
$GLOBAL0 =1

IF ($X.DEFINED == 1) THEN
    $GLOBAL0 = $GLOBAL0 * $X
    ; NOTE: If you use $X when it hasn't been passed, a "Callstack
    ; parameter not passed" task fault is generated.
ENDIF
IF ($Y.DEFINED == 1) THEN
    $GLOBAL0 = $GLOBAL0 * $Y
ENDIF
IF ($Z.DEFINED == 1) THEN
    $GLOBAL0 = $GLOBAL0 * $Z
ENDIF
IF ($U.DEFINED == 1) THEN
    $GLOBAL0 = $GLOBAL0 * $U
ENDIF
IF ($C.DEFINED == 1) THEN
    $GLOBAL0 = $GLOBAL0 * $C
ENDIF
IF ($D.DEFINED == 1) THEN
    $GLOBAL0 = $GLOBAL0 * $D
ENDIF
IF ($A.DEFINED == 1) THEN
    $GLOBAL0 = $GLOBAL0 * $A
ENDIF
IF ($B.DEFINED == 1) THEN
    $GLOBAL0 = $GLOBAL0 * $B
ENDIF
IF ($X.DEFINED == 1) THEN
    $GLOBAL0 = $GLOBAL0 * $x
ENDIF
IF ($Y.DEFINED == 1) THEN
    $GLOBAL0 = $GLOBAL0 * $y
ENDIF
IF ($Z.DEFINED == 1) THEN
    $GLOBAL0 = $GLOBAL0 * $z
ENDIF
IF ($U.DEFINED == 1) THEN
    $GLOBAL0 = $GLOBAL0 * $u
ENDIF
IF ($C.DEFINED == 1) THEN
    $GLOBAL0 = $GLOBAL0 * $c
ENDIF
IF ($d.DEFINED == 1) THEN
    $GLOBAL0 = $GLOBAL0 * $d
ENDIF
IF ($a.DEFINED == 1) THEN
    $GLOBAL0 = $GLOBAL0 * $a
ENDIF
IF ($p.DEFINED == 1) THEN
    $GLOBAL0 = $GLOBAL0 * $p
ENDIF
ENDDFS

```

▽ ▽ ▽

CHAPTER 4: COMPILER DIRECTIVE COMMANDS

In This Section:	Page
• Overview.....	4-1
• Define Statements.....	4-2
• Include Statement.....	4-6
• AxisNames Statement	4-7

4.1. Overview

This chapter describes the compiler directive commands. Compiler directives are instructions to the compiler, instructing it how to interpret the program text. They include or replace text in the program before compiling it. They are especially useful for accessing parameters and implementing user defined M-codes.

The preprocessor (the part of the compiler that executes directives) executes all compiler directives in the program text before handing off the program to the actual compiler. Refer to Figure 4-1.

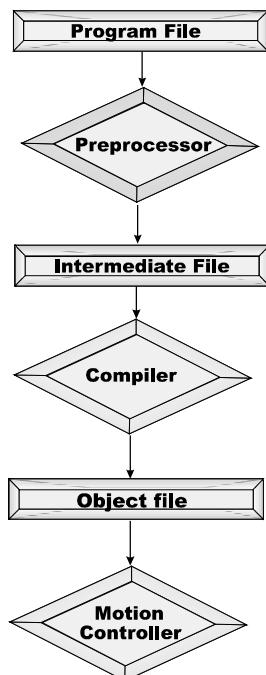


Figure 4-1. Flow of Execution of Compiler Directives

4.1.1. Compiler Directives Syntax

The user has the option to save either the intermediate or the object files to disk. Refer to the UNIDEX 600 Series Library Reference Manual P/N EDU156, under AerCompilerCompile() file, for more details.

SYNTAX: <Compiler-directive line> is <defineStatement> or <includeStatement>

The above states that compiler directives must be either include or define statements.

The syntax and semantics of UNIDEX 600 Series controller CNC language preprocessor directives are very similar, but not identical to the ANSI C language standard. Summarized below are the differences in order of their importance.

- The UNIDEX 600 does not have conditional preprocessor directives (#ifdef and #endif).
- The UNIDEX 600 does not allow # or ## operators in replacement strings.
- Multi-line substitution syntax rules differ slightly.
- Substitution within preprocessor directive text rules differ slightly.
- The UNIDEX 600 does not require double quotes around filenames.

4.2. Define Statements

SYNTAX: <defineStatement> is ~#**define** <targetWord> [<replacementString>]

The **#define** statement directs the compiler to replace all occurrences of the target word in the program with the replacement string. This directive is only active for all lines following the define statement. Occurrences of the target-word above (on previous CNC lines) will not be replaced.

The **#define** statement is powerful, since the replacement string can be nearly anything, including multiple lines. This means a single word in a program can represent multiple lines. Refer to the description in Section 4.2.4. Replacement with Multiple Lines.

The “#define” must be separated from the target word by white space and the target word must be separated from the replacement string by white space as well. The “#” sign may be preceded by white space.

#define directives can be overridden or deactivated at any time. If a **#define** statement has no replacement string argument, it is considered an “undefined”. It deactivates any previous define instruction having the same target word. After an “undefined”, no replacements of the target word are made.

If the user undefined using a target word that is not currently active or if the user redefines a target word already active, the compiler delivers a warning.

EXAMPLES:

```
#define LaserState $BI7      ; After this statement, "BI7" will be replaced  
                             ; with "LaserState"  
#define $BI7                 ; After this statement, "BI7" will no longer be  
                             ; replaced.
```

4.2.1. The Target Word

SYNTAX: <targetWord> is <alphanumeric>₁
except #~**define**
except #~**include**

There is no limit to the allowable length of a target word.

The target word consists of characters, letters, and the underscore. Target words are not keywords. Identification of the target word is case sensitive. For example, the target words: “DOG”, “dog”, and “Dog” are all legal and represent different targets.

There are two exceptions, the define and include command keywords are not allowed as target words.



4.2.2. Recognition of the Target Word

Substitution of the replacement text only takes place at locations where there is recognition of the target word. Recognition of the target will not take place within comments. Recognition of the target word in the text only occurs if the two characters on either side of the word are not alphanumeric characters. For example, if the statement “#define dog spot” is made, then the following text conversions take place.

“my dog is friendly”	becomes	“my spot is friendly”
“the dogs are friendly”	becomes	“the dogs are friendly”
“7*dog[6]”	becomes	“7*spot[6]”

Recognition of the target word only occurs on lines following the define referencing the target word. Lines appearing in the file before that define, or appearing in files included before that define, will not be searched for the target word referenced in that define. The same comment applies to “undefines”. Refer to the example below.

```
$GLOB0 = CONSTANT          ; no substitution of CONSTANT done here  
#define CONSTANT 2           ; this defines CONSTANT  
$GLOB0 = CONSTANT          ; substitution of CONSTANT with 2 is done here  
#define CONSTANT            ; this “undefines” CONSTANT  
$GLOB0 = CONSTANT          ; no substitution of CONSTANT done here
```

becomes:

```
$GLOB0 = CONSTANT  
$GLOB0 = 2  
$GLOB0 = CONSTANT
```

Replacement of target words within a replacement string (recursive substitution) occurs under some circumstances (see Section 4.2.5. for details).

4.2.3. The Replacement String

SYNTAX: <replaceCharacter> is <ASCIICharacter>
or / x <Hexdigit> <hexdigit>
except \
except ;
except <lineTerminator>
<replacementString> is <replaceCharacter>₁
except #~define
except #~include

Unlike the target word, a replacement string allows almost every other character, including spaces, tabs and quotes. However, the comment character ‘;’ and the ‘\’ are not allowed in the replacement string. Also, the replacement string cannot be the line terminator or the #define and #include keywords.

There is no limit to the allowable length of a replacement string.

The line terminator “;” and “\” characters end the replacement text. More definitively, the last non-white-space character preceding the first “;” or “\” or line terminator found is the last character in the replacement string. This allows the programmer to comment the define lines.

```
#define dog spot ; this comment is not replaced into the text
$STRGLOB1 = "dog" ; this comment is retained
```

becomes:

```
$STRGLOB1 = "spot" ; this comment is retained
```

Replacement characters may also be specified as hexadecimal codes, for example, the replacement string “12” is equivalent to “\x31\x32”. This is especially useful for adding spaces at the end of replacement strings, since trailing spaces are normally removed from replacement strings (see EXAMPLE 3 of Chapter 7: Custom Commands).

Replacement of target words within a replacement string (recursive substitution) occurs under some circumstances (see Section 4.2.5. for details).

4.2.4. Replacement with Multiple Lines

The “\” character is not allowed in a replacement string because it is reserved as the line terminator character. If the define directive line sees this character “\”, then the preprocessor appends a line terminator to the replacement string and continues looking for more replacement string text on the next line. All text following the “\” on that same line, if any, is ignored. Refer to the example below.

```
#define M1000\      ; random comment 1  
BI[0]=0\          ; random comment 2  
BI[1]=1          ; random comment 3  
M1000            ; random comment 4
```

becomes:

```
BI[0]=0  
BI[1]=1
```

The comments for the lines within the replacement string are not preserved in the text.



4.2.5. Replacement within Replacement Strings

Text within a replacement string may contain target words established in earlier defines, and normally replaces the target word. This new replacement string may contain other target words that will be replaced. This complicated process is called recursive substitution.

However, once making a particular replacement(s) within a replacement string, that replacement is not made again in that string. Refer to the example below.

```
#define END_OF_TRAVEL    10.0  
#define X_END           $x(END_OF_TRAVEL)  
#define BAD_SYNTAX      (10.0 - BAD_SYNTAX)  
#define Y_END           $Y(BAD_SYNTAX)  
  
G1 X_END Y_END
```

becomes:

```
G1 x$(10.0) $Y((10.0 - BAD_SYNTAX))
```

4.3. Include Statement

SYNTAX: <Include-statement> is ~#~**include** [[“]]<filename>[[“]]

The preprocessor inserts the entire text of the file named into the program. The text inserted receives the identical treatment as the actual text in the program. This means that any preprocessor directives found in the included text will be obeyed. For example, the user can include files within included files. There is no limit to the ‘nesting’ level of includes. The included file will be searched for in the current program directory, this is \U600\Programs by default.



The user can surround the filename with double quotes, since this has no effect on the function of the include statement.

4.3.1. Filenames

SYNTAX: <filename> is <filespecletter>₁

<filespecletter> is <alphanumeric> or \ or [or] or :

In addition to the above rules, a filename must represent a valid Windows NT filename. Filenames can be up to 32 characters long. The user can place valid directory specifications as a filename, including paths. If the user does not provide a path, it will use the default path. The default path is the working directory for the first included file seen in the program. However, for all subsequent includes, the default path is the directory where the first included file was found. In other words, if a path is found in the first file included, then the default path changes to the path of the first file included.

The filename will not be searched for replacements from defines, meaning the following program text will not include the file “okFile.pgm”, it will try to include the file MY_FILE.

```
#define MY_FILE okFile.pgm  
#include MY_FILE
```

EXAMPLES:

```
#include afile.txt  
#INCLUDE ..\anotherfile.pgm  
#include "e:\u600\aecmplr>this_file.pgm"
```

4.3.2. Standard Include Files

The ‘\U600\Programs\AerParam.Pgm’ file is always automatically included within the program automation page of the setup page of the UNIDEX 600 MMI. The program automation page will not indicate this file is auto included, but it is and cannot be removed. This file contains aliases for parameters and certain constants. Without this file the programmer cannot access parameters with their names.

4.4. AxisNames Statement

SYNTAX: <axisNameStatement> is ~#~axisnames <axisname>₁

<axisName> is (<letter> or _)

except : certain one letter axisnames are illegal.

EXAMPLE: #axisnames Xaxis, Yaxis, Spindle

The axisnames statement defines alternatives to the axis letters when referring to axes. An axis name can be any string of letters and underscores excluding digits in the name. Names are case sensitive, meaning, ‘XAXIS’ is a different axis compared to ‘xaxis’. Also, many one letter names are illegal, since they can be confused with command letters. The only legal one letter axis names are: X,Y,Z,U,V,W,A,B,C,D,L, and x, y, z, u, v, w, a, b, c, d, and l.

The position of an axis name in the axisnames statement corresponds to the axis whose name is replaced. In the above example, the name ‘Xaxis’ is the new name for ‘X’, and ‘Spindle’ is the new name for the ‘Z’ axis. The default set of axis names are: ‘X, Y, Z, U, A, B, C, D, x, y, z, u, a, b, c, d’. The user can rename from one to sixteen axes, but if the user desires to rename only one axis, the user must also rename axis names for all positions up to the position of that axis. For example, to rename only the ‘D’ axis to ‘newaxis’ the user must use the statement: ‘#axisnames X,Y,Z,U,A,B,C,newaxis’. Once an axis is renamed, the old name cannot be used; the new name must be used.

These new names can be used equivalently to the axis letters in axis masks, and parameter specifications: (‘BIND X, newaxis,Z’ and ‘DRIVE.newaxis=1’ are fine). However, when the axisname has more than one letter, special considerations are necessary when using the axisname in an axis point with constants (‘X6 Y7’). The user must separate a multi-letter axis name from the constant value with a space, =, or parenthesis. See the example on the following page.

There can be one unique axisnames statement per program. The position of the axisnames statement in the program is irrelevant; the statement always applies to the whole file regardless of its placement in the file.

EXAMPLE:

```
#axisnames X,SPINDLE
G1 X7.8          ; OK
G1 X$glob9       ; OK
G1 SPINDLE7.8    ; INVALID SYNTAX
G1 SPINDLE 7.8   ; OK
G1 SPINDLE=7.8   ; OK
G1 SPINDLE(7.18) ; OK
G1 SPINDLE $glob9 ; OK
```

▽ ▽ ▽

CHAPTER 5: G-CODE COMMANDS

In This Section:	Page
• Introduction.....	5-2
• CNC Block Syntax	5-13
• Non-Contoured Motion (G0)	5-23
• Contoured Motion (G1, G2, G3).....	5-24
• Dwell (G4)	5-35
• Velocity Blending (G8, G9, G108, G109)	5-36
• Contoured Motion on Coordinate System # 2 (G12, G13).....	5-41
• Coordinate System #1 Configuration (G16 – G19).....	5-43
• Normalcy Motion Overview (G20, G21, G22)	5-45
• Corner Rounding (G23, G24)	5-50
• Coordinate System #2 Configuration (G26 – G29).....	5-51
• Software Limits Overview	5-52
• Safe Zones (G34, G35, G36, G37)	5-52
• Backlash Compensation (G38, G39).....	5-57
• Cutter Radius Compensation (G40, G41, G42, G43, G45).....	5-58
• Polar/Cylindrical Transformations (G45, G46, G47).....	5-68
• Fixture Offsets (G53 – G59)	5-74
• Contoured Accel/Decel Overview (G60, G61)	5-81
• Profile Resolution Time (G62).....	5-84
• Accel/Decel Rates and Modes (G63 -> G68).....	5-84
• Metric/English Units (G70, G71).....	5-89
• Restore Preset Position Registers.....	5-90
• Transformation Overview (G83, G84).....	5-91
• Positioning Modes (G90, G91)	5-95
• Preset Positions (G92).....	5-97
• Feedrate Modes (G93, G94, G95).....	5-99
• Dominant Feedrate Overview (G98, G99)	5-105
• Spindle Shutdown Modes (G100, G101)	5-108
• Modal Velocity Profiling (G108, G109)	5-109
• Circular Direction Codes (G110, G111)	5-110
• Block Delete Mode (G112, G113).....	5-112
• Optional Stop Mode (G114, G115).....	5-113
• Dry Run Mode (G116, G117)	5-113
• Servo Update Rate (G130, G131)	5-114
• Cutter Tool Offset Compensation Overview (G143, G144, G149).....	5-115
• Scale Factor (G150, G151)	5-117
• Suspend All Fixture Offsets	5-119
• Rotary Axis Acceleration Rates (G165, G166).....	5-120
• Block Delete2 Mode (G212, G213).....	5-121
• CNC Block Look-Ahead (G300, G301)	5-122
• High Speed Machining (G310, G311)	5-126
• M-codes	5-128

5.1. Introduction

This chapter describes the syntax and functionality of all G-code lines (also called G-code blocks). Refer to Chapter 2: Commands for a basic description of CNC programs.

The term G-code used in this document refers to more than just CNC words starting with a “G” (i.e., G90). The term G-code refers to all the syntax defined in RS-274D including G-codes, M-codes, F-codes, etc. For example, the line N07 G2 X5 Y6 I5 J7 F30 is legal G-code syntax.

However, this chapter does not include any information on I/O M-codes. Only the data on the RS-274 defined M-codes such as M0, M1 (program control) and M3, M4 (spindle control) can be found in this chapter. Refer to Chapter 7: Custom Commands for details on the I/O M-codes.



A number of basic CNC language elements used in G-codes are not described here. Instead the descriptions are in Chapter 3: Expressions. Table 5-1 explains where to find descriptions of these items.

Table 5-1. Where to Find Details

Term	Example 2	Example 2	Reference
<Expression>	7*6+\$GLOB0	7.8	Section 3.7
<CNCMask>	X Y z	x	Section 3.4
<axisPoint>	X55.6 Z5	Z\$GLOB0	Section 3.6.1.2
Command Words (start with G or M)	G1	M2	See Table 5-3
All other Words (start with F,E,I,J,...)	F100	I8.9	See Section 5.2.
Motion Details	n/a	n/a	See Section 5.1.1

The reader should note that there are a number of details that concern all G-code motion. These concern the position, velocity, and acceleration of all G-code moves, as well as prerequisites for moving an axis. Refer to Section 5.1.1 for details. We strongly recommend that the programmer become familiar with these before using any motion G-codes.

Table 5-3 alphabetically lists all the G-code commands available to the UNIDEX 600 Series controller CNC language and where to find information on that G-code. The sections covering the G-codes and M-codes follow the same convention within this chapter.

5.1.1. Motion Types Available

There are two types of motion that can be generated from CNC language commands: synchronous and asynchronous.

When executing synchronous motion CNC commands, the controller does not move to the next CNC command in the program until the motion finishes and the axes are in position. The user can move any number of axes (up to 16) at once in a synchronous motion command. Synchronous motion commands include **G0**, **G1**, **G2** and **HOME**.

In asynchronous motion, the motion is initiated, but the controller immediately moves to the next CNC command in the program. The controller does not wait for the motion to complete. All asynchronous motion commands can only move one axis at a time.

Asynchronous motion offers more versatility, allowing the user to perform other tasks during a time consuming move. However, asynchronous moves potentially, are more dangerous, since the programmer is responsible for making sure the move finishes if an ensuing command requires the move to already be in position. This verification can easily be done by monitoring the *status3* task parameter. For example, the two code fragments below are equivalent.

HOME X Y	; Homes X and Y axis synchronously
----------	------------------------------------

HOMEASYNC X HOMEASYNC Y :waithere if (Status3.MotionActive EQ 1) goto waithere	; This code block imitates a HOME XY
-----------------------------------------------------------------------------------------	--------------------------------------

If an asynchronous move is not complete and the program attempts to move an axis (either with a synchronous or asynchronous motion), then the CNC stops and waits until the first motion completed, before starting the second move.

5.1.2. Motion Commands Available

Table 5-1 shows the available CNC move options. Note that some commands can only move one axis at a time.

Table 5-2. CNC Move Options

CNC Command	Sync Type	Multi Axis	Example	Description
G0	Sync	Y	G0 X5Y5	Non-contoured linear motion
G1	Sync	Y	G1 X5Y5 F100	Contoured linear motion
G2	Sync	Y	G2 Y5X5 I4J5 F100	Contoured circular motion
HOME	Sync	Y	HOME X Y	Homes axes
MOVETO	Async	N	MOVETO X 5 100	Move X to 5 (absolute coord) at 100 speed
INDEX	Async	N	INDEX X 5 100	Move X 5 (relative coord) at 100 speed
STRM	Async	N	STRM X 1 100	Starts motion (freerun) at 100 speed, in + direction
ENDM	Async	N	ENDM X	Ends freerun motion.
FEDM	Async	N	FEDM X 5 100	Feeds in an axis 5 at speed 100
OSC	Async	N	OSC X 5 100	Start oscillations of amplitude 5 at 100 speed
HOMEASYNC	Async	N	HOMEASYNC X	Homes axis asynchronously

There are a few other miscellaneous ways to indirectly generate motion. Like cutter compensation (**G21**) and normalcy (**G41**), but these are not commonly used, and too complex to summarize here.

5.1.3. Prerequisites for Initiating Motion from the CNC

The U600 offers enormous flexibility in motion; unfortunately this means the user must choose between many options. Before any movement on an axis, including CNC moves, at minimum, accomplish the following on the axes involved in the move.



Only step 1 cannot be performed from the CNC language.

1. Configure the axes. Refer to *EDU157 UNIDEX 600 Series User's Guide*, Getting Started/Axis Configuration in Chapter 2.
2. Set the required Axis/Machine parameters. Refer to *EDU157 UNIDEX 600 Series User's Guide*, Getting Started/Axis parameters and Machine parameters in Chapter 2.

This includes at a minimum:

- a. Gains (*KP*, *KI*, etc. axis parameters)
 - b. *Type* machine parameter (rotary .vs. linear)
 - c. *CountsPerInch* or *CountsPerDeg* machine parameters.
3. Enable the axes. (Refer to the **ENABLE** command)
 4. Set the axes' fault masks. Refer to *EDU157, UNIDEX 600 Series User's Guide*, under Faults in Chapter 2.

There is no strict requirement to follow step 4, since the fault masks are set to -1 (abort motion on any fault). However, since there are a number of conditions, called faults, that might occur during motion and may or may not require special handling, we strongly recommend the user refer to *EDU157, UNIDEX 600 Series Users Guide*, under Faults in order to understand fault handling on the U600.

5.1.4. Command vs. Actual

The controller directs the motors, via the drives, to their commanded positions at specific velocities. The actual position and velocity is read from the feedback devices (encoders/resolvers). The error values are the actual values minus the command values. The function of the servo loop, is to attempt to make the actual values equal the command values. In other words, make the error values zero.

However, there are certain modes where the error values do not converge to zero; in offsets (**G92**), mirroring (**G83**), and parts rotation (**G85**). The coordinates provided by the user specifies their transformation before usage. The command values are always what the user supplies, however, the actual values may converge to other values if the user

activates one or more of the above modes. For example, if the user moves to X=5, sets this as the origin (**G92 X0**), then after a **G90 G1 X9**, the command value will display 9, while the actual value displays 14.

5.1.5. Target Positions

The user can specify target positions relative to the current position or relative to a fixed origin (absolute). Refer to the particular command for details on the available mode.

If the axis is defined as a rotary axis with modulo position and moving in a contoured or rapid move, then the controller moves to the target position in the direction of least distance. For example, if B is rotary and at 0 degrees, then a move to 270 degrees causes a counterclockwise rotation of 90 degrees, not a clockwise rotation of 270 degrees. If the amount to be moved is exactly 180 degrees, then the motion is clockwise.

There are some minor considerations due to the fact that the user specifies floating point numbers, but the actual positions (counts) are integers. Normally, these are insignificant because the errors are always less than a count. However, it does mean that the floating point numbers reported as positions/velocities will not exactly match the floating point values specified by the user. The following information provided is for those interested in knowing the details of the floating point to counts conversion. First of all, the controller always truncates position values. This means that the move may be as much as one count short of the desired position. For example, if there are 100 counts per foot and the user specifies a move of 1 inch, the controller moves 8 counts, not 8.333 counts. Also, velocities truncate to the nearest user-unit (user-units can be either mm or inches) per second. Due to truncation, the controller may not be able to satisfy the acceleration, the truncated velocities, and the distance exactly, since the move velocity and distance were computed correctly in floating point. What the controller does, is satisfy both the truncated distance and velocity exactly and make up for any mismatch in the decel phase of the move. Therefore, the slope of the velocity during decel may vary slightly from the expected result. A final consideration involves profiled or blended moves, (Refer to Chapter 5: G-code Commands) that do not have a decel phase. Here the velocity during the constant phases is reduced, so that the truncated distance is satisfied.

5.1.6. Simultaneous Movement of Multiple Axes

The CNC programmer can specify movement of multiple axes simultaneously in a G-code move. The controller, then moves all the specified axes to the specific targets. However, there are some subtle points concerning the velocities and the accomplishments of these moves.

A **G0** command is a rapid move, while **G1/G2/G3/G12/G13** are contoured moves. A **G0** moves each axis at that axis' rapid feedrate (found under the Machine Parameters Screen), and no effort is made to coordinate the separate axis' motion. Each axis finishes its move at different times and the **G0** is not complete until the last axis completes its motion.

G1, G2, G3, G12, and G13 are contoured moves. The axes move in a coordinated fashion so that all axes finish their motion at the same time. The E and F words determine the speed of the motion. However, the way the E and F word determine these speeds can

be complex if the user moves rotary and linear axes simultaneously. Refer to the **G98** command for details.

G1 is a linear move, while **G2** and **G3** are circular moves. The user can specify a linear and circular move to occur simultaneously by putting a **G1** and a **G2/G3** on the same line. Also, the user can perform two circular moves simultaneously by using **G12** and **G13**. These G-codes are the same as the **G2/G3** counterparts for simultaneous motion. For example, the following code performs two circular moves simultaneously.

```
G2 I1 J1 X0 Y0 G13 I3 J3 Z0 A0
```

All CNC G-code motion statements wait until there motion is complete before proceeding to the next CNC statement. These CNC statements are called synchronous. The controller does provide asynchronous CNC motion statements that initiate a motion, but then immediately continue to the next CNC statement (see **INDEX** or **STARTM** extended commands). If such a statement executes, a following CNC statement that tries to execute a move on an axis still moving due to the first asynchronous motion statement, will wait until the first motion is complete before executing.

5.1.7. Velocity

For **G0s**, the machine parameter *RapidFeedrateIPM* specifies the speed of each axis in the move. Since the **G0** is not a contoured move, each axis acts independently and can achieve its own rapid feedrate.

For contoured moves (**G1**, **G2**, **G3**, **G12**, or **G13**) the F and E words determine the vector feedrate (see chapter 5, under Rate Words). This vector feedrate will be the speed of the motion along the path described by the motion in space. However, for contoured moves, no axis is allowed to exceed its individual maximum speed value (machine parameter *MaxFeedrateIPM* or machine parameter *MaxFeedRateRPM* for rotary axes). The feedrate of the entire move automatically scales down until no axis in the motion exceeds its own maximum feedrate.

If the axes in a move are all linear, then the F word determines the vector feedrate. If all the axes are rotary, then the E word determines the vector feedrate. However, if a move contains axes of both rotary and linear type, then the controller probably will not be able to achieve both the E and F feedrates at once. In these cases the controller picks one and ignores the other. The feedrate it picks is called the dominant feedrate. The dominate feedrate in a given move can be a complex subject. Refer to the **G98** command for details.

Asynchronous moves direct one axis at a time, so there is no speed problem. It simply uses the speed provided in the asynchronous command parameter. Asynchronous moves are not limited by the maximum feedrate machine parameter.

5.1.8. Acceleration/Deceleration

The acceleration rates of both **G0** moves and asynchronous moves are determined by the axis parameters: *ACCELMODE*, *ACCEL*, and *ACCELERATE*. Similarly, the deceleration rates of **G0** moves and asynchronous moves are determined by the axis parameters: *DECACELMODE*, *DECACEL*, and *DECACELRATE*. Refer to Appendix C: Parameters in the *UNIDEX 600 Series User's Guide P/N EDU157* for details.

The acceleration and deceleration rates of contoured synchronous moves (**G1**, **G2**, **G3**, **G12**, **G13**) is set by the G-codes **G60** through **G68** (some of these G-codes are equivalent to setting task parameters). There are many options for setting the acceleration and deceleration rates for contoured moves, refer to the **G60-G61** commands for details.

Also, there are some codes that determine whether acceleration and deceleration will take place in between two contoured moves; **G8**, **G9**, **G108** and **G109**. Refer to these G-codes for details.

There are some situations of importance, when the controller is forced to command an instantaneous or infinite acceleration or deceleration. These situations are fully described under the **G8** command.



5.1.9. Further Information

Further elaboration on the following topics of interest can be found discussed under the documentation section shown to the right of the topic.

Position	(see G90/G91 and G70/G71)
Velocity	(see F and E codes)
Acceleration	(see G60 or G61)
Transition between contoured motion blocks	(see G8 , G9 , G108 , and G109)
Mixing of rotary and linear type axes	(see G98 or G99)

5.1.10. Modal

Many G-code words define modes of operation that exist until changed by another G code, these are Modal G codes. Otherwise, the G code affects only the current CNC program line.



5.1.11. Default

During initialization, the UNIDEX 600 Series controller defines the initial modes as shown in the fourth column of the table below. Some of these default modes may be redefined within the task initialization page of the MMI600. These are shown in the fifth column of Table 5-3.

Table 5-3. G-code and M-code Summary

G-code	Page	Description	Modal	Default
a	n/a	Axis Coordinate (see chapter 3)	n/a	n/a
A	n/a	Axis Coordinate (see chapter 3)	n/a	n/a
b	n/a	Axis Coordinate (see chapter 3)	n/a	n/a
B	n/a	Axis Coordinate (see chapter 3)	n/a	n/a
c	n/a	Axis Coordinate (see chapter 3)	n/a	n/a
C	n/a	Axis Coordinate (see chapter 3)	n/a	n/a
d	n/a	Axis Coordinate (see chapter 3)	n/a	n/a
D	n/a	Axis Coordinate (see chapter 3)	n/a	n/a
E	n/a	Rotary Feedrate Specifier (see section 5.2.3.2)	Y	n/a
F	n/a	Linear Feedrate Specifier (see section 5.2.3.2)	Y	n/a
S	n/a	Spindle Speed Specifier (see section 5.2.3.2)	Y	n/a
G0	5-23	Rapid Traverse, Point-to-Point Motion	Y	✓
G1	5-24	Linear Interpolation	Y	
G2	5-25	CW Circular Interpolation on Plane #1	Y	
G3	5-34	CCW Circular Interpolation on Plane #1	Y	
G4	5-35	Dwell	n/a	
G8	5-39	Instantaneous Acceleration	N	
G9	5-40	Force Deceleration to Zero Velocity (see G109)	N	✓
G12	5-41	CW Circular Interpolation on Plane #2	N	
G13	5-42	CCW Circular Interpolation on Plane #2	N	
G16	5-43	Assign Plane 1 Axes	N	
G17	5-44	X/Y Plane Selection Set #1	Y	✓
G18	5-44	Z/X Plane Selection Set #1	Y	
G19	5-44	Y/Z Plane Selection Set #1	Y	
G20	5-48	Disable Normalcy Mode	Y	✓
G21	5-48	Normalcy On Left	Y	
G22	5-49	Normalcy On Right	Y	
G23	5-50	Corner Rounding Mode	Y	✓

Table 5-3. G-code and M-code Summary Con't

G-code	Page	Description	Modal	Default
G24	5-50	Disable Corner Rounding Mode	Y	
G26	5-51	Assign Plane 2 Axes	Y	
G27	5-51	X/Y Plane Selection Set #2	Y	✓
G28	5-51	Z/X Plane Selection Set #2	Y	
G29	5-51	Y/Z Plane Selection Set #2	Y	
G34	5-54	Set Safe Zone Minimum	n/a	
G35	5-54	Set Safe Zone Maximum	n/a	
G36	5-54	Enable Safe Zones	Y	
G37	5-55	Disable Safe Zones	Y	✓
G38	5-57	Enable Backlash Compensation	Y	
G39	5-57	Disable Backlash Compensation	Y	✓
G40	5-63	Disable Cutter Compensation	Y	✓
G41	5-64	Enable Cutter Compensation Right	Y	
G42	5-65	Enable Cutter Compensation Left	Y	
G43	5-66	Set Cutter Compensation Radius	Y	
G44	5-67	Set Cutter Compensation Axes	Y	
G45	5-68	Disable Polar/Cylindrical Coordinates	Y	✓
G46	5-68	Activate Polar Coordinates	Y	
G47	5-71	Activate Cylindrical Coordinates	Y	
G51	5-73	Monitor Touch Probe	N	
G52	5-73	Define Polar/Cylindrical Axes	Y	
G53	5-74	Cancel Fixture Offset	Y	✓
G54	5-74	Set Fixture Offset #1	Y	
G55	5-76	Set Fixture Offset #2	Y	
G56	5-76	Set Fixture Offset #3	Y	
G57	5-77	Set Fixture Offset #4	Y	
G58	5-78	Set Fixture Offset #5	Y	
G59	5-79	Set Fixture Offset #6	Y	
G60	5-83	Set Acceleration Time	Y	
G61	5-83	Set Deceleration Time	Y	
G62	5-84	Set Profile Time	Y	
G63	5-84	Set Sinusoidal Acceleration Mode	Y	

Table 5-3. G-code and M-code Summary Con't

G-code	Page	Description	Modal	Default
G64	5-86	Set Linear Acceleration Mode	Y	✓
G65	5-86	Set Acceleration Rate for linear-type axes	Y	
G66	5-87	Set Deceleration Rate for linear-type axes	Y	
G67	5-87	Acceleration/Deceleration Time Based	Y	✓
G68	5-88	Acceleration/Deceleration Rate Based	Y	
G70	5-89	English Programming Mode (in.)	Y	✓
G71	5-90	Metric Programming Mode (mm.)	Y	
G82	5-90	Clear G92 (software home) Command	n/a	
G83	5-91	Mirror	Y	
G84	5-93	Rotate	Y	
G90	5-95	Absolute Programming Mode	Y	✓
G91	5-96	Incremental Programming Mode	Y	
G92	5-97	Software Home (preset positions)	n/a	
G93	5-99	Inverse Feedrate Mode	Y	
G94	5-100	Normal Feedrate Mode	Y	✓
G95	5-101	Feedrate per Spindle #1 Revolution Mode	Y	
G96	5-102	RPM Spindle #1 Feedrate Programming	Y	
G97	5-104	Constant Spindle Surface Speed Programming	Y	
G98	5-106	Rotary Feedrate Dominant	Y	
G99	5-107	Linear Feedrate Dominant	Y	✓
G100	5-108	Disable spindle shutdown mode	Y	✓
G101	5-108	Enable spindle shutdown mode	Y	
G108	5-109	No Deceleration to Zero Velocity (modal)	Y	✓
G109	5-109	Deceleration to Zero Velocity (modal)	Y	
G110	5-110	Normal Circular Interpolation	Y	✓
G111	5-111	Reverse Circular Interpolation	Y	
G112	5-112	Block Delete Mode ON	Y	
G113	5-112	Block Delete Mode OFF	Y	✓
G114	5-113	Optional Stop Mode ON	Y	
G115	5-113	Optional Stop Mode OFF	Y	✓
G116	5-113	Dry Run Mode Enabled	Y	
G117	5-113	Dry Run Mode Disabled	Y	✓

Table 5-3. G-code and M-code Summary Con't

G-code	Page	Description	Modal	Default
G130	5-114	4 Kilohertz Servo Update Rate	Y	✓
G131	5-114	1 Kilohertz Servo Update Rate	Y	
G143	5-115	Activate Positive Cutter (Tool) Offsets	Y	
G144	5-116	Activate Positive Cutter (Tool) Offsets	Y	
G149	5-116	Deactivate Cutter (Tool) Offsets	Y	✓
G150	5-117	Clear Scaling Factor	Y	✓
G151	5-117	Set Scaling Factor	Y	
G153	5-119	Suspend All Fixture Offsets	Y	✓
G165	5-120	Set Acceleration Rate for rotary-type axes	Y	
G166	5-120	Set Deceleration Rate for rotary-type axes	Y	
G212	5-121	Block Delete2 Mode ON	Y	
G213	5-121	Block Delete2 Mode OFF	Y	✓
G295	5-101	Feedrate per Spindle #2 Revolution Mode	Y	
G296	5-102	RPM Spindle #2 Feedrate Programming	Y	
G300	5-123	Disable Multi-Block Look-Ahead	Y	✓
G301	5-123	Enable Multi-Block Look-Ahead	Y	
G310	5-126	Disable High Speed Machining	Y	✓
G311	5-126	Enable High Speed Machining	Y	
G360	5-127	Continue when Velocity Command Is Zero	Y	✓
G361	5-127	Wait till In Position	Y	
G395	5-101	Feedrate per Spindle #3 Revolution Mode	Y	
G396	5-102	RPM Spindle #3 Feedrate Programming	Y	
G495	5-101	Feedrate per Spindle #4 Revolution Mode	Y	
G496	5-102	RPM Spindle #4 Feedrate Programming	Y	
I	n/a	(see section 3.3)	n/a	
J	n/a	(see section 3.3)	n/a	
K	n/a	(see section 3.3)	n/a	
M0	5-128	Program Stop	n/a	
M1	5-128	Optional Program Stop	n/a	
M2	5-128	End of Program	n/a	
M3	5-129	Spindle #1 On Clockwise	Y	
M23	5-129	Spindle #2 On Clockwise	Y	

Table 5-3. G-code and M-code Summary Con't

G-code	Page	Description	Modal	Default
M33	5-129	Spindle #3 On Clockwise	Y	
M43	5-129	Spindle #4 On Clockwise	Y	
M4	5-129	Spindle #1 On Counterclockwise	Y	
M24	5-129	Spindle #2 On Counterclockwise	Y	
M34	5-129	Spindle #3 On Counterclockwise	Y	
M44	5-129	Spindle #4 On Counterclockwise	Y	
M5	5-130	Spindle #1 Off	Y	✓
M25	5-130	Spindle #2 Off	Y	✓
M35	5-130	Spindle #3 Off	Y	✓
M45	5-130	Spindle #4 Off	Y	✓
M19	5-130	Spindle #1 Off/Reorient	Y	
M219	5-130	Spindle #2 Off/Reorient	Y	
M319	5-130	Spindle #3 Off/Reorient	Y	
M419	5-130	Spindle #4 Off/Reorient	Y	
M30	5-130	Reset to Beginning of Program and Wait for Cycle Start	n/a	
M41	5-131	Machine Lock Enabled	Y	
M42	5-131	Machine Lock Disabled	Y	✓
M47	5-131	Restart Program Execution	n/a	
M48	5-132	Feedrate Override Lock	Y	
M49	5-132	Feedrate Override Unlock	Y	✓
M50	5-132	Spindle Feedrate Override Lock	Y	
M51	5-132	Spindle Feedrate Override Unlock	Y	✓
M97	5-133	Loop Over Near Call To Subroutine	N	n/a
M98	5-133	Loop Over Far Call To Subroutine	N	n/a
M103	5-134	Spindle #1 On Clockwise Asynchronously	Y	
M123	5-134	Spindle #2 On Clockwise Asynchronously	Y	
M133	5-134	Spindle #3 On Clockwise Asynchronously	Y	
M143	5-134	Spindle #4 On Clockwise Asynchronously	Y	
M104	5-134	Spindle #1 On Counter-Clockwise Asynchronously	Y	
M124	5-134	Spindle #2 On Counter-Clockwise Asynchronously	Y	
M134	5-134	Spindle #3 On Counter-Clockwise Asynchronously	Y	
M144	5-134	Spindle #4 On Counter-Clockwise Asynchronously	Y	

Table 5-3. G-code and M-code Summary Cont.

G-code	Page	Description	Modal	Default
N	n/a	Line Number	N	n/a
O	n/a	Program parameter (see section 3.3)	n/a	n/a
P	n/a	Program parameter (see section 3.3)	n/a	n/a
Q	n/a	Program parameter (see section 3.3)	n/a	n/a
R	n/a	Program parameter (see section 3.3)	n/a	n/a
S	n/a	Spindle Feedrate (see Section 3.3)	Y	n/a
T		Tool Code		
u	n/a	Axis Coordinate (see chapter 3)	n/a	n/a
U	n/a	Axis Coordinate (see chapter 3)	n/a	n/a
x	n/a	Axis Coordinate (see chapter 3)	n/a	n/a
X	n/a	Axis Coordinate (see chapter 3)	n/a	n/a
y	n/a	Axis Coordinate (see chapter 3)	n/a	n/a
Y	n/a	Axis Coordinate (see chapter 3)	n/a	n/a
z	n/a	Axis Coordinate (see chapter 3)	n/a	n/a
Z	n/a	Axis Coordinate (see chapter 3)	n/a	n/a

5.2. CNC Block Syntax

This section describes the valid G-code block syntax. It provides an overview of the G-code groups and their function. It does not provide detailed descriptions of the command word functions (CNC words starting with G or M). The user must consult Table 5-3 to find the page for a description of the functionality of the specific command word. In order to understand this chapter, the reader must be familiar with the overall structure of CNC programs (see Chapter 2: Commands) and the basic syntax for CNC words, axis masks, axis points, and Expressions (refer to Chapter 3: Expressions).

5.2.1. CNC Blocks

SYNTAX: <GCodeBlock> is (<CNCWord>~)₁<lineTerminator>

EXAMPLE: N03 G2 X7 Y8 z\$GLOB5 F500

In general, G-code blocks are composed of consecutive CNC words, optionally separated by spaces, ended with a line terminator. G-code blocks are sometimes called CNC blocks, or CNC lines. Sometimes the term ‘wordset’, used below, simply indicates a G-code block with no terminator.

The general syntax description above is inadequate, because of the many restrictions based on the type of CNC words found in the block. The more specific syntax description below attempts to breakdown CNC words by type, and their use.

SYNTAX: *<GCodeBlock>* is $[[<nWord>]]\sim[[simpleModes]]\sim[[motionBlock]]$ or
 $[[<nWord>]]\sim<standAloneBlock>$ or
 $[[<nWord>]]\sim<paramSettingBlock>$

EXAMPLES: N03 G2 X7 Y8 z\$GLOB5 F500 ; a Motion Word set block
 N03 G70 ; a Stand-alone block
 N03 G4 F500 ; a Parameter setting block

Motion wordset blocks are the only type that cause motion. However, they do not always cause motion, as the ‘G1’ Stand-alone blocks set modes. Parameter setting blocks assign task, axis or machine parameters a given value.

5.2.2. N Words

SYNTAX: *<nWord>* is $\mathbf{N}\sim<integer>$

EXAMPLE: N03

If used, N words must appear as the first word in the block. N words are ignored by the UNIDEX 600 Series Controller, they are optional and may not be used as the target of a jump statement. An exception to this is if the #MAKENCODESLABELS command is used.

5.2.3. Motion Blocks

SYNTAX: *<motionBlock>* is

$[[<motionType>]]\sim[[axisPoint]]\sim[[offsetWordSet]]\sim[[<rateWordSet>]]$

EXAMPLE: G2 X7 Y8 z\$GLOB5 F500

Motion blocks are the basic unit used to generate motion. However, only the presence of the axis point generates the motion. If the axis point is absent, there is no motion.

In general, the motionType indicates the kind of motion, the axisPoint the target, and the rateWordSet the speed. The offsetWordSet is a special wordset used only in G2 / G3 type motion.

Referencing the syntax above, a motion block can direct many actions on the same line. The list below indicates the order of execution on the controller when multiple words are on a line.

1. Simple Modes (includes M codes)
2. Rate Words (F, E and S codes)
3. Motion Modes
4. Motion Modifiers
5. Motion Types
6. Axis Points

This means that all feedrate adjustments and I/O settings in a block are guaranteed to finish before the motion in the block (if any) occurs. It also means that the motion (the axis points) executes last in the block.

For details on axis points, refer to Chapter 3: Expressions.

5.2.3.1. Simple Mode Words

SYNTAX:

<modes> is <modesAccType> or <modesAbsRel> or <modesUnits> or
<modesAccMode> or <modesDomin> or <modesSpindleOff> or
<modesMotCont> or <modesCircDir> or <modesFeedOver> or
<modeSpindleOver> or <mCode> or <modeSpindleOver>

EXAMPLE: M1050 G90 G71 G48 G99

Modes are single CNC words that must appear before any other CNC word on the CNC block, except the N word. They include M-codes and all G-codes that set modes, like the Metric/English units mode. These always execute before any other command in the block. There is no defined sequence of execution of modes with respect to each other.

All the modes, with the exception of M-codes, are paired, where the two codes perform toggling or opposite functions as each other. The user cannot include more than one from each pair in the same block. Shown below are the pairs.

<mCode>	is	M~<integer>
<modesFeedOver>	is	M48 or M49
<modesSpindOver>	is	M50 or M51
<modesUnits>	is	G70 or G71
<modesAccType>	is	G63 or G64
<modesAccMode>	is	G67 or G68
<modesAbsRel>	is	G90 or G91
<modesDomin>	is	G98 or G99
<modesSpindleOff>	is	G100 or G101
<modesMotCont>	is	G108 or G109
<modesCircDir>	is	G110 or G111

5.2.3.2. F, E and S Codes (Rate Words)

SYNTAX: $\langle rateWords \rangle$ is $[[\langle Fword \rangle \sim \langle Eword \rangle \sim \langle Sword \rangle]]$

except: the order of the E, F and S words with respect to each other is irrelevant

$\langle FWord \rangle$ is $E \sim \langle fExpression \rangle$

$\langle EWord \rangle$ is $F \sim \langle fExpression \rangle$

$\langle SWord \rangle$ is $S \sim [[[\langle integer \rangle]]] \sim \langle fExpression \rangle$

EXAMPLES: E100 F(\$GLOB0+1)

S100

S[2]100

GENERAL:

Rate words specify speeds, or feedrates. The F specifies a feedrate for linear axes; E for rotary (non-spindle) axes, and the S for spindle axes. The feedrate setting always takes place before motion, if any, on the same CNC block. Feedrates are absolute values, and do not have directionality (negative feedrates are illegal).

The user specifies feedrates via the **F**, **E** and **S** words, or, equivalently may use the task parameters as shown below. The units of the **F**, **E**, and **S** words and the parameters indicated below will vary, based upon the user units mode active and G93/G94/G95 settings. Actual speed during the move, may vary from the specified feedrate in a number of cases, see the “Actual Feedrate” section below for details.

The task parameters shown below provide more details.

F LinearFeedRate

E RotaryFeedRate

S S1 RPM, S2 RPM, S3 RPM, S4 RPM

The only difference between the feedrate words and their respective parameters, is that the parameters can be set via the MMI600 parameters page, and saved to an .Ini file. These values will then serve as the default feedrate values every time the MMI600 is started.

S words apply only to spindle commands (M3, M4, M5). The **E** and **F** words apply only in contoured moves (G1, G2, G3, G12, G13). In contoured (G1/G2 type) moves, all of the speeds of each axis are coordinated so that all the axes complete their move at the same time. Therefore, in a contoured move, even when moving multiple axes, the programmer specifies only one feedrate, the vector feedrate, which the move follows. **F** and **E** words do not apply to G0 moves, camming motion, spindle motion, or any other asynchronous motion.

$$\begin{aligned} \text{F word (user units/minute)} &= \frac{\text{square root of } (X^*X + Y^*Y \dots)}{\text{duration of move}} \\ \text{E word (RPM)} &= \frac{\text{square root of } (A^*A + B^*B \dots)}{\text{duration of move}} \end{aligned}$$

Where, X,Y etc. are the individual velocities of each *linear* axis involved in the move, and A,B etc. are the individual velocities of each *rotary* axis involved in the move.

The **S**, **E**, and **F** commands do not hold CNC program execution until the programmed velocity has been accelerated to, that is the command completes “instantaneously.”

Order of execution of codes:

When an **F**, **E** **T** and/or **S** word is specified on the same line as a motion block, the **F**, **E**, **T**, or **S** word is executed first. However, when a Simple Mode Word such as a G71, is on the same line with an **F**, **E**, **T**, or **S** word, the Simple mode word is executed before the **F**, **E**, **T** or **S** word. For example, if you are in G70 (inch mode) and execute:

G71 G1 X4 F100

The X axis will move to 4 millimeters, and will attain 100 millimeters/minute as its top speed.

In general, codes will be executed in the following order, regardless of where they appear on the line (note that some codes are forced to appear in certain positions on the line):

Simple mode words (G90, G70, M48...)

T words (T3)

F code (F4)

E code (E4)

S code (S3)

Motion modes (G0, G1, G2...)

Motion words (X4Y6)

User defined M codes(I/O) are an exception, in that are executed in the order they appear in the line. For example:

M1000 G90 M1001 G1 X4 M1002

In the above line the M1000 is executed before the G1 move, the M1001, is executed just after the G90, and the M1002 is executed the move.

Linear and rotary axes' speeds are handled independently in contoured moves. If no rotary axes are in a contoured move, then the **E** word has no effect; and similarly the **F** word has no effect in contoured moves with no linear axes. If the contoured G-code move involves the simultaneous movement of both rotary and linear axes, then the use of the **E** and **F** words can be complex; because the controller can only obey one or the other of the two words. Refer to the G98/G99 Overview for details on this special case.

The **S** word applies only to spindle motion (motion initiated by a M3 or M4). There is an extra complication, due to the fact that there are four spindles on the UNIDEX 600. If you say ‘S300’ for example, this refers always to spindle one (which axis is spindle one is specified by the task parameter S1_Index). However, if you say ‘S[2]300’, you are referring to the feedrate for spindle 2 (which axis is spindle two is specified by the task parameter S2_Index). Note that the programmed **S** word values can be viewed in the task parameters S1_RPM, S2_RPM etc. The actual spindle speed may vary from the programmed rate, if the MSO value (task parameter MSO) is not one.



The F code word can also be used on the same line as, and following certain G codes, to specify parameters related only to those G codes, for example, "G4 F.5". In these cases the F code does not specify a feedrate, but specifies a parameter used by that particular G code.

Actual Feedrates

The user can view the current programmed values of the F and E word in the task parameters LinearFeedrate and RotaryFeedRate respectively. The user can view the actual feedrate in the task parameters LinearFeedrateActual and RotaryFeedRateActual respectively. In summary, the actual feedrate values can differ from the programmed values in five cases:

1. The MFO (task parameter MFO) is not one.
2. Feedrate limiting is occurring due to an axis MaxFeedRateIPM (or MaxFeedRateRPM) machine parameter.
3. Feedrate limiting by programmed acceleration is occurring
4. The contoured move involves both rotary and linear axes (see **G98**)
5. Feedrate limiting due to programmed acceleration limiting is occurring.

T Word:

This command loads a new tool, as specified by the fExpression. The specified tool must be contained in the current toolfile, or the run time task fault: “Invalid tool specified.” is generated.

SYNTAX: <Tword> is T~<fExpression >

EXAMPLES: T10

T(\$GLOB0+1)

Loading a tool consists of taking the data stored in the tool file for that tool, and loading it into the appropriate tool parameters. In general, loading a tool does

not have any effect, until the program “leads on” to the part by executing the G codes: G143, G144, G41, or G42. Also, note that the programmer must establish the Tool axes with the G44 command, prior to loading the tool. Loading a tool file assigns to at least the following parameters:

Tool radius

Tool length

Tool offsets

S and F feedrates

T words can be executed on the same line as **F**, **E**, and **S** codes and many other **G** codes, however, be aware that if the tool specifies a required speed, then the **F** or **S** word on that line, or any following line, will override that specification, as defined by, the Order of execution of codes.

Tools must be numbered with non-zero positive integers. Specifying a tool word with a number of 0, selects the “null tool.” The “null tool” has zero cutter radius, zero tool length, and zero tool offsets. Specifying the null tool will not restore old S and F feedrates that were current before the previous tool was loaded.

Note: Only one tool can be active for all tasks

Linear and rotary axes’ speeds are handled independently in contoured moves. If no rotary axes are in a contoured move, then the **E** word has no effect; and similarly the **F** word has no effect in contoured moves with no linear axes. If the contoured G-code move involves the simultaneous movement of both rotary and linear axes, then the use of the **E** and **F** words can be complex; because the controller can only obey one or the other of the two words. Refer to the Section 5.27 for details on this special case.

The **S** word applies only to spindle motion (motion initiated by a M3 or M4). There is an extra complication, due to the fact that there are four spindles on the UNIDEX 600. If you say ‘S300’ for example, this refers always to spindle one (which axis is spindle one is specified by the task parameter S1_Index). However, if you say ‘S[2]300’, you are referring to the feedrate for spindle 2 (which axis is spindle two is specified by the task parameter S2_Index). Note that the programmed **S** word values can be viewed in the task parameters S1_RPM, S2_RPM etc. The actual spindle speed may vary from the programmed rate, if the MSO value (task parameter MSO) is not one.

The **F** code word can also be used on the same line as, and following certain **G** codes, to specify parameters related only to those **G** codes, for example, "G4 F.5". In these cases the **F** code does not specify a feedrate, but specifies a parameter used by that particular **G** code.



ACCELERATION:

After an **F** or **E** word is executed, then an automatic acceleration/deceleration to the new feedrate will begin with the next contoured move (G1/G2/G3/G12/G13) to occur. This automatic acceleration/deceleration will follow the same parameters as the acceleration of a contoured move up from zero speed. The automatic acceleration computed by the trajectory generator, can in some cases, conflict with the feedrate specified by the user in the CNC program.

After an **S** word, if the spindle is running, an acceleration/deceleration up to the new speed begins immediately. The spindle acceleration/deceleration is determined by the G0 and Asynchronous motion acceleration parameters, which are the same for the M3 command.

FEEDRATE LIMITING:

Regardless of the **F**, **E** or **S** word setting, no individual axis speed can exceed the value in the MaxFeedRateIPM machine parameter (or *MaxFeedRateRPM* machine parameter for rotary type axes). The velocities of all the axes in the contoured move automatically scale down so that no axis exceeds its maximum feedrate. If these parameters are zero, (this is the default) then there will be feedrate limiting.

The user can view the current programmed values of the **F** and **E** word in the task parameters LinearFeedrate and RotaryFeedRate respectively. The user can view the actual feedrate in the task parameters LinearFeedrateActual and RotaryFeedRateActual respectively. In summary, the actual feedrate values can differ from the programmed values in four cases:

1. The MFO (task parameter MFO) is not one.
2. Feedrate limiting is occurring due to an axis MaxFeedRateIPM (or *MaxFeedRateRPM*) machine parameter.
3. Feedrate limiting by acceleration is occurring
4. The contoured move involves both rotary and linear axes (see G98)

5.2.3.3. Motion Modifier Words

SYNTAX: *<modNormal>* is **G20** or **G21** or **G22**
<modCutter> is **G40** or **G41** or **G42**
<modVelocityBlend> is **G8** or **G9**
<motionModes> is *<modCutter>* or *<modNormal>* or *<modBlend>*

EXAMPLE: **G20 G41 G9**

These serve as modifiers to the motion type. They include cutter compensation, normalcy, and profile blending. Each of these three types has an overview explaining the modifiers use.

5.2.3.4. Motion Type Words

SYNTAX: *<motionType>* is **G0** or **G1** or **G2** or **G3** or **G12** or **G13**

EXAMPLE: **G1**

These codes direct a certain type of motion mode. They do not execute motion, they just dictate the type of motion to execute by the axis points. If they (motion type words) are modal, they not only affect the axis point on that block, but all those following until the mode is changed.

Ellipses may be created via G1 CNC programming commands or via Camming.



5.2.3.5. Offset Words

SYNTAX: *<offsetLetter>* is **I** or **J** or **K**

<offsetWord> is (*<offsetLetter>*~*<fExpression>*)₁

except: may not contain more than one word that uses the same CNC letter

EXAMPLE: I8.9 J(6+8) K(\$GLOB0)

I, **J**, **K** words specify the center coordinate of a circle and are used only in conjunction with **G2**, **G3**, **G12**, or **G13** motion G-codes. Offset words are always specified as relative coordinates, regardless of the **G90/G91** mode, measured relative to the current point. Refer to the **G2** command for more details on their use.

5.2.4. Stand-Alone Blocks

SYNTAX: *<standAloneWord>* is **G17** or **G18** or **G19** or **G27** or **G28** or **G29** or **G45** or **G93** or **G94** or **G95** or **G130** or **G131**

EXAMPLE: **G17**

Stand-alone words set modes. Refer to the section on the particular code for specifics on that G-code.

5.2.5. Parameter Setting Blocks

Parameter setting blocks take one parameter. They usually set task, axis, or machine parameters. The single parameter can be an **F** word, axis mask, or an axis point, depending on the code.

5.2.5.1. F-code Parameter Blocks

SYNTAX: $\langle FParamBlock \rangle$ is $\langle fCodeParamWord \rangle \sim \mathbf{F} \langle fExpression \rangle$
 $\langle fCodeParamWord \rangle$ is **G4** or **G43** or **G60** or **G61** or **G62** or
 G65 or **G66** or **G129** or **G165** or **G166**

EXAMPLE: G4 F5.6

These single parameter words set task or axis parameters. They must accompany an **F** word that specifies the parameter value, or a mask specifying multiple values.

5.2.5.2. Mask Parameter Blocks

SYNTAX: $\langle maskParamBlock \rangle$ is $[[\langle maskParamWord \rangle \sim \langle CNCMask \rangle]]$
 $\langle maskParamWord \rangle$ is **G16** or **G26** or **G36** or **G37** or **G39** or
 G44 or **G46** or **G53** or **G82** or **G83**
 G125 or **G126** or **G127** or **G128**

EXAMPLE: G16 X v z

Mask parameter blocks require a parameter specifying a set of axes. The user may specify any number of axes. In most cases, the axis mask is optional and the axes are implied by the context.

5.2.5.3. Point Parameter Blocks

SYNTAX: $\langle pointParamBlock \rangle$ is $[[\langle pointParamWord \rangle \sim \langle axisPoint \rangle]]$
 $\langle pointParamWord \rangle$ is **G34** or **G35** or **G38** or **G54** or **G92**

EXAMPLE: G34 X3 v2.8 z\$GLOB8

Point parameter blocks set axes or machine parameters for multiple axes simultaneously. In most cases, the user may omit the axisPoint, and the axes are implied by the context.

5.3. Non-Contoured Motion (G0)

5.3.1. Point-to-Point Positioning at a Rapid Feedrate (Motion) G0

SYNTAX: G0

EXAMPLE: G0 X4.5 Y0



The G0 command specifies axis movement for synchronous, non-contoured point-to-point movement at the rapid traverse feedrate (machine parameter *RapidFeedRateIPM*, or *RapidFeedRateRPM* for rotary type axes). The E and F keywords have no effect on G0 motion. G0 motion always decelerates to zero velocity, the G8 and G108 commands have no effect. A G0 command moves each axis specified by the *axis point* at that axes rapid traverse feedrate. No effort is made to coordinate the separate axes motion. Acceleration, deceleration, ramp time, and type (of acceleration/deceleration) are specified separately, by their axis parameters: *ACCEL*, *DECEL*, *ACCELRATE*, *DECERATE*, *ACCELMODE*, and *DECELMODE*.

The target values supplied are either relative or absolute units, dependant upon the G90 / G91 mode. The user may specify G0 motion on either linear or rotary Type axes.

We strongly recommend that the user read Section 5.1.1 through Section 5.1.9, before undertaking any motion from the CNC.



Typically, moves of this type are for operations similar to moving a tool to the workpiece or moving a finished part out to the loader. To keep cycle time to a minimum, these moves are usually performed as quickly as possible.

Although multi-axis moves of this type begin motion of all axes at the same time, all axes may not finish at the same time (each axis may have a different rapid traverse feedrate associated with it, along with a different target displacement). They do not produce contoured motion.

The **G0** command is a synchronous motion command, so the command does not complete until all axes completes their motion.

EXAMPLE

G0 G90 X10.	;Moves to X10. using rapid traverse feedrate
G0 G91 X5. Y10.	;X5.0 and Y10.0 using rapid traverse feedrate

The **G0** feedrate is limited to 100% MFO maximum.





5.4. Contoured Motion (G1, G2, G3)

5.4.1. Linear Interpolation (Motion)

G1

SYNTAX: G1

EXAMPLE: G1 X1.2 Y2.3

The **G1** command specifies synchronized, contoured linear motion. This differs from a **G0** type move, since all axes commanded to move, begin and end at the same time.

The target values supplied are either relative or absolute, based on the G90 / G91 mode. The user may specify **G1** motion on either linear or rotary Type axes.

Normally, the **F** word (or **E** word for rotary axes) determines the speed of the motion. However, in complex motion where the rotary and linear axes move simultaneously, the dominant Type of axes determines the **E** or **F** word used. Refer to the Section 5.27 for details on this.



We strongly recommend that the user read Section 5.1.1 through Section 5.1.9, before undertaking any motion from the CNC.



The acceleration/deceleration type used is defined by the current operational mode of the Ramp Type and Accel Mode G-code groups.

Consecutive contoured moves may be blended together.

EXAMPLE PROGRAM:

G91 G70 G1 X4.0 Y3.0 F50. ; A straight line will be produced from the current
; position to the X=4.0, Y=3.0 coordinate position,
; at a feedrate of 50 inches per minute.

5.4.2. Circular Interpolation CW on Coordinate System #1 (Motion)

G2

SYNTAX: G2~<EndingPoint>~<CenterPoint>

EXAMPLES: The 4 examples below are alternate specifications of the same circular arc, which is shown in Figure 5-1 (examples assume **G17**, and starting point of {.5,3}).



G91 G2 X2 Y-2 I.5 J-1.5	; End Point and circle center specified (IJK ; method)
G91 G2 X2 Y-2 R1.58114	; End Point and radius specified (R method)
G91 G2 P108.435 Q-18.435 R1.58114	; Angles and radius specified (PQ Method and ; R method)
G91 G2 Q-18.435 I.5 J-1.5	; Ending angle and circle center (Q Method ; and IJK method)

The **G2** command generates synchronous, contoured motion that produces a clockwise (CW) circular arc by the coordinated motion of two axes (see Figure 5-1). Viewing the axes plane from the negative direction of a perpendicular axis (per the right hand rule), the arc direction is clockwise. Use **G3** to generate counterclockwise motion on coordinate system #1. The axes specified in the <EndingPoint> must be linear Type axes. The F word determines the velocity. When an F word is specified on the same line as a **G2**, **G3**, **G12** or **G13** command, the F word is executed first.

EXAMPLE PROGRAM:

```

G16 X Y Z      ; Coord1I=X, Coord1J=Y, Coord1K=Z (Default)
G17            ; Coord1Plane={Coord1I,Coord1J} (Default)
G90            ; Absolute programming mode
G1 X.5 Y3
G2 X2.5 Y.5 I1 J-1.5 ; FOLLOWS PATH SHOWN IN Figure 5-1
                      ; Starting coordinate = {.5, 3}
                      ; Ending coordinate = {2.5, 1}
                      ; Center coordinate = {1, 1.5}

```

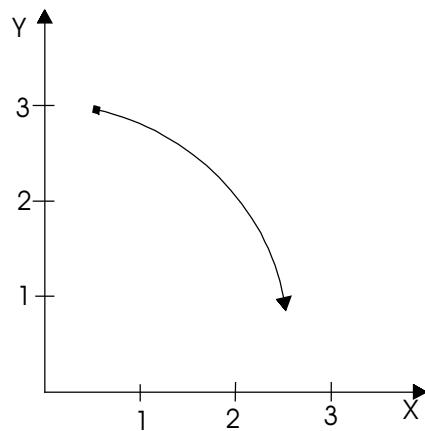


Figure 5-1. CW Circular Interpolation

Clockwise and Counter-Clockwise Circular Axis Plane

By default, this command will produce circular motion in the X and Y axis plane as shown in Figure 5-1. However, this command can produce motion in any one of three planes (see Figure 5-2 and Figure 5-3), 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 axis plane that is used within coordinate system 1 and 2, is selected by the **G17/G18/G19** and **G27/G28/G29** commands, respectively.

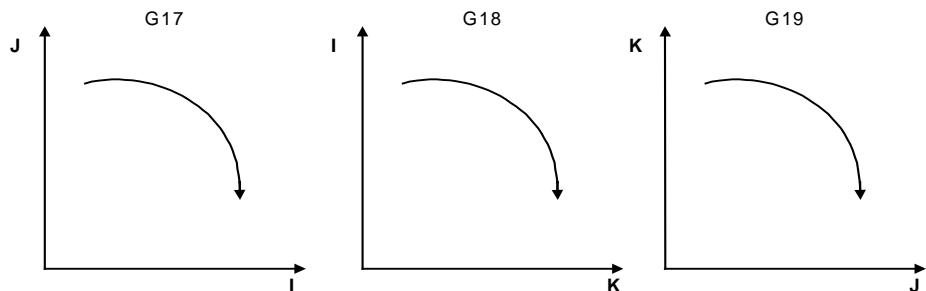


Figure 5-2. Orientation of a G2, in various planes in Coord. System #1

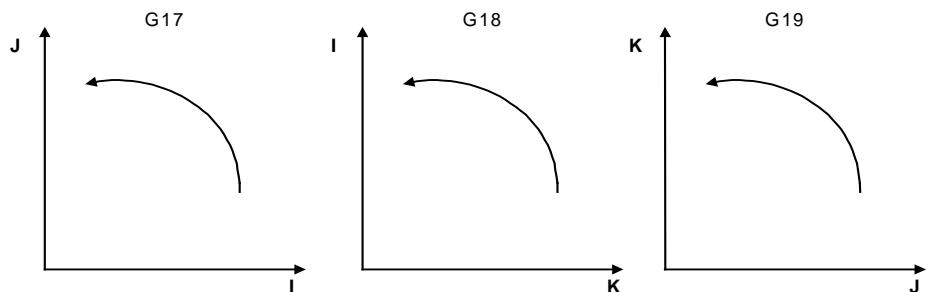


Figure 5-3. Orientation of a G3, in various planes in Coord. System #1

Circular Speed

Normally, the **F** key word determines the vectorial speed of the axes. However, the **E** or **F** key word alone may determine the axis component speeds in complex cases where the user moves rotary and linear type axes simultaneously. Refer to the **G98** G code for more details on this situation. However, in some cases Feedrate Limiting or the BlendMaxAccelCircleIPS2 task parameter may reduce the vectorial speed. When an F word is specified on the same line as a **G2**, **G3**, **G12** or **G13** command, the F word is executed first.

Circular Starting and Ending Points

The **G2** / **G3** / **G12** / **G13** arc always starts at the current position and ends at the <EndingPoint>. The ending point may be specified in two distinct ways; explicitly by X Y Coordinates or by an angle (PQ method).

X Y Coordinate Circular Ending Point

Normally, the ending point of an arc is explicitly specified within the CNC block, similar to **G1** moves (See the first and second examples, in the **G2** command, section 5.4.2). If **G90** mode is active, the ending point coordinates are interpreted as absolute coordinates, if **G91** mode is active, the ending point coordinates are interpreted as relative. If the ending point is specified by X Y coordinates (or the two axes generating the circular motion), one or both of the target values may be omitted, using the default circular values. If the ending position is equal to the current position, a 360° circle is produced.

PQ Method and Q Method

The user has the option of not specifying the ending position as an X Y coordinate, but, instead specifying the absolute starting angle (P Word) and ending angle (Q Word) of the arc. Each of these absolute angles are measured counter-clockwise, from a line drawn through the center point running parallel to the X axis (see Figure 5-4, also, see the third and fourth examples, in the **G2** command, section 5.4.2).

Starting Angle (P Word)

The absolute starting angle is specified via the P Word, in degrees. The modulo value of the angle specified will be used to create a value between 0 and 360 degrees.

Ending Angle (Q Word)

The absolute ending angle is specified in the Q word, in degrees.

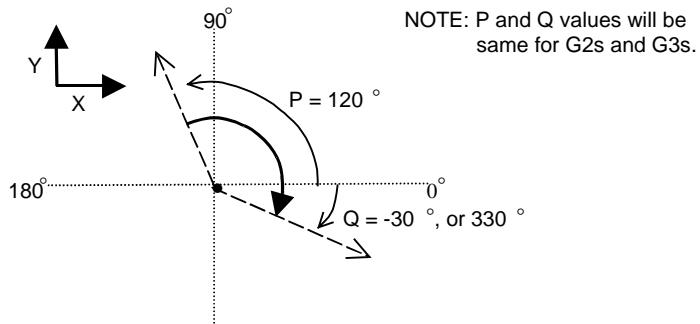


Figure 5-4. PQ Method Example

The Center Point

The radius and center of the circle can be defined in two distinct ways: by “IJK” codes, or by “R” codes. The “IJK” method is the original RS-274 CNC standard, but the newer R method offers the following advantages:

1. Specifying R is simpler and less confusing.
2. Specifying R avoids the possibility of radius errors (see below).

However, there are disadvantages:

1. The R method cannot be used to specify 360-degree arcs.
2. You must use negative radii values to perform arcs from 180 to 360 degrees.

Circular Radius “R” Method

The R code specifies the radius of the circle. However, this does not specify which of the two possible center points, when used with the X Y coordinates of the desired endpoint, as shown below. The sign of the radius specifies this. If the radius is positive, the controller will generate the shorter arc (always 180 degrees or less). If the radius value is negative it will generate the longer arc (always between 180 degrees and 360 degrees). See Figure 5-5. There are some start/endpoint/radius combinations where both the negative and positive radius values will produce the same 180 degree arc. You cannot use the R word to specify a 360 degree arc, or a fault will be generated.

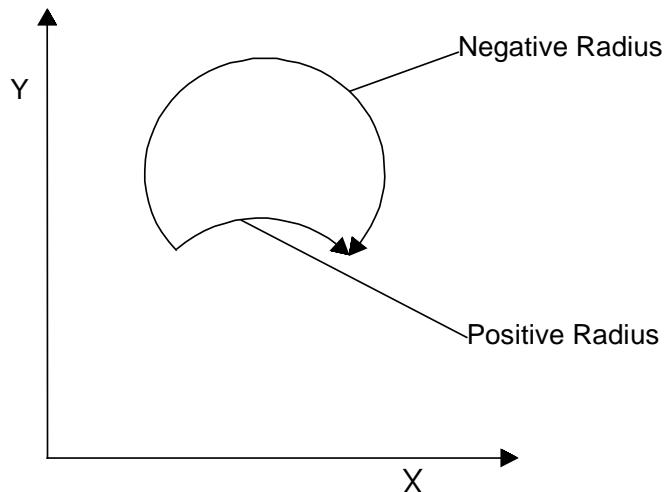


Figure 5-5. “R” Method Example

“IJK” Method

The user must define the radius of the arc by specifying the offsets to the center point of the circle, in an offset block, using the two appropriate **I**, **J** or **K** offset specifiers. The offsets are always specified relative to the current position, regardless of the **G90/G91** mode. The IJK method, unlike the R method, can result in plane errors and radius errors, if specified inaccurately.

The **I**, **J** and **K** keywords always refer to the axes specified by the coordinate task parameters:

Coordinate System #1: Coord1I, Coord1J and Coord1K task parameters, set via **G16**.

Coordinate System #2: Coord2I, Coord2J, and Coord2K task parameters, set via **G26**.

Refer to the circular starting and ending points and this command’s example programs for more clarification of this complex issue.

Circular Plane Errors

Plane errors can occur, if the center point is being specified by the IJK method. The **I**, **J** and **K** keywords always refer to the axes specified by the coordinate task parameters:

Coordinate System #1: Coord1I, Coord1J and Coord1K task parameters, set via **G16**.

Coordinate System #2: Coord2I, Coord2J, and Coord2K task parameters, set via **G26**.

The user must supply two axes to specify the endpoint, as well as two offsets to specify the center. If the axes names provided in the ending point do not match the axes inferred by the I/J/K offset values provided, a plane error occurs.

Circular Radius Errors

Radius errors may occur, if the center point of an arc (G2 / G3) is being specified by the IJK method. These may be due to rounding or limited precision of the calculations, such as from a CAD package. When using the IJK method, the controller verifies that the current point and end point are equidistant from the center point. The distance between the center point and the current position must be equal to the distance between the ending position and the center point (within the tolerance defined by the MaxRadiusError task parameter). If this condition is not true, the controller generates a ‘Radius Error’ fault.

If the radius and two endpoints do not agree, but fall within the MaxRadiusError tolerance, the controller will re-compute a new center point for the arc, which is as close as possible to the specified center point, and also having an equal radius for the starting and ending points.

Users of CAD software packages who encounter the Radius Error Fault can easily resolve this problem by having the CAD software package output a greater number of decimal places or by adjusting the MaxRadiusError task parameter, or by having the CAD package output arcs with the R method. Also, you may set the MaxRadiusError task

parameter to -1 . This will disable circular radius error messages, causing the controller to recalculate the center point, as required.



Prior to version 6.103, the controller handled this situation differently, assuming the ending point to programmed center point as the radius, and “jumped” to the starting point necessary to satisfy this radius. For backward compatibility, this functionality may be reinstated by setting bit 2 of the CompatibilityMode task parameter. See the MaxRadiusAdjust task parameter also.

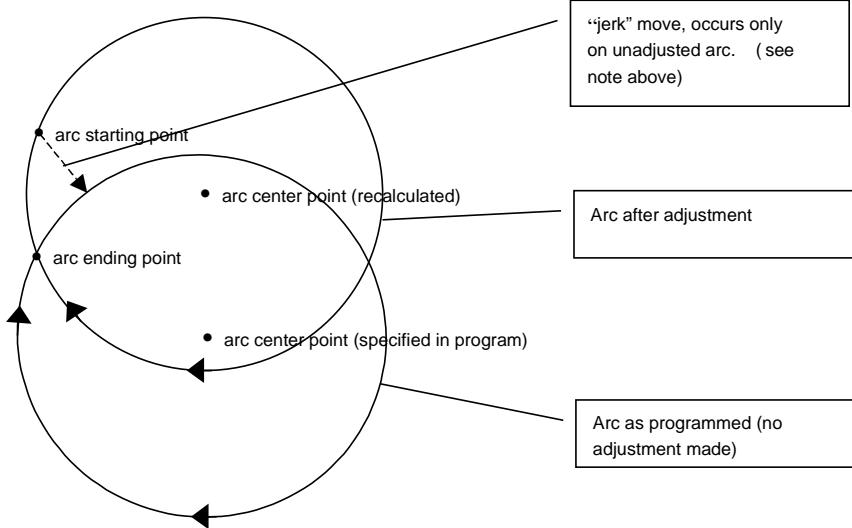
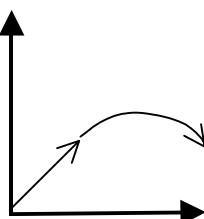


Figure 5-6. Circular Radius Example

Imprecision of Target Values

Many CNC programs are generated by CAD/CAM packages. These software packages only output values out to a certain number of digits, yielding imprecision in the motion. For example:



```
N01 G91 G70 ; inches, relative coordinates
N02 G1 X1. Y1
N03 G2 X2.6666 Y0 I1.3333 J-1.3333
```

The intent here is to generate motion at a 45° angle to $\{1,1\}$, then a 90° arc of 1 and $1/3$ inch radius, that begins tangential to the 45° move. However, due to limited precision, (to be perfectly accurate the values given for X, J, and I should be carried out to an infinite number of places, i.e. $1.33333333\dots$), the circle will not be correct. This imprecision may cause three problems:

1. The arc radius as measured from the start point, will be different than that measured from the end point possibly producing a circular radius error.

2. The arc will not be exactly tangential to the line (this may cause jerking if the Normalcy mode is active).
3. The arc will not be exactly tangent to the line (this may cause link arc degenerates to line error, jerking or motion queue starvation if Cutter Compensation is active).

In all three cases the controller provides parameters that may be adjusted, allowing small imprecision errors to be ignored.

MaxRadiusError - errors in starting/ending positions of **G2 / G3** moves

NormalcyToleranceDeg - errors in tangency between moves, as it relates to Normalcy mode

CutterToleranceDeg - errors in tangency between moves, as it relates to Cutter Compensation

Correcting Circular Radius Errors

There are a number of solutions for circular radius errors. The preferred solutions are 1 or 2, since they fix the problem rather than "work around" it. If you use solution 3 or 4, then the program will readjust your circle, as discussed after the list.

1. Use the R method instead of the IJK method for specifying the circle. In this method you specify the target point of the arc, and the radius of the arc. For example "G2 R5 X10" as opposed to "G2 I5 X10". In the R method you can never get a circular radius error because you are not over-specifying the problem. There are some considerations in the R method for performing arcs subtending angles more than 360 degrees, please see the U600 help file under "Circular Radius Method" (use the Index) for details.
2. Force your CAD "post-processor" to produce higher precision in its outputs, i.e.; output more decimal places. This applies to the I, J, K as well as to the X and Y coordinates.
3. Raise the MaxRadiusError task parameter so that the program "accepts" these limited precision coordinates without generating an error.
4. Set the MaxRadiusError task parameter to -1, to avoid all radius errors. However, this is dangerous, as errors in the CAD post-processor will no longer be detected.

If you use solutions 3 or 4 above, the program will "change" the arc center so that it no longer contradicts the endpoint/startpoint. It changes it by moving the center the shortest possible distance it can, that fits the endpoint/startpoint.

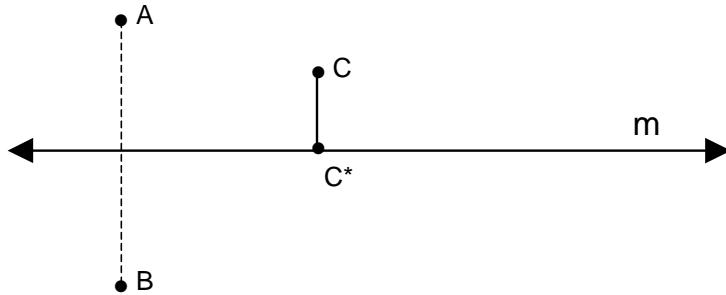


Figure 5-7. Arc Center Change

If the user is at point A, and moves to point B along an arc, but specifies the erroneous point C as the center, the program will compute (assuming a circular radius error is not generated) a new center point C* as shown below. Note, that all legal centers to the arc lie on the line m, which lies perpendicular to line segment AB, and bisects segment AB. The center chosen (C*) lies on line m, such that the segment C* is perpendicular to m.

Defaults

The controller supplies a default value of zero for any offset omitted from the CNC program block. The controller supplies a default value equal to the current position for any target omitted from the CNC program block. The CNC program block must have at least one offset value, and need not have any target values. However, omitting targets or offsets, makes the CNC program more difficult to read, since the default value is based on the **G16** and **G17/G18/G19** active modes.

See the code fragment below, which assume a starting position of {X=4, Y=4} G17, G16 X Y Z:

G2 G90 X6.0 Y4.0 I1.0	; Same as G2 G90 X6.0 Y4.0 I1.0 J0.0
G2 G90 X6.0 I1.0	; Same as G2 G90 X6.0 Y4.0 I1.0 J0.0
G2 G90 I-2.0	; Same as G2 G90 X4.0 Y4.0 I-2.0 J0.0

Helices and Dual Circular Motion

To produce helical interpolation it is necessary to program two axes to generate circular interpolation and a third axis to generate linear interpolation. This is accomplished by specifying a **G2 / G3** (or **G12 / G13**) and a **G1** on the same line.

Also, the user can specify two circles to execute simultaneously (on four different axes) via the second coordinate system **G12/G13** circular commands. However, in this case the **F** feedrate determines the velocity for the circular motion in both coordinate systems.

To produce a helix or dual circular motion, the G codes and their associated parameters must be grouped together, as follows:



G1 Z5. G2 X1. Y. I1 J0. F100. ; Helical motion

G2 X1. Y1. I1. J0. G13 x1. Y1. I1. J0. F100 ; Dual circular motion

In other words, the parameters to the **G1** command must follow the **G1** command, the parameters to the **G2** command must follow the **G2** command, the parameters to the **G13** command must follow the **G13** command, etc.

EXAMPLE PROGRAM:

```
Here are some examples to help clarify the G16/G17 type options
;
G2 X1 Y1 I5 J6      ; This is OK
G2 U1 V1 I5 J6      ; INVALID SYNTAX: neither U nor V is one of the Coord1 axes
G16 U A B           ; Change: Coord1I=U, Coord1J=A, Coord1K=B
G2 U1 A1 I5 J6      ; Corrected: U and V are now Coord1I and Coord1J, respectively
G16 A U B           ; Switch around U and A axes: Coord1I=A, Coord1J=U, Coord1K=B
G2 U1 A1 I5 J6      ; Still OK, but now the I is for A axis, J is for U axis
G16 X Y Z           ; Change back to default: Coord1I=X, Coord1J=Y, Coord1K=Z
G19                 ; But change Coord1Plane to {Coord1J, Coord1K}
G2 X1 Y1 I5 J6      ; INVALID SYNTAX: I is no longer in coordinate 1 plane.
G2 X1 Y1 I5 K6      ; Corrected: note that now J,K must be used (J,K still match
; to the X,Y axes since no change in G16 setting)
```

5.4.3. Circular Interpolation CCW on Plane #1 (Motion)**G3****SYNTAX:** G3~<Ending Point>~<Center Point>**except:** the user must supply 2 axis letters, and 2 offset letters**EXAMPLE:** G3 X5 Y5 I-5 J0 ; Execute a CCW circle

A **G3** command generates a counterclockwise (CCW) arc and in every other respect is identical to a **G2** command. Refer to the **G2** description for all other details on the **G3** command. Compare Figure 5-8 with Figure 5-1 under the **G2** command description.

EXAMPLE PROGRAM:

```
G90G3X5Y3I0J2 ;Starting coordinate={5,3}  
 ;Ending coordinate={2.5,1}  
 ;Center coordinate={5,1}
```

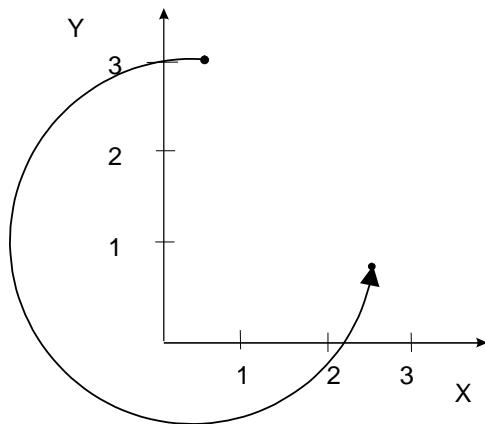


Figure 5-8. CCW Circular Interpolation

5.5. Dwell (G4)

5.5.1. Dwell

G4

This command causes a delay in program execution. The duration of the delay must be specified in seconds using the **F** keyword. The resolution of the delay is 0.001 seconds or 1 millisecond.

Use the WAIT statement to cause a delay based on a condition.

SYNTAX: **G4 F</Expression>** ; where the expression is the time in seconds

EXAMPLE PROGRAM:

G4F5. ;Dwell 5 seconds

5.5.2. Asynchronous Dwells

Users are recommended to use the **G4** command for dwells, however, advanced CNC programmers may require asynchronous dwells. That is, execute a dwell (on the same task) while executing other CNC program lines. This technique also allows you to dwell until a condition is satisfied, rather than dwell a particular time. This may be accomplished as in the following CNC program fragment.

```
; The following CNC program will set an output via M5000, dwell approximately 15
; milliseconds,
; then, clear the output, within the subroutine labeled EVAL.

DVAR $TIME_DELAY
$TIME_DELAY = 15           ; milliseconds

M5000 = 1                 ; set output
CLOCK.X = 0                ; clear 1 msec. timer
ONGOSUB( CLOCK.X > $TIME_DELAY ) FARCALL "" EVAL      ; "" implies current program
;
; Rest of program here
;
ENDWHILE
M2

EVAL:
M5000 = 0                 ; clear output
ONGOSUB CLEAR( CLOCK.X > $TIME_DELAY )
RETURN RETURNTYPE_START
```

5.6. Velocity Blending (G8, G9, G108, G109)

G8 and **G9** relate to how the controller behaves when two consecutive contoured CNC motion blocks (**G1**, **G2**, **G3**, **G12** or **G13**) execute consecutively. The controller, by default, blends two contoured G-code moves together, smoothly accelerating or decelerating to the new speed between the two moves. **G8** and **G9** can alter this behavior. **G9** forces the controller to decelerate to zero (obeying the deceleration settings detailed in **G60-G68**) at the end of the block that the **G9** appears. **G8** forces the acceleration/deceleration to the new speed to be instantaneous (the **G60-G68** accel/decel settings are not obeyed).

The **G8** and **G9** commands only apply to the block in which they appear. Refer to **G108/G109** to perform the **G9** action modally. The **G8** action cannot be performed modally. However, the *AccelTimeSec*, *DeclTimeSec*, *AccelRateIPS2*, and *DecelRateIPS2* task parameters could be changed providing a modal **G8** mode. A **G8** must be included on every block if instantaneous acceleration/deceleration is desired on every block. Refer to Figure 5-9 for all **G8** and **G9** velocity profile combinations.



G8 and **G9** have no effect on **G0** or asynchronous moves. The controller always decelerates smoothly to zero velocity between **G0** moves.

EXAMPLE PROGRAM:

G91 G68	;Relative coordinates, linear accel/decel
G1 X10F10	;Move 1,(default) blends with accel to move 2
G1 X10F20G9	;Move 2, forces decel to zero at end of move
G1 X10F10G8	;Move 3,instantaneous accel and decel
G1 X10F20G8 G9	;Move 4,instantaneous accel,force to zero decel

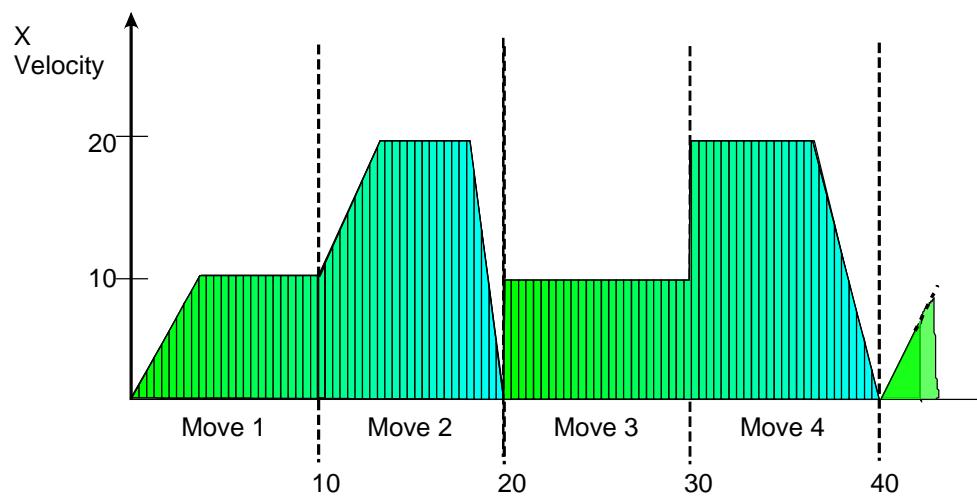


Figure 5-9. G8 and G9 Velocity Profile

Instantaneous Accelerations and Decelerations

Instantaneous accelerations/decelerations during transitions between contoured (**G1**, **G2**, **G3** commands), blended (see **G9**) moves can occur in some circumstances, despite the accelerations and decelerations defined by the user.

Three typical causes of this are:

- Corners
- Direction Reversals
- Exiting Constant Velocity Mode

Corners

Attempting constant velocity around the corners (90 degree angles) of a part may cause instantaneous accelerations. Examine the following code fragment, where X and Z are linear axes.

```

G108 ; Enforce automatic velocity blending
G90 G0 X0 Z0 ; Goto {0,0}; use absolute coordinates from now on
G1 X100 Z0 F100 ; X at 100 units/sec, Z does not move.
G1 X200 Z100 F141.4 ; X at 100 units/sec, Z at 100 units/sec

```

The controller will not decelerate in between the two moves (no G9 specified); so the X axis travels at 100 units/min. at the instant the second move begins. This means that the Z axis must also be 100 units/min. at that instant, in order for the two axes to finish at the same time. However, in the last instant (the end of the first move) the Z axis was not moving. This results in an instantaneous velocity change of the velocity command of the Z axis. This instantaneous velocity change will take place in 1 millisecond, resulting in a very high acceleration.

There are only three viable alternatives to instantaneous accelerations, such as corners or direction reversals.

Exiting Constant Velocity Mode

Instantaneous deceleration may occur in the constant velocity mode. Examine the following example.

```

G90 G0 X0 Z0 ; Goto {0,0}; use absolute coordinates from now on
G1 X100 F100 ; X at 100 units/sec
G1 X1 F100 ; X at 100 units/sec
G4 F.5

```

Instantaneous deceleration may occur at the end of a sequence of blended moves. There is no deceleration at the end of each move, except for the last move in the sequence. However, if the last move specifies a very short distance, relative to the velocity, then the controller does not have time to generate the specified deceleration. It must apply a deceleration that brings it to the specified target from the speed achieved in the last move. This can result in deceleration much faster than specified by the user and can generate nearly instantaneous deceleration.

Direction Reversals

Changing the direction of motion of an axis while in the constant velocity mode may cause instantaneous accelerations.

```
G108      ; Enforce automatic velocity blending
G91      ; Use relative coordinates
G1 X100 F100    ; X moving in positive direction
G1 X-100 F100   ; X moving in negative direction
```

Eliminating Instantaneous Accelerations and Decelerations

Direction reversals or instantaneous acceleration/deceleration, causing axis jerking, can easily occur when in **G108** mode and there is no **G8** or **G9** on the line. There are only three viable alternatives for instantaneous accelerations and decelerations, such as corners or direction reversals.

Decelerate to a stop in between the moves (i.e., place a **G9** on the line, or use **G109** mode). However, this will take significantly longer to execute the move.

Accelerate the axis up as quickly as possible starting at the beginning of the second move, without altering the first axis speed (Use **G23** or *VELTIMECONST*). However, this means the path will not be a sharp corner, it will be ‘rounded’.

The controller can automatically decelerate like the **G9** command, which is activated by the *BlendMaxAccelLinearIPS2* (or *BlendMaxAccelRotaryDPS2* for rotary axis) task parameter.

It is by no means clear which of these solutions is preferable, it depends on the application. So the controller leaves it up to the user to decide. If solution 1 is desired, the user must add a **G9** to the first **G1** CNC program block.

Solution 2 can be accomplished with the **G23** command (or the *VELTIMECONST* axis parameter). If neither of these solutions may be applied, then the velocity command will make an instantaneous change. However, it is clear that the actual velocity change will never be instantaneous, but will be limited by the mechanics of the system. So, if neither a **G9** nor the *VELTIMECONST* parameter is used by the programmer, then the mechanics imposes a form of solution 2, however, the rapid acceleration change will most likely affect part quality and or generate RMS current or position error faults.

5.6.1. Instantaneous Acceleration

G8**SYNTAX:** G8

This command causes the axis to accelerate to the new velocity instantaneously, at the beginning of a contoured move (G8 applies only to G1, G2, G3, G12, and G13 moves). Figure 5-10 displays the velocity profile with the G8 command. Without this command, acceleration is performed based upon the current settings of the ACCELMODE and ramp type operational modes (G60 through G68). G8 applies only to contoured motion. A G8 must be on every CNC line if instantaneous acceleration/deceleration is desired on each line. If the user desires a modal G8, they must set the *AccelTimeSec*, *DecelTimeSec*, *AccelRateIPS2*, and *DecelRateIPS2* task parameters to zero for all axes.

The acceleration can not occur in a time period less than the value specified by the UpdateTimeSec task parameter.

**EXAMPLE PROGRAM:**

```
G90G1 G8X1.F100.  
G90G1 X1 F200.  
G90G1 X1 F100.
```

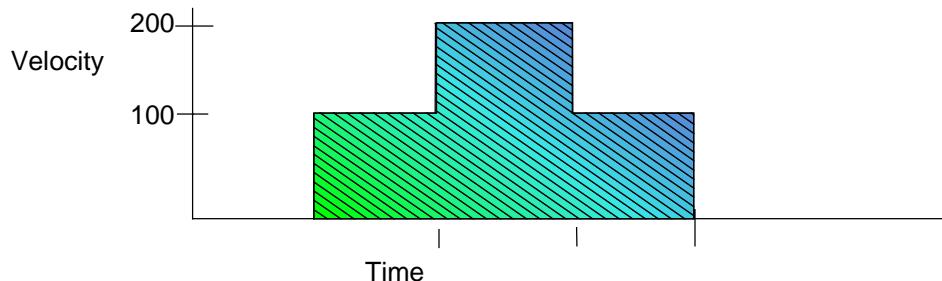


Figure 5-10. Velocity Profile with G8

If a move executing in the G8 mode is placed into the feedhold mode, the deceleration to zero is instantaneous. The same applies to MFO adjustments during a G8 mode move.





Forced instantaneous acceleration, even while a **G8** is not active, may occur under some circumstances. See Corners for more information.

5.6.2. Force Deceleration

G9

The **G9** command forces the axes to decelerate to zero velocity at the completion of a contoured move (**G9** applies only to **G1**, **G2**, **G3**, **G12**, and **G13** moves). Figure 5-11 and Figure 5-12 give a comparison of the velocity profile with and without **G9**. The subsequent moves then accelerate from zero velocity to the commanded feedrate. This command is not modal. For the equivalent modal command, see the **G109** command. The following example illustrates the effect of this command upon a sequence of motion blocks.

SYNTAX: **G9**

EXAMPLE PROGRAM: Velocity profile without **G9**

```

G91      ; Incremental positioning mode
F60.    ; Set the vector feedrate to 60
G1 X1.   ; Move the Xaxis 1.0
F120.   ; Set the vector feedrate to 120
G1 X2.   ; a second time 2.0
F60.    ; Set the vector feedrate to 60
G1 X1.   ; and another move of 1.0
G4 F1.   ; Dwell for 1 second
G1 X1.   ; This move will have an acceleration, because a G4 preceded it

```

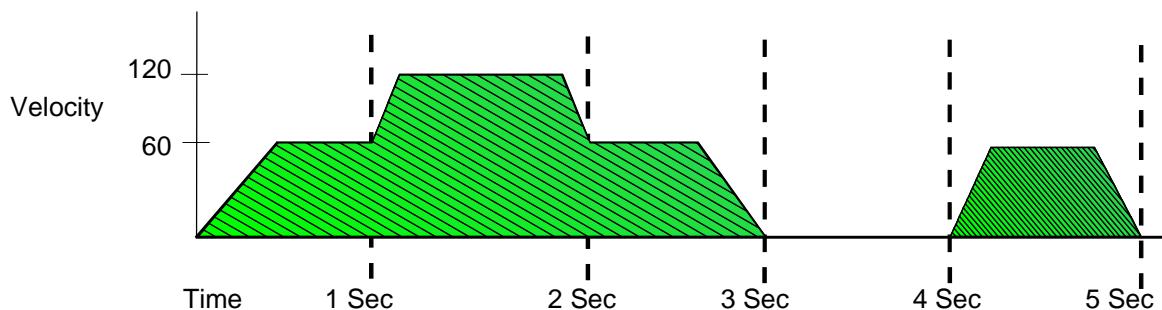


Figure 5-11. Velocity Profile Without **G9**

EXAMPLE PROGRAM: Velocity profile with G9

```

G91      ;Incremental pgm. mode
F60.    ;Set the vector feedrate to 60
G1 G9 X1. ;Move the X axis 1.0
F120.   ;Move at a Velocity of 120
G1 G9 X2. ;a second time 2.0
F60.    ;Set the vector feedrate to 60
G1 G9 X1. ;and another move of 1.0
G4 F1.   ;Dwell for 1 second
G1 X1.   ;Move 1.

```

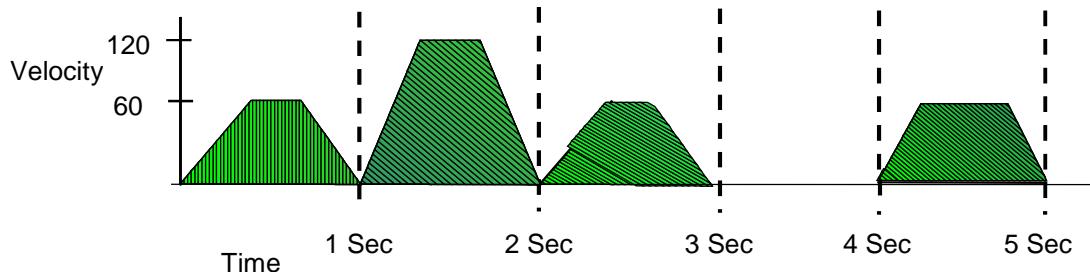


Figure 5-12. Velocity Profile with G9

When executing a return move (from jog and return) the controller always decelerates to zero before resuming the move (if any) that was interrupted by the jog and return.



5.7. Contoured Motion on Coordinate System # 2 (G12, G13)

G12 and **G13** produce circles just like **G2** and **G3**, but on Plane #2. This allows the programmer to produce simultaneous circular motion on two planes.

5.7.1. Circular Interpolation CW on Coordinate System #2 (Motion) G12

SYNTAX: **G12~<axisPoint>~<offsetBlock>**

EXAMPLES: The 4 examples below are alternate specifications of the same circular arc, which is shown in Figure 5-1 (examples assume **G27**, and starting point of {.5,3})

G91 G12 X2 Y-2 I.5 J-1.5	; CW, target+circle center specified (IJK method)
G91 G12 X2 Y-2 R1.58114	; CW, target+radius specified (R method)
G12 P101.31 Q-11.31 R1.58114	; CW, angles+radius specified (PQ and R method)
G12 Q-11.31 I.5 J-1.5	; CW, angles+circle center (IJK and Q method)

The **G12** command generates contoured synchronous motion, that produces a Clockwise (CW) circular arc by the coordinated motion of two axes in coordinate system #2, see Figure 5-2 and Figure 5-3. Viewing the axes plane from the negative direction of a perpendicular axis (per the right hand rule), the arc direction is clockwise. Use **G13** to generate counterclockwise motion on coordinate system #2. The axes specified in the <axis point> must be linear Type axes. The F word determines the velocity. When an F word is specified on the same line as a **G2**, **G3**, **G12** or **G13** command, the F word is executed first.

The **G12** command provides the programmer the ability to generate two circular motions simultaneously. This is accomplished by programming a **G12 /G13** on the same line as a **G2/G3**. The F feedrate would specify the feedrate for both circles.

Refer to the task parameters *Coord1Plane* and *Coord2Plane*, for more details on how to define the axes planes.

5.7.2. Circular Interpolation CCW on Coordinate System #2 **G13**

SYNTAX: **G13~<axisPoint>~<offsetBlock>**

EXAMPLES: The 4 examples below are alternate specifications of the same circular arc, which is shown in Figure 5-1 (examples assume **G27**, and starting point of {.5,3})

G91 G13 X2 Y-2 I.5 J-1.5	; CCW, target+circle center specified (IJK method)
G91 G13 X2 Y-2 R1.58114	; CCW, target+radius specified (R method)
G13 P101.31 Q-11.31 R1.58114	; CCW, angles+radius specified (PQ and R method)
G13 Q-11.31 I.5 J-1.5	; CCW, angles+circle center (IJK and Q method)

The **G13** command generates contoured synchronous motion, that produces a Counter-Clockwise (CCW) circular arc by the coordinated motion of two axes in coordinate system #2, see Figure 5-2 and Figure 5-3. Viewing the axes plane from the negative direction of a perpendicular axis (per the right hand rule), the arc direction is counter-clockwise. Use **G12** to generate clockwise motion on coordinate system #2. The axes specified in the <axis point> must be linear Type axes. The F word determines the velocity. When an F word is specified on the same line as a **G2**, **G3**, **G12** or **G13** command, the F word is executed first.

The **G13** command provides the programmer the ability to generate two circular motions simultaneously. This is accomplished by programming a **G12 /G13** on the same line as a **G2/G3**. The F feedrate would specify the feedrate for both circles.

Refer to the task parameters *Coord1Plane* and *Coord2Plane*, for more details on how to define the axes planes.

5.8. Coordinate System #1 Configuration (G16 – G19)

Coordinate System #1 is used only in context with the **G2** and **G3** commands and determines the axes comprising the first coordinate system (#1) XYZ plane where **G2** or **G3** will produce circular motion.

5.8.1. Assign Coordinate System #1 Axes

G16

SYNTAX: **G16~<axisMask>**

except: the user must supply exactly 3 axis letters (order of the letters is significant).

EXAMPLE: **G16 u v w**

This G-code is equivalent to assigning to the task parameters: *CoordII*, *CoordIJ* and *CoordIK* simultaneously. These task parameters and **G16** are used only in the context with **G2** or **G3** commands. *CoordII* is assigned to the first axis in the mask, *CoordIJ* is assigned to the second axis in the mask, and so on.

By default:

<i>CoordII</i>	is 0	(X task axis)
<i>CoordIJ</i>	is 1	(Y task axis)
<i>CoordIK</i>	is 2	(Z task axis)

The order of the axes in the axis mask is significant, because it determines the mapping of the **I**, **J**, **K**. For example, “**G16 X Y Z**” is different than “**G16 Z Y X**.”



Refer to the example programs for **G17**, **G18**, and **G19** for a better understanding of the complex issues surrounding **G16**.



5.8.2. Plane Selection Codes Set # 1

G17/G18/G19

SYNTAX: **G17** or **G18** or **G19**

EXAMPLE: **G17**

These G-codes are equivalent to assigning to the task parameter *Coord1Plane*. This task parameter and **G17**, **G18**, **G19** are used only in the context with **G2** or **G3** commands.

G17 selects Plane 1, **G18** selects Plane 2, and **G19** Plane 3 (see Figure 5-13). Reference the example program below.

EXAMPLE PROGRAM:

```

G16XY Z ;CoordII=X,CoordIJ=Y,CoordIK=Z(this is the default)
G17 ;Coord1Plane={CoordII,CoordIJ} (this is the default)
G2X1 Y1 I5 J6 ;This is OK
G2U1 V1 I5 J6 ;INVALID SYNTAX: neither U nor V is one of the CoordI axes
G16UAB ;Change: CoordII=U,CoordIJ=A,CoordIK=B
G2U1 A1 I5 J6 ;Corrected, U and A now CoordII and CoordIJ, respectively
G16AUB ;Switch around U and A axes, CoordII=A,CoordIJ=U,CoordIK=B
G2U1 A1 I5 J6 ;Still OK, but now the I is for A axis,J is for U axis
G16XYZ ;Change back to default: CoordII=X,CoordIJ=Y,CoordIK=Z
G19 ;But change Coord1Plane to {CoordIJ,CoordIK}
G2X1 Y1 I5 J6 ;INVALID SYNTAX: I is no longer in coordinate 1 plane.
G2X1 Y1 J5 K6 ;Corrected, note that now J,K must be used (J, K still match
;to the X, Y axes, since no change in G16 setting)

```

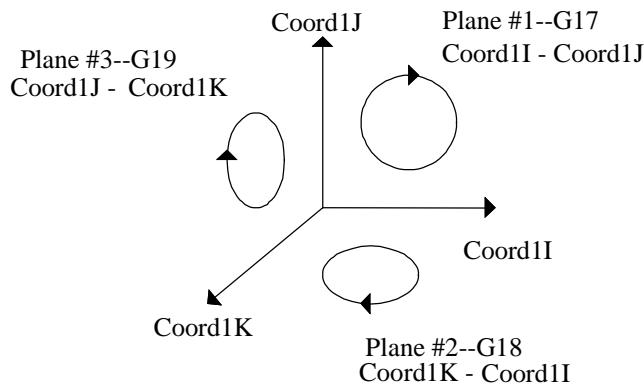


Figure 5-13. Coordinate System 1 (Clockwise or G2 motion)

5.9. Normalcy Motion Overview (G20, G21, G22)

Certain types of cutting tools require their orientation be perpendicular (normal) to the part being cut. Such tools typically mount to a rotary axis so that the orientation of the tool may be changed as the position of the part changes. Normalcy motion is generated from the commanded position (not feedback) of the Normalcy axes.

Figure 5-14 illustrates a part, where the arrows indicate the orientation of the normalcy axis at the time that the controller is on the point on the part where the head of the arrow is touching (Figure 5-14 shows normalcy left (G21)). The letters A, B, C, D and E denote different points on the part, referred to in the explanations below.

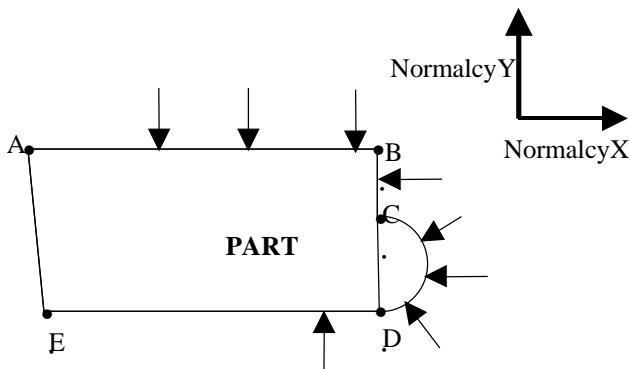


Figure 5-14. Tool Orientation

Although maintaining perpendicularity is possible using conventional G-code programming, it would be very cumbersome. The moves have to be broken up into small segments requiring extensive calculations to determine the appropriate rotary move distances and feedrates.

The UNIDEX 600 Series controller provides the normalcy mode, which keep the tip of the tool normal to the surface of the part being cut, alleviating the parts programmer from this duty.

In order to utilize this mode, the operator must supply several pieces of information. This includes the rotational axis of the cutting tool (B-axis) and the axes that make up the plane where the operator must maintain normalcy (XPlane/YPlane).

Normalcy motion will not be generated by Manual (MDI) command lines, normalcy motion is de-activated at the end of each MDI command line.



The homing cycle will disable normalcy motion.



While operating in the normalcy mode, the CNC automatically maintains the relationship between the B-axis (defined by the NormalcyAxis parameter) and the plane defined by the NormalcyX and NormalcyY task parameters. This requires two types of moves:

1. Normalcy Alignment Moves
2. Normalcy Concurrent Moves

Initial Normalcy Alignment

When Normalcy mode is activated, the normalcy axis will rotate to the required absolute angle determined by the active G21 / G22 mode, and by the controller looking for the next move within the CNC program. No initial alignment is required. This may however, require that the HomeOffsetDeg machine parameter be set to a non-zero value to align the tool to the angle that will be set by the initial normalcy alignment move.

For example, in Figure 5-15, when G21 is activated at Point A the Normalcy axis will rotate to the 270 degree absolute position. This is the same as any Normalcy Alignment Move.

Normalcy Alignment Moves

Once the tool achieves normalcy during a **G1** move, the remainder of the move does not require normalcy movement. However, at the start of a **G1** move, normalcy movement of the axis prior to the move may be required. For example, following the path of the part shown in Figure 5-14, the normalcy axis must rotate 90 degrees at Point B before proceeding down the next side. This is called a Normalcy Alignment Move; accomplished by a “Rapid” or **G0** move and moves at the speed specified by the *RapidFeedRateRPM* machine parameter. The direction of rotation of the Normalcy Alignment Move depends on whether you are in Normalcy Left (**G21**) or Normalcy Right (**G22**) mode.



When two moves are such that a Normalcy Alignment Move is required between them, the controller will force a **G9** between the moves so it can smoothly move the normalcy axis in-between the moves. However, the controller does not detect when the normalcy axis is required to change velocity abruptly as the result of blending two moves that move the normalcy axis during the move.



Normalcy Alignment Moves require look-ahead. There are some important issues the CNC programmer needs to be aware of in order to avoid unexpected results. See *CNC Block Look-Ahead* for details.

Avoiding Excessive Slow-Downs During Normalcy Mode

When two moves are non-tangential, such that a Normalcy Alignment Move is required between them (as in points B and C in Figure 5-14), the controller will force a G9 between the moves, so it can smoothly move the normalcy axis between the moves. However, this can cause unnecessary G9's between moves that are intended as tangential, but are very slightly non-tangential, because of the limited precision of the values entered in the G1's. For example, clearly, there should be no slowdown at Point D between move C to D, and move D to E (no normalcy axis alignment move needed). However, there may

be a slight angular difference, due to the precision of the numbers entered. Therefore, to allow the user to prevent this, the NormalcyAngleToleranceDeg parameter is provided. If an angle between two moves is less than this value, then the controller will not slowdown to do the normalcy axis alignment, but will instead make the alignment move during the first UpdateTimeSec of the second move. However, the user is warned that if the value of NormalcyAngleToleranceDeg is too high, it will “jerk” the normalcy axis too much during the alignment.

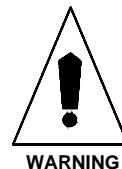
Normalcy Concurrent Moves

Unlike **G1** moves, circular moves (**G2** and **G3**) require that the normalcy axis move continuously throughout the move. These moves are called Normalcy Concurrent Moves, because the normalcy axis will be moved concurrently with the other two axes, as opposed to a Normalcy Alignment move. For example, following the path of the part in Figure 5-14, the normalcy axis must rotate 180 degrees, while traveling from Point C to D. The speed that the normalcy axis moves during a concurrent move is determined by the radius of the circle and the feedrate along the circular path. This is much like a complex move where the rotary and linear axes move and the linear axis is dominant. Meaning, the speed and acceleration of the rotary axis is slaved to the speed and acceleration of the linear axis. Therefore, many of the same problems can occur here, like those in a complex linear dominant move. This section just mentions how they differ from the cases discussed under the **G98** command.

In normalcy moves, the speed of the normalcy axis is not limited. The normalcy axis can easily be made to travel too fast.



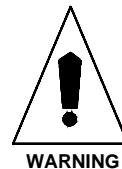
In normalcy concurrent moves, the normalcy axis acceleration and deceleration will be determined by the acceleration and deceleration of the linear axes. No acceleration/deceleration limits are applied, very easily causing the normalcy axis to travel too fast, exceeding its limitations.



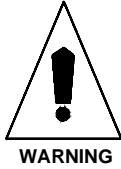
Normalcy only affects contoured motion, it has no effect on the **G0** moves.



The rotary axis cannot be moved explicitly with **G1/G2/G3** commands while normalcy mode is active (**G21/G22**). Doing so will lead to unexpected results.



You can verify the current state of the Normalcy mode, by viewing the 'NormalcyActiveLeft' and 'NormalcyActiveRight' bits of the task Status3 variable (use the AerStat.exe utility to do this), or, by looking for G21 and G22 in the active G code display in the lower left of the MMI 600 run or manual screens.



The controller does not detect when the normalcy axis is required to change velocity abruptly as the result of blending two moves (**G108**) that require the normalcy axis to rotate during the move. For example, in Figure 5-14, while moving from point C to point D, the normalcy axis is moving at some constant speed, so as to stay normal to the part. However, after point D, it should not be moving in order to maintain normalcy. But, if the user is in **G108** mode, then the controller will not decelerate to a stop between move B to C, and move C to D, – it will maintain a constant feedrate through the transition. Therefore, the normalcy axis will be forced to change to zero velocity instantaneously. This may cause unacceptable acceleration/deceleration, unless corner rounding mode is used.

MODAL

5.9.1. Disable Normalcy Mode

G20

SYNTAX: G20

EXAMPLE: G20

The **G20** command disables the normalcy mode of operation where the cutting tool is automatically kept perpendicular to the part being cut (normalcy mode). This command is the default.

Refer to the Normalcy Mode Overview for a general description of the implementation of this feature on the UNIDEX 600 Series controller.

MODAL

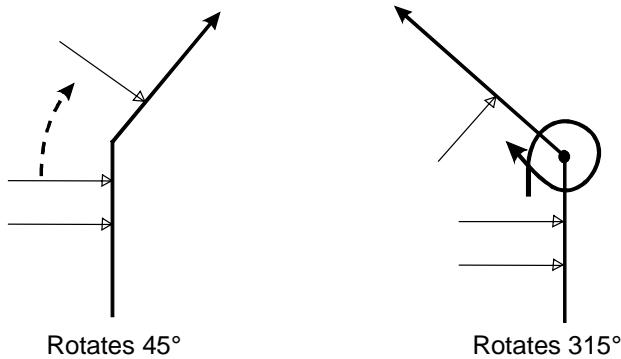
5.9.2. Activate Normalcy Mode Left

G21

SYNTAX: G21

EXAMPLE: G21

This command activates the mode of operation where the cutting tool is automatically kept perpendicular to the part being cut (normalcy mode) as shown in Figure 5-15.

**Figure 5-15. Normalcy Left**

Refer to the Normalcy Mode Overview for a general description of the implementation of this feature on the UNIDEX 600 Series controller. The active state of this command is indicated by bit 16 of the Status3 task parameter.

5.9.3. Activate Normalcy Mode Right

G22

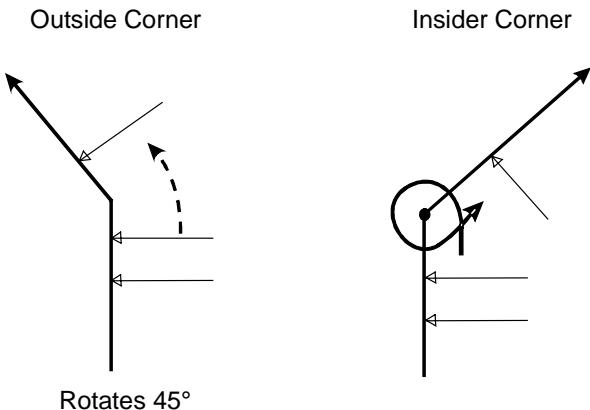
SYNTAX: G22

EXAMPLE: G22



This command activates the mode of operation where the cutting tool is automatically kept perpendicular to the part being cut (normalcy mode)(refer to Figure 5-16).

Refer to the Normalcy Mode Overview for a general description of the implementation of this feature on the UNIDEX 600 Series controller. The active state of this command is indicated by bit 17 of the Status3 task parameter.

**Figure 5-16. Normalcy Right**

Refer to the Normalcy Mode Overview for a general description of the implementation of this feature on the UNIDEX 600 Series controller.



5.10. Corner Rounding (G23, G24) G23

SYNTAX: **G23 F<fExpression>** ;where fExpression is the “rounding” time in seconds

G23 F<fExpression> ;where fExpression is the “rounding” time in seconds

In corner rounding mode, all normal acceleration/deceleration is disabled, and instead acceleration and deceleration are controlled by the VELTIMECONST axis parameter. Corner rounding is applied to all motion, including homing and camming motion.

The F (or P) parameter, must be a positive value, which sets the VELTIMECONST axis parameter, for all axes owned by the task. The F (or P) value has a resolution of .001 seconds. Specifying an F (or P) word value of less than .001 seconds, forces all motion to have instantaneous acceleration and deceleration. When a subsequent **G24** is executed, the VELTIMECONST axis parameter is restored to its value prior to the **G23** command, disabling corner rounding mode.

Additionally, when corner-rounding mode is activated it takes precedence over the acceleration/deceleration, producing motion as if a **G8** command were on every CNC program line, with the value specified in the **G23** command solely determining the acceleration/deceleration time. The corner-rounding mode temporarily overwrites the current values for the VELTIMECONST, ACCEL, DECEL, ACCELMODE and DECELMODE axis parameters. When corner rounding is disabled, the original values for these axis parameters will be restored.



WARNING

Do not change values for these axis parameters when corner-rounding mode is active or you will overwrite the original values.

You can verify the current status of corner rounding by viewing the “CornerRounding” bit of the task Status3 variable (use the AerStat.exe utility), or by looking for **G23** or **G24** in the active G code display in the lower left of the U600MMI-NT/95 run or manual screens.

5.10.1. Disable Corner Rounding Mode G24



SYNTAX: **G24**

Disables the **G23** corner-rounding mode. No parameters are required (see **G23** for details). Restores the values for VELTIMECONST and ACCEL/DECEL axis parameters stored by the **G23** command.

5.11. Coordinate System #2 Configuration (G26 – G29)

G26 through **G29** codes apply to plane 2 in the same way that **G16** and **G17** through **G19** function on coordinate system 1. Coordinate system 2 is used only in context with the **G12** and **G13** commands, and determines the axes comprising the second coordinate system (#2) XYZ plane where the **G12** or **G13** commands produce circular motion. Refer to **G16**, and **G17** through **G19** commands for more details on **G26**, **G27**, **G28**, and **G29**.

5.11.1. Assign Coordinate System #2 Axes

G26

SYNTAX: **G26~<axisMask>**

except: the user must supply exactly 3 axis letters (order of the letters is significant).

EXAMPLE: **G26 U V W**



This G-code is equivalent to assigning to the task parameters: Coord2I, Coord2J and Coord2K simultaneously. This G code determines the axes comprising the second (#2) XYZ plane where the **G12** or **G13** commands produces circular motion. Coord2I is assigned to the first axis in the mask, Coord2J is assigned to the second axis in the mask, and so on. This command is the **default**.

By default:

<i>Coord2I</i>	is 3	(U task axis)
<i>Coord2J</i>	is 4	(V task axis)
<i>Coord2K</i>	is 5	(W task axis)

5.11.2. Plane Selection Codes for Coordinate System #2 G27/G28/G29



SYNTAX: **G27** or **G28** or **G29**

EXAMPLE: **G27**

These G-codes are equivalent to **G17**, **G18**, and **G19**, respectively, but for plane 2 instead of plane 1. Refer to the descriptions for **G17/G18/G19** for details. **G27** selects Plane 1, **G28** selects Plane 2, and **G29** selects Plane 3 (refer to Figure 5-17). The active plane is indicated by bits 26-28 of the Status3 task parameter.

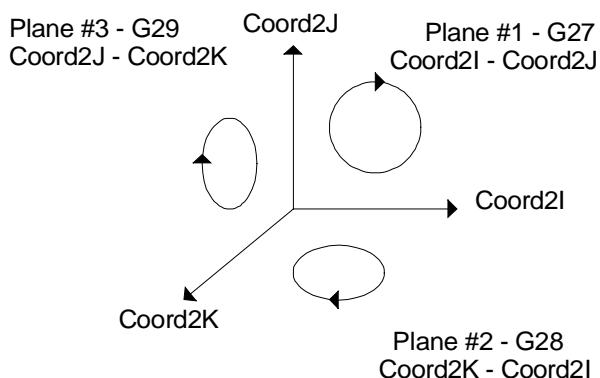


Figure 5-17. Coordinate System 2 Orientation (Clockwise or G2 Motion)

5.12. Software Limits Overview

Software limits effectively allow the hardware end of travel limits to be reduced to a range specified by the user. They are very similar to safe zones.

5.12.1. Configuring Software Limits

To enable Software Limits, they must be able to generate a fault. This requires the software limit bit be set within the axis FAULTMASK, additionally the software limit bit must also be set in one of the other faultmasks for the condition the fault will cause to occur. See configuring faultmasks for more information.

The minimum (more negative) software limit value must be defined in machine counts, via the CCWEOT axis parameter.

The minimum (more positive) software limit value must be defined in machine counts, via the CWEOT axis parameter.

The software limit action must be defined relative to homing, via the SOFTLIMITMODE axis parameter.

5.13. Safe Zones (G34, G35, G36, G37)

In some applications, it is desirable to define a particular area where all the axes' motion may occur. Conversely, other applications require the ability to define an area where axes are not permitted to enter. The UNIDEX 600 Series controllers Safe Zone feature provides the user with the ability to perform either of these two functions. Safe zones are similar to software limits.

Although multiple axes may be used in the specification of the safe zone, each axis is evaluated individually. A more sophisticated version of safe zones may be implemented via a user CNC program, such as this:

Multi-Dimensional Safe Zones

Note: Axis speeds will be limited by the scan rate of this program!

```
; Dynamic Safe Zone Example
;

; Be sure the safe zones are configured
; Be sure to then set the safe zone fault bit in the X & Y ABORTMASK axis parameters !
; See the AvgPollTimeSec global parameter for info. on determining the scan rate
; of this program.
; A safe zone fault will require moving the Z axis out of range, to clear the fault.

#define OFF          0      ; disable safe zones
#define DONT_ENTER   2
#define DONT_EXIT    1

SOFTLIMITMODE.X = 1      ; must home axes before soft limits become active
SOFTLIMITMODE.Y = 1

WHILE( 1 )               ; forever
; for each position of an axis, Z in this example
IF( PositionUnits.Z > 2.0 )

    ; limit the X axis
    SAFEZONECCW.X = -1 * CntsPerInch.X ; machine counts
    SAFEZONECW.X  = 1 * CntsPerInch.X ; machine counts
    SAFEZONEMODE.X = DONT_ENTER

    ; limit the Y axis
    SAFEZONECCW.Y = -2 * CntsPerInch.Y ; machine counts
    SAFEZONECW.Y  = 2 * CntsPerInch.Y ; machine counts
    SAFEZONEMODE.Y = DONT_ENTER

ELSE IF( (PositionUnits.Z > .5) AND (PositionUnits.Z < 2.0) )

    ; limit the X axis
    SAFEZONECCW.X = -4 * CntsPerInch.X ; machine counts
    SAFEZONECW.X  = 3 * CntsPerInch.X ; machine counts
    SAFEZONEMODE.X = DONT_ENTER

    ; limit the Y axis
    SAFEZONECCW.Y = -3 * CntsPerInch.Y ; machine counts
    SAFEZONECW.Y  = -2 * CntsPerInch.Y ; machine counts
    SAFEZONEMODE.Y = DONT_ENTER

;ELSE IF
; other conditions

ELSE
; disable any active safe zone
SAFEZONEMODE.X = OFF
SAFEZONEMODE.Y = OFF
ENDIF
ENDWHILE
```

5.13.1. Set Safe Zone Minimum Values**G34**

SYNTAX: G34~<axisPoint>
G34

EXAMPLE: G34 X5 Y5

This command defines the minimum safe zone value for the axes specified in the Axis Point. The values must be specified in inches. The value specified is translated to machine counts and entered into the appropriate axis SAFEZONECCW axis parameter.

5.13.2. Set Safe Zone Maximum Values**G35**

SYNTAX: G35~<axisPoint>
G35

EXAMPLE: G35 X5 Y5

This command defines the maximum safe zone values for the axes specified in the Axis Point. The values must be specified in inches. The value specified is translated to machine counts and entered into the appropriate axis SAFEZONECW axis parameter.

5.13.3. Enable Safe Zones**G36**

SYNTAX: G36~<axisMask>
G36

EXAMPLE: G36 X v

The **G36** command enables a safe zone on the specified axes. Once enabled, safe zone testing is performed prior to the execution of each motion command. A safe zone axis fault (See Axis Fault Mask Parameter Bit Definitions) occurs if the user attempts to command motion which violates the safe zone restrictions. For more information on safe zone fault handling, refer to the SAFEZONECCW and SAFEZONECW axis parameters.



Although multiple axes may be used in the specification of the safe zone for the P1 and P2 types, each axis is evaluated individually. See Figure 5-18.

EXAMPLE PROGRAM:

SAFEZONEMODE.X = 2	; X Area below will be 'cant enter' zone.
SAFEZONEMODE.Y = 2	; Y Area below will be 'cant enter' zone.
G34 X11 Y8	; Set safe zone minimums
G35 X13 Y9	; Set safe zone maximums
G36 X Y	; Activate safe zones

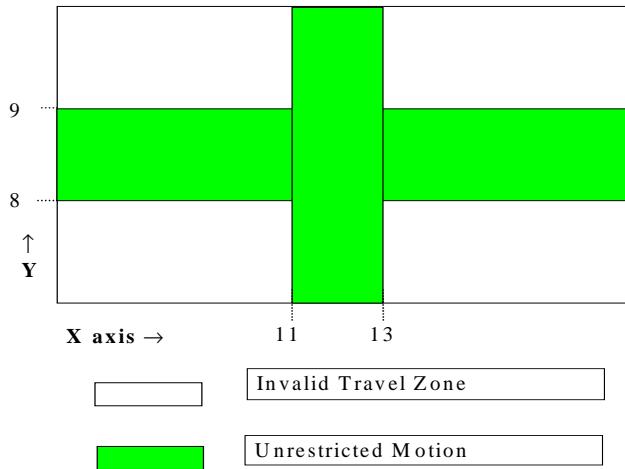


Figure 5-18. Unrestricted Safe Zones

Conversely, a **SAFEZONEMODE=1** safe zone is used to specify a multi-dimensional area where a set of axes may not travel. The user may not command motion through that area (refer to Figure 5-18).

Safe zone boundaries are evaluated only when computing the target position of a motion command. Therefore, if an axis is within the restricted area when a safe zone is enabled, a safe zone fault is not generated until the first commanded move. Furthermore, if the target position of that first motion command is outside the restricted area, the move executes normally.

In order for the safe zone feature to operate as described, the *SafeZone* bit in the **FAULTMASK** axis parameter must be set for that axis (see axis faults).



The safe zone parameters are interpreted as relative displacements if the CNC is in the **G91** mode when the **G36** command executes.



5.13.4. Disable Safe Zones

G37

SYNTAX: **G37~<axisMask>**
G37



EXAMPLE: **G37 X v**

The **G37** command disables safe zones for the specified axes. As with the enable safe zones command (**G36**), this command accepts parameters to permit the user to specify the

axes this command applies. Therefore, safe zones can be disabled individually or in groups.

This command is the **default**.

EXAMPLE PROGRAM:

G37 X Y Z	;Disable the safe zones active for the X, Y and Z axes
-----------	--------------------------------------------------------



The default operational mode of the CNC has safe zones disabled.

5.13.5. Safe Zone Activation

Assuming the axis faultmasks have been properly configured:

1. Activating safe zones while an axis is within a safe zone will generate a fault.
2. An axis crossing the boundary of a safe zone, will generate a fault.
3. Changing the range of the safe zone, will generate a fault if an axis is within the new safe zone.

Note: Although multiple axes may be used in the specification of the safe zone for the P1 and P2 types, each axis is evaluated individually.

5.13.6. Configuring and Using Safe Zones

To enable Safe Zones, they must be able to generate a fault based upon the Safe Zone. This requires the safe zone bit be set within the axis FAULTMASK, additionally the safe zone bit must also be set in one of the other faultmasks for the condition the fault will cause to occur. See configuring faultmasks for more information.

The minimum (more negative) safe zone value must be defined, via the **G34** command or the SAFEZONECCW axis parameter.

The maximum (more positive) safe zone value must be defined, via the **G35** command or the SAFEZONECW axis parameter.

The safe zone action must be defined relative to homing, via the SOFTLIMITMODE axis parameter.

The safe zone mode of operation must be defined via the SAFEZONEMODE axis parameter.

Safe zones must be enabled via the **G36** command.

Safe zones are disabled via the **G37** command.

5.14. Backlash Compensation (G38, G39)

5.14.1. Enable Backlash Compensation

G38



SYNTAX: G38~<axisPoint>
G38

EXAMPLE: G38 X5 Y5

The **G38** command activates backlash compensation on the specified axes, with the specified values for each axis. It is equivalent to setting the REVERSALMODE axis parameter for each specified axis to the specified value. The values must be specified in inches.

See the REVERSALMODE axis parameter for more details on backlash compensation (See the UNIDEX® 600 Series User's Guide, P/N: EDU157, Appendix C).

Backlash compensation is disabled via the **G39** command.

5.14.2. Disable Backlash Compensation

G39



SYNTAX: G39~<axisMask>
G39

EXAMPLE: G39 X Y

The **G39** command deactivates backlash compensation on the specified axes. It is equivalent to setting the REVERSALMODE axis parameter for each specified axis to zero. See the REVERSALMODE axis parameter for more details on backlash compensation (See the UNIDEX® 600 Series User's Guide, P/N: EDU157, Appendix C).

This command is the **default**.

Backlash compensation is enabled via the **G38** command.

5.15. Cutter Radius Compensation (G40, G41, G42, G43, G45)

While machining a part, it is sometimes necessary to consider the radius of the cutting tool. For example, when a tool cuts a part, the center of the tool follows the programmed path. The outside edge of the tool cuts the actual part, offset from the programmed path by the tool's radius.

Intersectional Cutter Radius Compensation (ICRC) is a feature that allows the operator to program the path along the outside edge of the tool without regard to its size. Without this feature, the operator would have to offset the actual part dimensions based on the radius of the tool.

This feature significantly decreases the programming effort required for this type of application. It also allows the user to run the same program with tools of different diameters by simply changing the tool diameter information.

Cutter compensation is only intended with contoured moves (**G1**, **G2**, **G3**, **G12**, **G13**). You may command non-contoured motion on axes that are not part of the cutter radius compensation plane while in cutter compensation mode, however, this will lead to undesirable results, since cutter compensation has no effect on the other types of motion.

Cutter Radius Compensation requires CNC block look-ahead, to find the next contoured Move, to compensate for inside and outside corners (see below). If commands other than contoured motion commands are executed while cutter compensation is active, the controller may not be able to find the next contoured motion statement, and therefore may not follow the desired path. The CNC Block Look-Ahead Failures section describes these situations.

Cutter Radius Compensation will not operate over multiple lines entered in MDI mode, as after each MDI line, the cutter compensation is canceled.

Homing an axis will disable cutter radius compensation

The user must provide a tool radius with the **G43** command (or by setting the task parameter *CutterRadiusInch*). The user must also provide the cutter compensation axes representing the plane where the compensation will be performed in the **G44** command (this can be done with the task parameters *CutterX* and *CutterY* as well). The first axis (*CutterX*) in the **G44** command is called the horizontal axis; the second axis (*CutterY*) is called the vertical axis. Two circumstances of interest in cutter compensation are the inside and outside corners, shown in Figure 5-19.

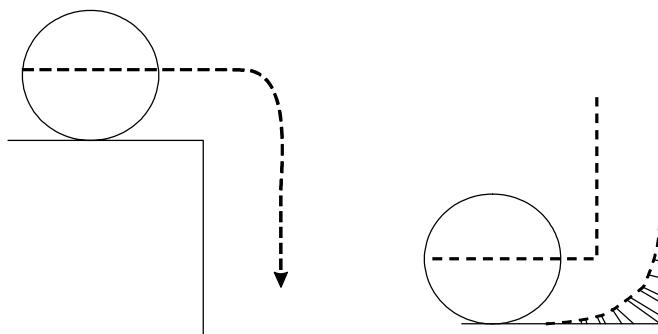
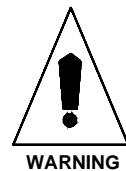


Figure 5-19. Cutter Radius Compensation Path

Inside corners and outside corners both have their limitations, whereby the tool will not stay inside the part. This is because cutter radius compensation “sees” only the next move when determining the tool placement.



Cutter compensation can be used during parts rotation (**G84**), mirroring (**G83**), or normalcy (**G21/G22**).

5.15.1. CNC Block Look-Ahead Requirements in Cutter Compensation Mode

In the process of computing targets in cutter compensation paths, the controller must know the next move in the sequence. In Figure 5-20, the cutter must stop short of the actual target during the horizontal move, so that it can follow the correct path (solid line) during the vertical move. However, if the controller fails to find the next move, it will follow the incorrect (dashed line) path. The process of finding the next move is called CNC block look-ahead and there are important limitations that need to be conformed to allow the look-ahead process to work properly.

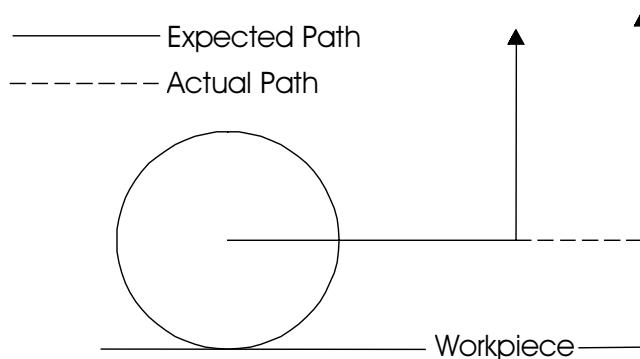


Figure 5-20. Cutter Compensation with Intervening Statements

5.15.2. Cutter Radius Compensation Lead-On and Lead-Off Moves

Normally, when entering (**G41**, **G42**) and exiting (**G40**) Cutter Compensation, you provide a contoured move on the same line. For **G41**, **G42**, this move is called the “lead-on” move, for **G40** this is called the “lead-off” move. The lead-on and lead-off moves must be a contoured move.

Lead-on and lead-off moves must be carefully constructed, so as not damage the tool or part. Figure 5-21 is an example of a safe lead-on move. See **G40** for an example of a safe lead-off move.

Lead-on moves will ‘blend in’ the tool radius gradually during the move, in order to compensate for the tool radius, on the correct side of the part. When a lead-on move is completed, the cutter radius is fully compensated for.

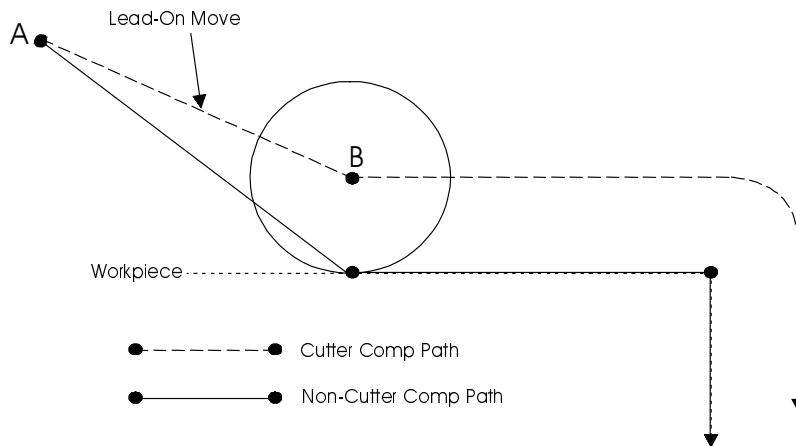


Figure 5-21. Cutter Radius Compensation Lead-On Moves

5.15.3. Interaction of Mirroring and Cutter Compensation Commands

When mirror mode (G83) is active, cutter compensation left becomes cutter compensation right, and vice versa.

5.15.4. Cutter Compensation Limitations within Inside Corners

Figure 5-22 below, illustrates that for inside corners, the controller will not generate a link move. Be aware that the horizontal move will stop short of the actual target (because of the tool radius) and a small circular wedge of uncut material will remain in the inside corner.

NOTE: Material left “uncut” in shaded section
 NOTE: Limitations, as discussed below.

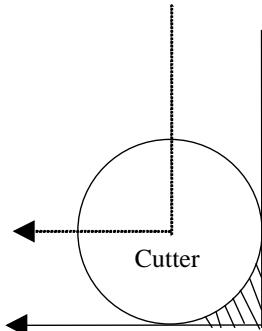
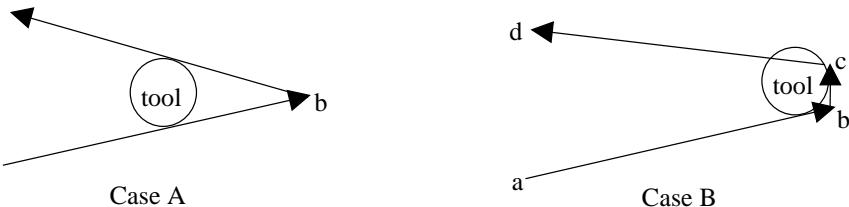
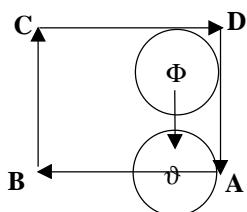


Figure 5-22. Inside Corner

Inside Corner Situation 1: This condition is a form of over-cutting (cutting too much material away). It occurs because cutter compensation only looks-ahead one move. In the figure below, case A illustrates that the tool stays inside the part, and case B where it does not. In case B, the tool does not see the move from point c to point d, when placing the cutter after the move from point a to point b. This situation, in general, will occur when the length of a move (such as from point B to C) is shorter than the cutter (tool) radius. Note that this form of over-cutting will usually generate an Over-Cutting warning. However, not all cases of over-cutting will trigger this warning. The example below will.



Inside Corner Situation 2: Another, more important situation where the tool will not stay inside the part can occur, due to cutter compensation not seeing previous moves.

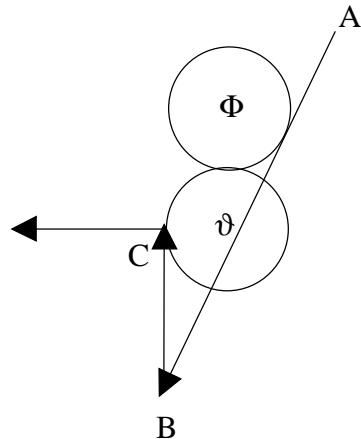


Note that this situation, like the Inside Corner Situation #1, may trigger an overcutting warning. The above example above will trigger such a warning.

The user programs the path, from points, A to B to C to D and back to point A, before withdrawing the tool, while in cutter comp right (**G42**). After executing segment C to D the cutter lies in position Φ , as expected. But, after segment D to A, the cutter lies in

position ϑ , which moves through the part. This is because the controller does not see any piece of the part after move D to A. The solution is to move towards B (from A) some small amount, after the move from point D to A. Then you will stay within the part.

Any previous move may cause this problem, so this situation can occur in different forms. The diagram below shows the problem occurring for a severe inside corner.



Here the cutter moves to position Φ (we are in cutter right or G42), after move A to B. But after move B to C, it positions itself at position ϑ , and does not see that it breaks through the previous move, move A to B.

Inside Corner Situation 3: Another form of overcutting on an inside corner can occur if an arc on the part has a shorter radius than the cutter radius. If this occurs, an over-cutter warning is generated.

5.15.5. Cutter Compensation within Outside Corners

Figure 5-23 illustrates that the controller generates a link move (in this case, 90 degrees), to move around the outside corner. The speed of this move is the same as the last move (down the left edge) and there will be no deceleration between the user provided move and the auto-generated circular arc. However, if the speed along the circular link move must be limited due to the MaxFeedRateIPM / BlendMaxAccelLinearIPS2 machine / task parameters, then the previous move will be limited in the same way.

NOTE: Two possibel paths, see CutterToleranceDeg

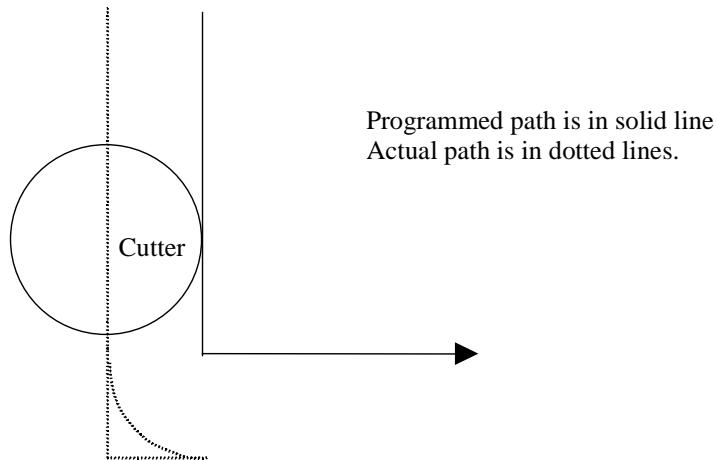


Figure 5-23. Outside Corner (Diagram A)

5.15.6. Deactivate Cutter Compensation (ICRC)

G40

SYNTAX: G40

EXAMPLE: G40



The **G40** mode exits cutter radius compensation mode. If you are not in cutter radius compensation mode, this command is ignored. You must be in the **G1** mode to execute this command.

Normally, when entering exiting (**G40**) Cutter Compensation, you provide a contoured move on the same line, this is called the “lead-off” move.

Lead-off moves must be carefully constructed, so as not damage the tool or part. The diagram below is an example of a safe lead-off move.

Lead-off moves will ‘blend out’ the tool radius gradually during the lead-off move, in order to remove compensation for the tool radius. If you do not provide a “lead-off” move on the same line as the **G40**, then, it will use the next contoured move it executes, as the lead-off move. Any number of CNC statements that are not contoured moves (except **G41**, **G42**, **G40**, **G143**, **G144**, and **G149**) can be placed in-between the **G40**, and the “lead-off” move. If it cannot find a next contoured move to use as a lead-off, it will not remove the tool radius compensation.

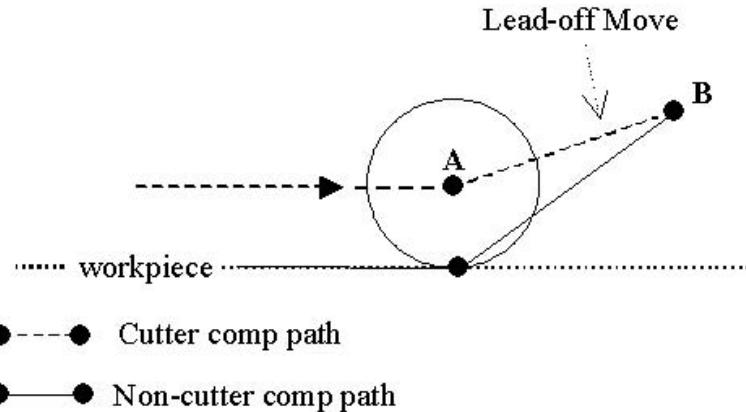


Figure 5-24. Lead Off Moves

You must be careful when programming lead-off moves. By the time the move reaches the target of the lead-off, the cutter radius is “blended out” of the move. Therefore, in the example above, the distance between target point B and the workpiece must be greater than the cutter radius or the lead-off move will actually move the tool downwards into the workpiece.

EXAMPLE PROGRAM:

```
G40 G1 X1. Y1. F100. ;Deactivate the cutter radius compensation
;and remove the offset during the end move
```

5.15.7. Activate ICRC Left

G41



SYNTAX: **G41** [[*Lead-In Move*]]

EXAMPLE: **G41**

The **G41** command activates Intersectional Cutter Radius Compensation (ICRC) to the left of the programmed tool path relative to the direction of tool motion, (refer to Figure 5-25). The center of the tool nose is kept on a line normal to the programmed path until ICRC is deactivated. If you are already in cutter compensation mode, this command is ignored. The lead-on move, is defined as the next contoured move on the same line as the **G41**, or on the line following the **G41**. You must be in the **G1** / **G2** or **G3** mode to execute this command. The active state of this command is indicated by bit 13 of the Status3 task parameter.

If you do not provide a “lead-on” move on the same line as the **G41/G42**, then, it will use the next contoured move it executes, as the lead-on move. Any number of CNC statements that are not contoured moves (except **G41**, **G42**, **G40**, **G143**, **G144**, and **G149**) can be placed in-between the **G41/G42**, and the “lead-on” move. If it cannot find a next contoured move to use as a lead-on, it will not enter cutter radius compensation.

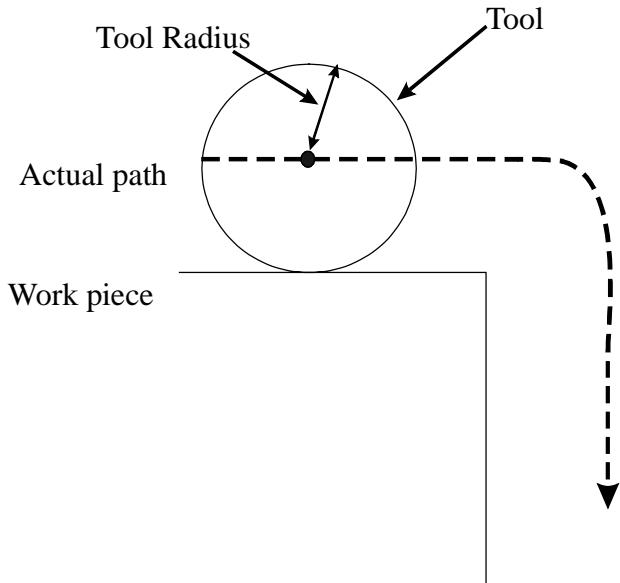


Figure 5-25. Path Compensation Left

Refer to the Intersectional Cutter Radius Compensation Overview for a general description of the implementation of this feature on the UNIDEX 600 Series controller.

5.15.8. Activate ICRC Right

G42

SYNTAX: **G42** [[*Lead-In Move*]]

EXAMPLE: **G42**



The **G42** command activates Intersectional Cutter Radius Compensation (ICRC) to the right of the programmed tool path relative to the direction of tool motion (refer to Figure 5-26). The tool radius will be incorporated into the execution of the next contoured motion command. The center of the tool nose will then be kept on a line normal to the programmed path until ICRC is de-activated. If you are already in cutter radius compensation mode, this command is ignored. The lead-on move, is defined as the next contoured move on the same line as the **G42**, or on the line following the **G42**. You must be in the **G1 / G2** or **G3** mode to execute this command. The active state of this command is indicated by bit 14 of the Status3 task parameter.

If you do not provide a “lead-on” move on the same line as the **G41/G42**, then, it will use the next contoured move it executes, as the lead-on move. Any number of CNC statements that are not contoured moves (except **G41**, **G42**, **G40**, **G143**, **G144**, and **G149**) can be placed in-between the **G41/G42**, and the “lead-on” move. If it cannot find a next contoured move to use as a lead-on, it will not enter cutter radius compensation.

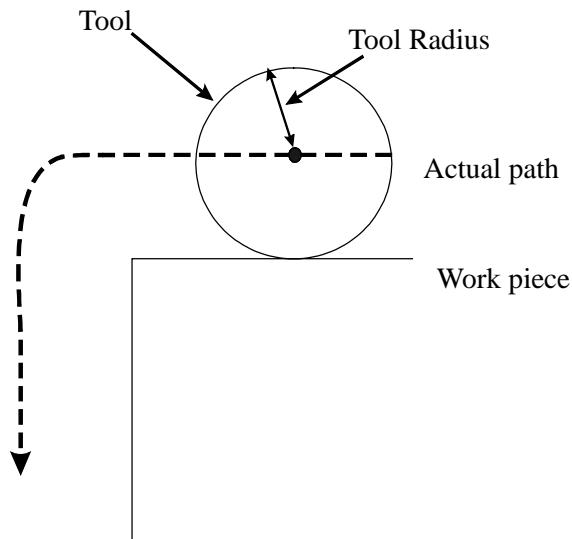


Figure 5-26. Path Compensation Right

Refer to the Intersectional Cutter Radius Compensation Overview for a general description of the implementation of this feature on the UNIDEX 600 Series controller.

MODAL

5.15.9. Set Cutter Compensation Radius

G43

SYNTAX: **G43 F<fExpression>** ; Where the expression is the tool radius
G43 P<fExpression > ; Where the expression is the tool radius

EXAMPLE: **G43 F.001**

G43 P.01

The **G43** command sets the *CutterRadiusInch* task parameter for the current task. The unit of measure associated with this radius is inches, unless bit 4 is set to one in the *CompatibilityMode* global parameter, in which case it is user units. The **F** or **P** keyword may specify the tool radius.

Use this command only when cutter compensation is not active. Refer to the **G40** Overview for more information.



A **G44** must be executed before a **G43**.



You may alternately perform the same function as the **G43**, through the use of tool filesT_Word, or by setting the task parameter: CutterRadiusInchCutterRadiusInch directly.

5.15.10. Set Cutter Compensation Axes

G44

SYNTAX: **G44 <axisMask>** ; Order of the axis in the mask is significant
except: the axis mask must contain two or three axis letters.

EXAMPLE: **G44 X v**

G44 X Y Z

The **G44** command defines the axes comprising the X and Y plane, to be used for cutter radius compensation, and/or cutter offset compensation.



The order of the axes specified is very important. The first two axes must match the order specified by the circular plane specification, or the results will be unexpected. The first and second axes in the **G44** command must also be the first and second axes specified in the **G17**, **G18**, **G19** command. **G44** by itself, causes the *CutterX* and *CutterY* task parameters to define the axes. Use Table 5-4 to determine the correct axis order for the **G44** command.

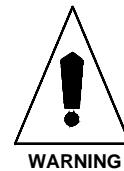


Table 5-4. Required Order of Axes in a G44, when no G16 has been Executed

Circular Plane Mode	G17	G18	G19
G44 Order	G44 X Y	G44 Z X	G44 Y Z

The optional, third axis is used to define the “Z” or tool length axes. This axis, if provided in the **G44** command, is used to compensate for a tool length supplied in a tool file, and activated by a **G143**.

You may alternately perform the same function as the **G44** by setting the task parameters: *CutterX*, *CutterY*, and *CutterZ* directly.



5.16. Polar/Cylindrical Transformations (G45, G46, G47)

5.16.1. Disable Polar or Cylindrical Coordinate Transformation G45



The **G45** command disables the polar or cylindrical coordinate transformations. This command is the **default** — that is, both polar and cylindrical conversions are disabled. You may enable either polar transformations (**G46**) or cylindrical transformations (**G47**), but not both at the same time. See **G46** or **G47** for more details.

SYNTAX: **G45**

EXAMPLE: **G45**

See the example under the **G46** command.

5.16.2. Enable Polar Coordinate Transformation G46



SYNTAX: **G46~<axisMask>**

except: the axis mask must contain exactly two axis letters.

EXAMPLE: **G46 d v** ; Sets d to be the radial axis, v to be the angular axis

The **G46** command enables a transformation from an X/Y Cartesian axis plane to a polar coordinate system. **G45** disables the polar coordinate transformation. The **G46** command requires the use of the **G52** command to define the axes, as described below and in Table 5-5.

The polar coordinate system is comprised of a linear positioning device holding the tool and a rotary device that holds the part centered about its axis of rotation. Part programming is in Cartesian coordinates via the **G1/G2/G3** commands acting upon virtual horizontal and vertical axes, as specified in the **G52** command. The controller translates these virtual coordinates to polar coordinates {r, theta}, which are then used as position commands for two real axes, specified by the **G46** command.

The first axis specified in the **G46** command is the radial axis, the second is the angular or rotary axes. The first axes specified in the **G52** command is the horizontal axis and the second axis is the vertical axes (see diagram below).

Table 5-5. Transformation from an X/Y Cartesian Plane to a Polar Coordinate System

Function	Configured	Assigned	Name In Figure 5-27
Horizontal axis	virtual	1st letter in G52 command	X
Vertical axis	virtual	2nd letter in G52 command	Y
Radial axis	To linear table	1st letter in G46 command	U
Angular axis	To rotary stage	2nd letter in G46 command	C

You cannot generate asynchronous motion (or jog) while this mode is active.

The polar axes coordinates at the time of executing the **G46** are interpreted as offsets to be applied to these axis. In other words, the polar axes position commands {r, th} are given by the below equations, where h and v represent the horizontal and vertical Cartesian coordinates, and where r_0 and th_0 represent the positions of the polar axes when the **G46** is first executed:

$$\begin{aligned} r &= r_0 + \sqrt{h^2 + v^2} \\ \text{th} &= \text{th}_0 + \arctan(v/h) \end{aligned}$$

You may verify the current state of the Polar transformation mode, by viewing the ‘RThetaPolarActive’ bit 1 of the status3 task variable (use the AerStat.exe utility), or by looking for **G46** in the active G code display in the lower left of the MMI run or manual screens.

Cylindrical coordinate transforms (**G47**) cannot be used in the **G46** mode.



Cutter compensation may be used in the **G46** mode. The virtual Cartesian axes must be and the real radial polar axis must be a linear Type axis.



Motion through the “origin” (Cartesian axes = {0,0}) should not be attempted because, it will cause a discontinuous step in the angular coordinate position command.



Only **G1/G2/G3** commands are valid within the polar coordinate transformation, **G0** commands or any other asynchronous motion command should not be used when **G46** is active.

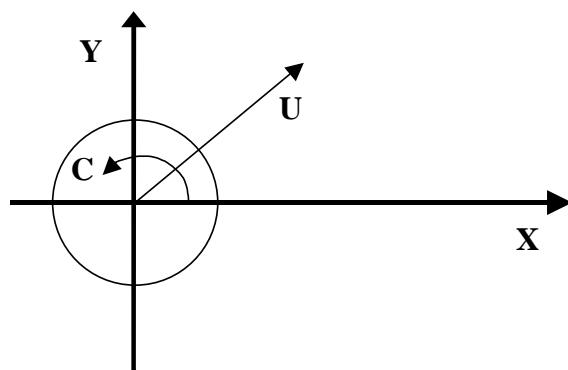


Figure 5-27. Polar/Cylindrical Transformations Diagram

Figure 5-27 shows the geometry following the execution of a “G52 X Y”, and a “G46 U C” command.

The following example illustrates a 4 inch/mm square part with rounded corners (1.0 inch/mm radius) centered at the X/Y origin. Contact with the part will occur at X=2.0, Y=0.0. The rotary axis in the polar coordinate system is C and the linear axis is U.

EXAMPLE PROGRAM:

```
; Draw a 2 inch square with rounded corners
G70 G90 F100          ; Absolute mode, set feedrates
G0 X0 Z0 U0 C0          ; "home"
G9 G1 U20 C5          ; Real axes offsets (5 deg, 20 inches)

; Coordinate system setup
G16 X Y Z             ; Define IJK for circles (I=X, J=Y, K=Z)
G18                   ; Use IK plane for circles
G52 X Z               ; X, Z are virtual axes (horz/vert)
G46 U C               ; U is RADIAL axis, C is ANGULAR axis

; Part
G9 G1 X2.0 Z0.0

G91 G1 Z1.0
G2 X-1.0 Z1.0 I-1.0 K0
G1 X-2.0
G2 X-1.0 Z-1.0 I0 K-1.0
G1 Z-2.0
G2 X1.0 Z-1.0 I1.0 K0
G1 X2.0
G2 X1.0 Z1.0 I0 K1.0
G9 G1 Z1.0             ; Back to starting point

G9 G1 X1.0             ; Move off of the part
G45                   ; Disable polar coordinate transform
```

EXAMPLE PROGRAM:

```
G90 G1 X2 Y0 U0 C2      ; ends at U0° C2
G52 X Y
G46 U C
G90 G1 X0 Y1             ; ends at U90° C1
```

5.16.3. Enable Cylindrical Coordinate Transformation

G47

SYNTAX: **G47~<axisMask>F~<radius>**

except: the axis mask must contain exactly one axis letter.

EXAMPLE: **G47 X F.05**



The **G47** command enables a transformation from the X/Y Cartesian axis plane into a cylindrical (C, Y) coordinate system. The **G45** command disabled the cylindrical transformation. The **G47** command requires the use of the **G52** command to define the axes, as described below (and in Table 5-6).

The cylindrical coordinate system is comprised of a rotary axis holding the part to be machined (the C axis, in the diagram below) and a linear axis (the X axis, in the diagram below) that moves parallel to the center of rotation of the rotary axis. An additional axis perpendicular to the center line of rotation of the rotary axis may be present for positioning a tool in the proximity of the part to be machined, however this axis is not independent of, and not controlled by the coordinate transformation. Part programming is via the **G1/G2/G3** commands acting upon the linear (X axis, in the diagram below) and a virtual Y axis.

In summary there are three axes functions involved in cylindrical coordinates, as shown in the table below. Note that any particular axis can be assigned to any of the four axis functions. In the table and diagram below we have chosen as example: X, Y, and C. You command your motion using the axial and circumferential axes (must be configured as virtual), but the actual motion is performed over the axial and rotational axes.

Table 5-6. Transformation from an X/Y Cartesian Plane to a Cylindrical Coordinate System

Function	Configured	Assigned	Name in Figure 5-28
Axial axis	To linear axis	1st letter in G52 command	X
Circumferential axis	Virtual	2nd letter in G52 command	Y
Rotational axis	To rotary axis	1st letter in G46 command	C

The controller will compute the appropriate distance to move the real C and X axes. All subsequent Y axis position commands refer to circumferential distances around a part of the specified radius according to the following relationship:

$$\text{C_axis_command} = \text{Y_axis_commanded_position} * 360 / (2 * \pi * \text{current_radius})$$

Where r is the radius specified in the **G46** command (or equivalently the *RThetaRadiusInch* task parameter). The specified radius will be in user units, defined by the **G70/G71** modal command group, unless, its bit is set in the *CompatibilityMode* global parameter, in which case the radius will be in inches.

The X/Y axis plane is defined by the **G44** command (or equivalently the *RThetaX* and *RThetaY* task parameters). The C axis is defined in the **G47** command (or equivalently by

the *RThetaT* task parameter). The radius of the cylinder is also provided in the **G47** command (or equivalently the *RThetaRadiusInch* task parameter).

Polar coordinate transformations (**G46**) cannot be used in the **G47** mode. You cannot generate *asynchronous motion* while this mode is active.

You may verify the current state of the Cylindrical transformation mode, by viewing the ‘RthetaCylindricalActive’ bit 2 of the Status3 task variable (use the AerStat.exe utility), or by looking for ‘**G47**’ in the active G code display in the lower left of the MMI run or manual screens.



Only **G1/G2/G3** commands and cutter compensation generated motion are valid under this mode of operation. **G0** commands or any other type of axis motion should not be attempted when the cylindrical coordinate transformation is active.

Figure 5-28 is an illustration of the relationship between the X, Y, rotational, and optional infeed axis.

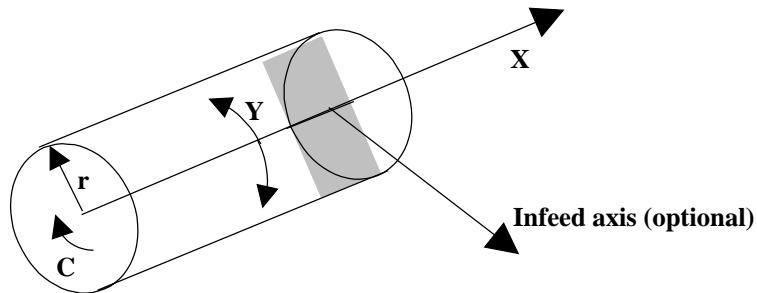


Figure 5-28. X, Y, Rotational and Optional Infeed Axis

The example program that follows illustrates the commands required to enable and disable cylindrical coordinate transformation. The axis designations used in this example are consistent with Figure 5-28.

EXAMPLE PROGRAM:

G52 X Y	; Define X axial axis, Y tangential axis
G47 C F3.0	; Enable cylindrical coordinates
G9 G1 X2.0 Y2.0	; Perform axis motion
.	;
.	; G1/G2/G3 motion commands
.	;
G45	; Disable cylindrical coordinates

EXAMPLE:

- ; To cut 90 degrees of thread, ascending in Clockwise on the X axis,
- ; looking up the X axis, on a 3 inch radius cylinder,
- ; with a pitch of .1 inch

G52 X Y

G47 C F3

; cylinder radius = 3

G91 G1 Y4.712389 X0.1

; $4.712389 = 2\pi r \cdot 90^\circ / 360^\circ$

5.16.4. Monitor Touch Probe**G51**

SYNTAX: **G51 <axismask>** ; Monitor probe input

The UNIDEX 600 Series controller provides support for digital touch probe measuring. It is designed to permit you to determine the location of the part in space.

The Probe command initializes the touch probe, the **G51** command activates probe monitoring. When the probe input is detected, the CNC aborts the move in progress and stores the current position of each axis into the variable array defined by the Probe command. This enables you to start the part moving toward the probe and have the part stop moving when it reaches the probe. The program may then use the position information returned to determine the physical location of the part in space.

Once a probe cycle is initiated, the CNC actively monitors the appropriate input channel until the probe input is detected. When a probe touch occurs the cycle is complete. To initiate a probe measuring cycle again, execute another **G51** command.

Refer to the extended commands **Probe** and **DVAR** for more information.

The **G51** command can be used after a previous execution of a **G51** command. This will repeat the cycle. A **G51** without an *axismask* disables probe monitoring. The axis positions of the specified axes are stored in the variable array in the order of the *standard axis names*, not in the order of the axis names specified in the **G51** command.

EXAMPLE:

```
DVAR $POSDATA[16]           ;Define an array to hold positions
PROBE 10 0 $POSDATA[0]      ;Initialize touch probe input on virtual input bit 10.
                             ;The probe being used is active low. Positional
G51 X                      ;information will be placed into the POS array.
G1 X5.0 F30                 ;Monitor probe input
                             ;Start motion towards probe
```

5.16.5. Define Polar/Cylindrical Transformation Axes**G52**

SYNTAX: **G52 <axisMask>** ; Order of the axis is significant !

except: the axis mask must contain exactly two axis letters.

EXAMPLE: **G52 D v**



The **G52** command defines a virtual axis pair to be used for the Polar Coordinate Transformation (**G46**), or the Cylindrical Coordinate Transformation (**G47**). You cannot use both Polar and Cylindrical transformations on the same task at the same time.

The order of the axes specified is very important. The exact meaning of the axis depends on whether a Polar or Cylindrical transformation will be used. See **G46** or **G47** for details.

5.17. Fixture Offsets (G53 – G59)

The fixture offset feature provides the user with the ability to program part dimensions relative to a fixed point in space, not knowing the absolute coordinates of that point. Dimensional distances specified in the absolute (**G90**) mode are relative to this point in space, as opposed to the last software home position.

The UNIDEX 600 Series controller provides support for two such points in space. These points are referred to as fixture offset #1 and fixture offset #2. Separate G-codes have been implemented to permit the user to define which fixture offset is currently active.

5.17.1. Cancel Fixture Offset

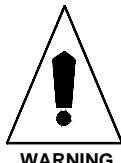
G53

SYNTAX: G53~<*axisMask*>
G53

EXAMPLE: G53 X v



The **G53** command cancels fixture offsets on all axes or if a mask is provided, on selective axes. The preset position register(s) of the axes where the fixture offset is applied are immediately updated to reflect the change in the coordinate system. The machine position registers are unaffected. Please note that only one offset may be active per axis. The offsets are not cumulative. If the command is invoked with an *axismask*, then the action takes place only for the specified axes. If there is no *axismask*, the action takes place for all axes, owned by the current task. This command is the **default**.



WARNING

You may selectively remove fixture offsets from some axes, however, this practice is not recommended, and can lead to extreme confusion. It is strongly recommended that G53 be used without parameters.

5.17.2. Set Fixture Offset #1

G54

SYNTAX: G54~<*axisPoint*>
G54

EXAMPLE: G54 X8.9 Y\$GLOB0 z9

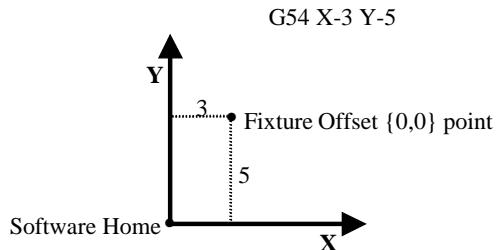


The **G54** command specifies the absolute coordinates (measured from software home) of the point referred to as fixture offset #1. After a **G54** the preset position register(s) of the specified axes are updated immediately to reflect the change in the coordinate system (the machine position registers are unaffected).

After a **G54**, all absolute position coordinates (coordinates provided when you are in **G90** mode) specified by the user are measured from Fixture offset #1. Move Coordinates provided when in **G91** (incremental) mode are unaffected by fixture offset #1.

If the command is invoked with an *axispoint*, then the fixture offset value is stored in the FixtureOffset machine parameter, for each axis in the *axispoint*. After a subsequent **G53** (cancels the fixture offset), then you can execute a **G54** without an *axispoint*. If there is no

axispoint, the action takes place for all axes assigned to the task, using the FixtureOffset machine parameter values. See the example below:



NOTE: Software home is the same as hardware home, if no presets are active (**G92**)

If no fixture offsets are active, then a **G53** will set the preset position registers equal to the machine position registers for the specified axes.

EXAMPLE PROGRAM:

See the comprehensive example (Table 5-7) also, which shows fixture offset #1 and #2 in combination with software home.

Line#	ProgramLine	X,Y Preset	X,Y Machine
N10	HOME X Y	0,0	0,0
N20	G54 X3	-3,0	0,0
N30	G90 G1 X1 Y1	-2,1	1,1
N40	G53	1,1	1,1
N50	G54	-2,1	1,1
N40	G53	1,1	1,1



5.17.3. Set Fixture Offset #2

G55

SYNTAX : G55~<axisPoint >
G55

EXAMPLE: G55 X8.9 Y\$GLOB0 z9

The **G55** command specifies the absolute coordinates (measured from software home) of the point referred to as fixture offset #2. After a **G55** the preset position register(s) of the specified axes are updated immediately to reflect the change in the coordinate system (the machine position registers are unaffected).

After a **G55**, all absolute position coordinates (coordinates provided when you are in **G90** mode) specified by the user are measured from Fixture offset #2. Move Coordinates provided when in **G91** (incremental) mode are unaffected by fixture offset #2.

If the command is invoked with an axispoint, then the fixture offset value is stored in the FixtureOffset2 machine parameter, for each axis in the axispoint. After a subsequent **G53** (remove the fixture offset), then you can execute a **G55** without an axispoint. If there is no axispoint, the action takes place for all axes assigned to the task, using the FixtureOffset machine parameter values. See the example below:

See the comprehensive example (Table 5-7) also, which shows fixture offset #1 and #2 in combination with software home.

EXAMPLE PROGRAM:

```
G55 X10. Y5. Z3. ;Enable fixture offset #2. All dimensional data will now
;be relative to the point (10,5,3) instead of (0,0,0).
```



5.17.4. Set Fixture Offset #3

G56

SYNTAX : G56~<axisPoint >
G56

EXAMPLE: G56 X8.9 Y\$GLOB0 z9

The **G56** command specifies the absolute coordinates (measured from software home) of the point referred to as fixture offset #3. After a **G56** the preset position register(s) of the specified axes are updated immediately to reflect the change in the coordinate system (the machine position registers are unaffected).

After a **G56**, all absolute position coordinates (coordinates provided when you are in **G90** mode) specified by the user are measured from Fixture offset #3. Move Coordinates provided when in **G91** (incremental) mode are unaffected by fixture offset #3.

If the command is invoked with an axispoint, then the fixture offset value is stored in the FixtureOffset3 machine parameter, for each axis in the axispoint. After a subsequent **G53** (remove the fixture offset), then you can execute a **G56** without an axispoint. If there is

no axispoint, the action takes place for all axes assigned to the task, using the FixtureOffset machine parameter values. See the example below:

EXAMPLE PROGRAM:

See the comprehensive example (Table 5-7) also, which shows fixture offset #1 and #2 in combination with software home.

```
G56 X10. Y5. Z3.      ; Enable fixture offset #3. All dimensional data will
                      ; now be relative to the point (10,5,3) instead of
                      ; (0,0,0).
```

5.17.5. Set Fixture Offset #4

G57

SYNTAX : **G57~<axisPoint >**
G57

EXAMPLE: G57 X8.9 Y\$GLOB0 z9



The **G57** command specifies the absolute coordinates (measured from software home) of the point referred to as fixture offset #4. After a **G57** the preset position register(s) of the specified axes are updated immediately to reflect the change in the coordinate system (the machine position registers are unaffected).

After a **G57**, all absolute position coordinates (coordinates provided when you are in **G90** mode) specified by the user are measured from Fixture offset #4. Move Coordinates provided when in **G91** (incremental) mode are unaffected by fixture offset #4.

If the command is invoked with an axispoint, then the fixture offset value is stored in the FixtureOffset4 machine parameter, for each axis in the axispoint. After a subsequent **G53** (remove the fixture offset), then you can execute a **G57** without an axispoint. If there is no axispoint, the action takes lace for all axes assigned to the task, using the FixtureOffset machine parameter values. See the example below:

EXAMPLE PROGRAM:

See the comprehensive example (Table 5-7) also, which shows fixture offset #1 and #2 in combination with software home.

```
G57 X10. Y5. Z3.      ; Enable fixture offset #4. All dimensional data will
                      ; now be relative to the point (10,5,3) instead of
                      ; (0,0,0).
```

5.17.6. Set Fixture Offset #5**G58****SYNTAX :** G58~<axisPoint >
G58**EXAMPLE:** G58 X8.9 Y\$GLOB0 z9

The **G58** command specifies the absolute coordinates (measured from software home) of the point referred to as fixture offset #5. After a **G58** the preset position register(s) of the specified axes are updated immediately to reflect the change in the coordinate system (the machine position registers are unaffected).

After a **G58**, all absolute position coordinates (coordinates provided when you are in **G90** mode) specified by the user are measured from Fixture offset #5. Move Coordinates provided when in **G91** (incremental) mode are unaffected by fixture offset #5.

If the command is invoked with an axispoint, then the fixture offset value is stored in the FixtureOffset5 machine parameter, for each axis in the axispoint. After a subsequent **G53** (remove the fixture offset), then you can execute a **G58** without an axispoint. If there is no axispoint, the action takes place for all axes assigned to the task, using the FixtureOffset machine parameter values. See the example below:

EXAMPLE PROGRAM:

See the comprehensive example (Table 5-7) also, which shows fixture offset #1 and #2 in combination with software home.

```
G57 X10. Y5. Z3.      ; Enable fixture offset #4. All dimensional data will
                      ; now be relative to the point (10,5,3) instead of
                      ; (0,0,0).
```

5.17.7. Set Fixture Offset #6**G59****SYNTAX :** **G59~<axisPoint >**
G59**EXAMPLE:** **G59 X8.9 Y\$GLOB0 z9**

The **G59** command specifies the absolute coordinates (measured from software home) of the point referred to as fixture offset #6. After a **G59** the preset position register(s) of the specified axes are updated immediately to reflect the change in the coordinate system (the machine position registers are unaffected).



After a **G59**, all absolute position coordinates (coordinates provided when you are in **G90** mode) specified by the user are measured from Fixture offset #6. Move Coordinates provided when in **G91** (incremental) mode are unaffected by fixture offset #6.

If the command is invoked with an axispoint, then the fixture offset value is stored in the FixtureOffset6 machine parameter, for each axis in the axispoint. After a subsequent **G53** (remove the fixture offset), then you can execute a **G59** without an axispoint. If there is no axispoint, the action takes place for all axes assigned to the task, using the FixtureOffset machine parameter values. See the example below:

EXAMPLE PROGRAM:

See the comprehensive example (Table 5-7) also, which shows fixture offset #1 and #2 in combination with software home.

```
G59 X10. Y5. Z3.      ; Enable fixture offset #6. All dimensional data will
                      ; now be relative to the point (10,5,3) instead of
                      ; (0,0,0).
```

Table 5-7. Fixture Offset Example

Line #	Program Line	Comments	X, Y Preset	X, Y Machine
N10	HOME X Y	Move X and Y axes to hardware home	0,0	0,0
N20	G90 F100.	Use absolute distance mode and set feedrate.	0.0	0.0
N30	G1 X10. Y10.	Move the X and Y axes 10.	10,10	10,10
<hr/>				
N40	G92	Set software home	0,0	10,10
N50	G54 X5. Y3.	Activate fixture offset #1 at 5,3	-5,-3	10,10
N60	G1 X10. Y15.	Move the X axis 10.0 and Y axis 15.0	10,15	25,28
<hr/>				
N70	G53	De-activate fixture offsets	15,18	25,28
N80	G55 X-10. Y5.	Activate fixture offset #2 at -10,5	25,13	25,28
N90	G1 X10. Y15.	Move to absolute position 10,15	10,15	10,30
N100	G53	De-activate fixture offsets	0,20	10,30
<hr/>				
N110	G54 X5. Y5.	Re-activate fixture offset #1 at 5,5	-5,15	10,30
N120	G1 X5. Y5.	Move to absolute position 5,5	5,5	20,20
N130	G55 X10. Y10.	De-activate fixture offset #1 and activate fixture offset #2	0,0	20,20
<hr/>				
N140	G1 X-10. Y-10.	Move to absolute position -10,-10	-10, -10	10,10
N150	G53	De-activate all fixture offsets	0,0	10,10

When the fixture offsets were activated (N50, N80 and N110), the value of the fixture offset was subtracted from the value of the preset registers for those axes.

When the fixture offsets were deactivated (N70, N100 and N150), the offset currently being used was added into the value of the preset registers for those axes.

When changing from fixture offset #1 being active to fixture offset #2 being active, the values for fixture offset #1 were added into the preset registers before the values for fixture offset #2 were added into those registers.

5.18. Contoured Accel/Decel Overview (G60, G61)

The trajectory generator provides several types of automatic acceleration and deceleration for use on contoured motion (**G1**, **G2**, **G3**, **G12**, and **G13**). Note, that **G0**, spindle motion or asynchronous motion does not use the methods described here, but, instead, their own set of parameters to control the acceleration/deceleration in a similar manner. Table 5-8 summarizes all of the G-codes pertaining to acceleration/deceleration for contoured motion (**G1**, **G2**, **G3**, **G12**, and **G13**).

Table 5-8. Accel/Decel G-codes Summary

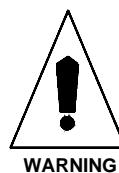
G-code	Function
G60 F<value>	Set acceleration time (via the <i>AccelTimeSec</i> task parameter)
G61 F<value>	Set deceleration time (via the <i>DecelTimeSec</i> task parameter)
G63	Set sinusoidal (1-cosine) accel/decel
G64	Set linear accel/decel
G65 F<value>	Set linear axes acceleration rate (via the <i>AccelRateIPS2</i> task parameter)
G66 F<value>	Set linear axes deceleration rate (via the <i>DecelRateIPS2</i> task parameter)
G67	Set time-based accel/decel (G60/G61 values used)
G68	Set rate-based accel/decel (G65/G66/G165/G166 values used)
G165 F<value>	Set rotary axes acceleration rate
G166 F<value>	Set rotary axes deceleration rate

The user should specify (using the G codes in Table 5-8) both “acceleration data” and “deceleration data”, which are normally set to the same value. The controller will always use the “acceleration data” supplied by the user to accel/decel between feedrates specified in subsequent CNC program blocks (using F or E words). The controller will use the “deceleration data” supplied by the user only to decelerate to zero speed at the end of a series of moves, or to decelerate to a lower speed dictated by the acceleration limiting feature.

The automatic acceleration computed by the trajectory generator can in some cases conflict with the feedrates supplied in the program. See 5.18.1. Explicit Feedrates and Automatic Acceleration for details on these situations.

In some cases, the axes may not reach the programmed vectorial feedrate, due to feedrate limiting. Feedrate limiting will scale down the vectorial velocity, so that no axis exceeds its maximum feedrate.

There are a number of conditions where the controller cannot obey the specified acceleration and may generate an instantaneous deceleration of the velocity command. See **G8** and **G9** for details.



WARNING

5.18.1. Explicit Feedrates and Automatic Acceleration

The controller will always use the acceleration/deceleration parameters supplied by the user to change between feedrates explicitly specified in the CNC program (using F or E words), even if such changes actually imply decelerations (changes to lower speeds).

When a feedrate is specified on, or immediately in front of, a contoured motion line (for example “G1 X3 F10”), the controller attempts to reach that feedrate exactly when the motion on that line is complete. However if a change in specified feedrates between two given moves is small compared to the specified acceleration data, then the controller will still follow the acceleration data and therefore reach the required feedrate before the move is complete (see Figure 5-29 diagram A). The remainder of the move is then completed at that constant feedrate. The controller will also follow the acceleration data if the change in feedrates is large compared to the acceleration data (see Figure 5-29 diagram B). In this case the specified feedrate is not reached by the end of the move. The controller will still continue to attempt to reach the specified feedrate during following moves, using the same acceleration data. Of course if a following move specifies a new feedrate, the old acceleration is abandoned and a new one is begun.

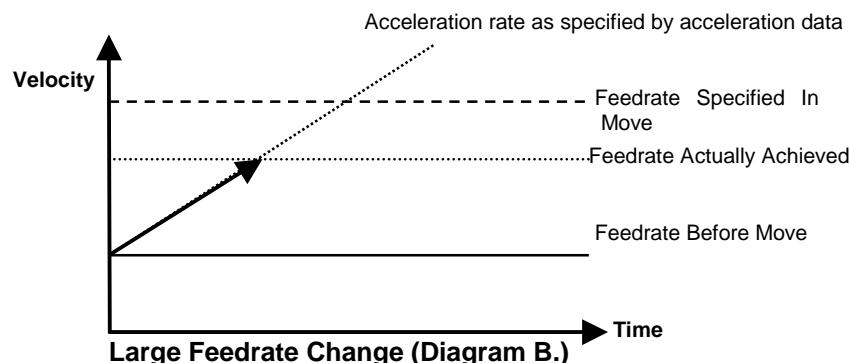
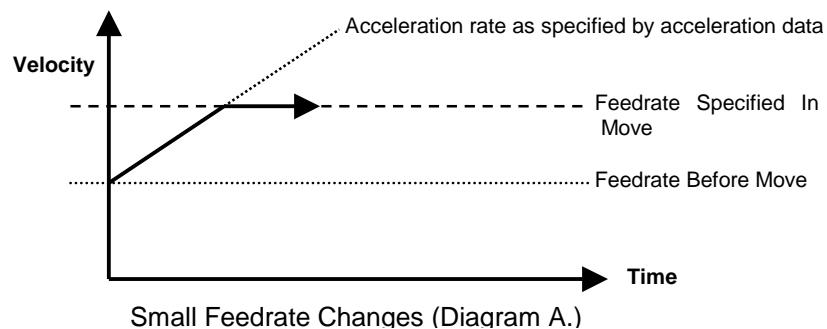


Figure 5-29. Feedrate Changes

5.18.2. Set Acceleration Time

G60

SYNTAX: **G60~F<fExpression>** ;Where the expression is the accel time in seconds

G60~P<fExpression> ;Where the expression is the accel time in seconds

EXAMPLE: **G60 F0.5** ;Sets new acceleration time to 1/2 sec (500 msec)



When performing acceleration using the time based (**G67**) parameters, the UNIDEX 600 Series controller uses the *AccelTimeSec* task parameter to determine the duration of the acceleration. **G60** sets this task parameter for the current task. The accel time may follow either an F or P keyword, as shown in the syntax definition above.

Subsequent motion commands accelerate to the commanded velocity within the specified time period. This time period is specified in seconds, with a resolution of 0.001 seconds (1 millisecond).

The same acceleration time is used for both rotary and linear axes.

The parameter may be set regardless of the current setting of the Ramp Type G-code group. However, the effect of this parameter will be apparent only when operating in a time based (**G67**) acceleration mode.



5.18.3. Set Deceleration Time

G61

SYNTAX: **G61~F<fExpression>** ;Where the expression is the decel time in seconds

G61~P<fExpression> ;Where the expression is the decel time in seconds

EXAMPLE: **G61 F.25** ;Sets new deceleration time to 1/4 sec (250 msec)



When performing deceleration using the time based (**G67**) parameters, the *DecelTimeSec* task parameter to determine the duration of the deceleration. **G61** sets this task parameter for the current task. The decel time may follow either an F or P keyword, as shown in the syntax definition above.

Subsequent motion commands accelerate to the commanded velocity within the specified time period. This time period is specified in seconds, with a resolution of 0.001 seconds (1 millisecond).

The same deceleration time is used for both rotary and linear axes.

The parameter may be set regardless of the current setting of the Ramp Type G-code group. However, the effect of this parameter will be apparent only when operating in a time based (**G67**) acceleration mode.





5.19. Profile Resolution Time (G62)

5.19.1. Set Profile Time

G62

SYNTAX: G62~F<fExpression>

;Where the expression is the time in seconds

G62~P<fExpression>

;Where the expression is the time in seconds

EXAMPLE: G62 F.005

;Sets the profile time to 5 milliseconds

The **G62** command sets the *UpdateTimeSec* task parameter for the current task, which defines the time between calculated points for the CNC profiler. The profile time may follow either an F or P keyword, as shown in the syntax definition above.

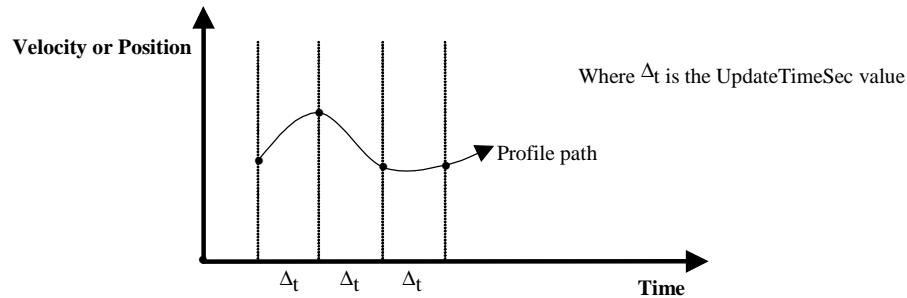


Figure 5-30. UpdateTimeSec Diagram

5.20. Accel/Decel Rates and Modes (G63 -> G68)

5.20.1. Sinusoidal (1-Cosine) Accel/Decel Mode

G63

SYNTAX: G63

EXAMPLE: G63



The **G63** command specifies that the acceleration and deceleration type to be used is sinusoidal. **G64** specifies that the acceleration and deceleration will be linear. The acceleration and deceleration profiles will always be of the same type (linear/sinusoidal). The sinusoidal/linear programming mode state is indicated by bit 2 of the *Mode1* task parameter. This command is the default.

The UNIDEX 600 Series controller offers the flexibility of choosing between two distinct types of acceleration/deceleration; linear or sinusoidal. Sinusoidal acceleration, also called (1-cosine), since this more exactly describes the acceleration curve:

$$\text{acceleration curve: } v_p = 100 * (1 - \cosine(\theta)) / 2$$

$$\text{deceleration curve: } v_p = 100 * (1 + \cosine(\theta)) / 2$$

where:

v_p is % of full speed, and θ varies from 0° at the beginning of accel/decel to 180° at the end of the accel/decel phase (T_1 and T_2 in Figure 5-31, respectively).

As illustrated in Figure 5-31, both Linear and Sinusoidal acceleration will take place over the same time interval (time interval determined by the **G67 / G68** modes). The **G63** command specifies that the acceleration and deceleration type to be used is sinusoidal.

Sinusoidal acceleration is typically used on systems containing a large inertial mass resistant to sudden changes in acceleration. As illustrated, the motion accelerates gradually, then accelerates steeply. As it approaches the commanded velocity, the acceleration gradually decreases until it reaches zero.

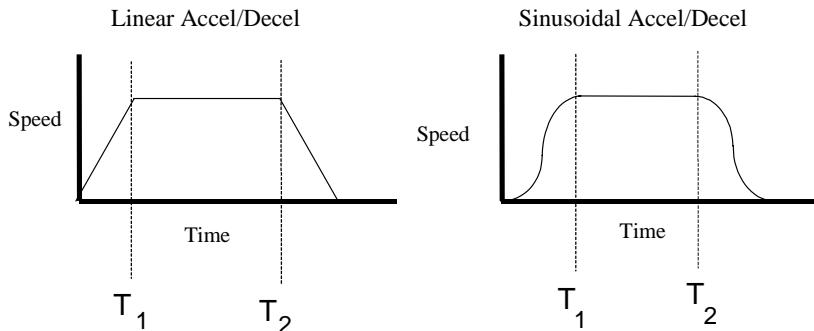


Figure 5-31. Constant vs. Cosine Acceleration

The acceleration and deceleration occurs over the same time period regardless of whether sinusoidal or linear mode is chosen.



Disadvantage of Sinusoidal Acceleration

The disadvantage of sinusoidal acceleration is that the acceleration rate will vary throughout the profile, it will show a higher acceleration rate (1.57 times higher) in the middle of the curve, than linear acceleration would. Another disadvantage of sinusoidal acceleration is that it is wasteful and sometimes less accurate when the move duration is less than the profile time.

Advantage of Sinusoidal Acceleration

Sinusoidal acceleration reduces jerk (sudden changes in acceleration) that occurs at the beginning and end of linear acceleration profiles.

5.20.2. Linear Accel/Decel Mode**G64****SYNTAX:** G64**EXAMPLE:** G64

The UNIDEX 600 Series controller offers the flexibility of choosing between two distinct types of acceleration/deceleration: linear or sinusoidal. The **G64** command specifies the acceleration/deceleration type to be used is linear. See **G63** for more details on linear and sinusoidal acceleration/deceleration. The sinusoidal/linear programming mode state is indicated by bit 2 of the *Mode1* task parameter.

Refer to Figure 5-31.

5.20.3. Set Acceleration Rate (for linear type axes)**G65****SYNTAX:** G65~F<*fExpression*> ; Where the acceleration rate is in inchesG65~F<*fExpression*> ; Where the acceleration rate is in inches**EXAMPLE:** G65 F100

When accelerating a linear type axis, using the rate based parameters (**G68**), the UNIDEX 600 Series controller uses the AccelRateIPS2 task parameter as the acceleration value. Also, this parameter may be set (for the current task) with a **G65** command. This rate is specified in inches/second/second, unless, bit 4 is set to one in the CompatibilityMode global parameter, in which case it is in user units/second/second. This setting is ignored if the acceleration is time based (**G67**). The acceleration rate may follow either an F or P keyword, as shown in the syntax definition above.

Note that the rate is applied over the vectorial distance of the move. For example, if a **G70 G90 X1 Y1** executes, then the specified rate is correct over the vectorial distance of 1.414 inches, not 1 inch.

G66 only applies when all the axes in the move are of linear type or if the move contains both rotary and linear axes and the dominance of the move is linear (refer to the **G98/G99** Overview for details on axes dominance). See **G165** if the axes are rotary, or if rotary axes are dominant.

EXAMPLE PROGRAM:

G70 G65 F0.5	;Sets the new acceleration rate to 0.5 inches/second/second
G71 G65 F1.27	;Sets the new acceleration rate to 1.27 mm/second/second

5.20.4. Set Deceleration Rate (for linear type axes)

G66

SYNTAX: **G66~F<fExpression>** ; Where the acceleration rate is in inches
G66~P<fExpression> ; Where the acceleration rate is in inches

EXAMPLE: **G66 F100**

When decelerating a linear type axis, using the rate based parameters (**G68**), the *UNIDEX 600 Series controller* uses the *DecelRateIPS2* task parameter as the deceleration value. Also, this parameter can be set (for the current task) with a **G66** command. This rate is specified in either inches/second/second, unless bit 4 is set to 1 in the *CompatibilityMode* global parameter, in which case it is user units/second/second. This rate is ignored if the deceleration is time based (**G67**). The decel rate may follow either an F or P keyword, as shown in the syntax definition above.

Note that the rate is applied over the vectorial distance of the move. For example, if a G70 G90 X1 Y1 executes, then the specified rate will be correct over the distance of 1.414 inches, not 1 inch.

G66 only applies when all the axes in the move are of linear type or if the move contains both rotary and linear axes and the dominance of the move is linear (refer to the **G98/G99** Overview for details on axes dominance). See **G166** if the axes are rotary, or if rotary axes are dominant.

EXAMPLE PROGRAM:

G70 G66 F0.1	;Sets the new deceleration rate to 0.1 inches/second/second
G71 G66 F0.254	;Sets the new deceleration rate to 1.27 mm/second/second

5.20.5. Time Based Acceleration/Deceleration

G67

SYNTAX: **G67**

EXAMPLE: **G67**

The *UNIDEX 600 Series controller* offers two modes of operation with respect to acceleration and deceleration: time or rate based. The **G67** command specifies that acceleration and deceleration are to occur with time-based parameters.

While operating in this mode, the *AccelTimeSec* and *DecelTimeSec* task parameters specify the amount of time all moves are to accelerate to and decelerate from the commanded velocity. These parameters may be changed using the **G60** and **G61** commands, respectively. This command is the **default**. The time/rate based programming mode state is indicated by bit 3 of the *Mode1* task parameter.





5.20.6. Rate Based Acceleration/Deceleration

G68

SYNTAX: G68

EXAMPLE: G68

The UNIDEX 600 Series controller offers two modes of operation with respect to acceleration and deceleration: time and rate based. The **G68** command specifies the occurrence of acceleration and deceleration based on the rate based parameters.

There are different rates for linear and rotary axes. Therefore, there are four relevant rates: linear type acceleration, linear type deceleration, rotary type acceleration and rotary type deceleration. The time/rate based programming mode state is indicated by bit 3 of the *Mode1* task parameter.

The *AccelRateIPS2* and *DecelRateIPS2* task parameters specify the amount the velocity is to change each second for linear type axes. Therefore, the amount of time used to reach the commanded velocity varies between moves. These parameters may be changed using the **G65** and **G66** commands, respectively.

The *AccelRateDPS2* and *DecelRateDPS2* task parameters specify the amount the velocity is to change each second for rotary type axes. Therefore, the amount of time used to reach the commanded velocity varies between moves. These parameters may be changed using the **G165** and **G166** commands, respectively.

Rate-based acceleration and deceleration are typically used on systems limited in acceleration and commanded to varying velocities.



When using sinusoidal acceleration (**G63**), in rate based mode, the acceleration rate will vary throughout the curve and show a higher acceleration rate (1.57 times higher) in the middle of the curve, than specified.

5.21. Metric/English Units (G70, G71)

The user can change the units at will, as interpreted in the floating point constants and variables that represent positions, distances, speeds, or accelerations. If a **G70** or **G71** is on the same line as motion, then the **G70/G71** executes before the motion.

5.21.1. Inch Dimension Programming Mode (Units) **G70**

SYNTAX: **G70**

EXAMPLE: **G70**



The UNIDEX 600 Series controller provides the user with the option of specifying all distances and feedrates in either English units (inches) or metric units (millimeters). The **G70** command indicates English units.

While the **G70** mode is active and an axis is a linear type, all distances are in inches, all speeds are in inches per minute, and all acceleration rates are in inches per second². If the axis is a rotary type, the distances are degrees and velocities are in RPM, regardless of the **G70/G71** settings. This command is the **default**. This G code also affects the positions and velocities as displayed by the MMI600-NT/95. Each of the 4 tasks maintains their own **G70/G71** mode. The English/Metric programming mode state is indicated by bit 0 of the *Mode1* task parameter.

EXAMPLE PROGRAM:

```
G70      ;Set English programming mode (inches)
G1 X10.   ;Move the X axis 10 inches in the positive direction
G1 Y-5.   ;Move the Y axis 5 inches in the negative direction
F100.    ;A feedrate of 100 inches per minute is established
```



5.21.2. Metric Dimension Programming Mode (Units)

G71

SYNTAX: G71

EXAMPLE: G71

The UNIDEX 600 Series controller provides the user with the option of specifying all distances and feedrates in either English units (inches) or metric units (millimeters). The **G71** command indicates that units are metric.

While the **G71** mode is active and an axis is a linear type, all distances are in millimeters, all speeds are in millimeters per minute, and all acceleration rates are in millimeters per second². If the axis is a rotary type, the distances are degrees and velocities are in RPM, regardless of the **G70/G71** settings. The English/Metric programming mode state is indicated by bit 0 of the *Mode1* task parameter.

EXAMPLE PROGRAM:

```

G71      ;Set metric programming mode (millimeters)
G1 X4.   ;Move the X axis 4 mm in the positive direction
G1 Y-2.  ;Move the Y axis 2 mm in the negative direction
F100.    ;A feedrate of 100 mm per second is established

```

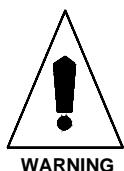
5.22. Restore Preset Position Registers

G82

SYNTAX: G82~<axisMask>
G82

EXAMPLE: G82 X v

The **G82** command restores the position registers to their value prior to executing a **G92** command. See the **G92** command for more details on position registers. If the command is invoked with an axismask, then the action takes place only for the specified axes. If there is no axismask, the action takes place for all axes.



The **G82** command is not valid in the **G42** or **G43** mode.

If no fixture offsets are active, then a **G82** will make the preset position registers equal the machine position registers, for the specified axes.

5.23. Transformation Overview (G83, G84)

These codes allow the programmer to apply transformations in a plane to the coordinates they supply in constants or variables without changing the actual values they supply in the **G1**, **G2** or **G3** commands. Note, mirroring and part rotation may not be used with **G0** commands. The motion to be transformed must be composed entirely of **G1**, **G2**, **G3**, **G12** and **G13** commands. Also, these transformations may be active simultaneously. When applied simultaneously, the order the transformations were executed is not significant.

If combinations of the following transformations are simultaneously active, they will be applied to the command in the following order:

Software Home	-G92
Parts Rotation	-G84
Mirroring	-G83
Polar	-G46
Cylindrical	-G47

5.23.1. Mirror Image

G83

SYNTAX: **G83~<axisMask>**
G83

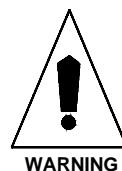
EXAMPLE: **G83 X v z**

The **G83** command activates or deactivates the mirror image function. The mirror function is enabled by a **G83** with an axis mask on the line, any axis not specified in this mask will have mirroring disabled when the command is executed, mirroring is not cumulative. The mirror function is disabled by the **G83** command with no axes specified. You can verify the current state of the Mirroring mode, by viewing the ‘MirrorActive’ bit of the Status3 task variable (use the AerStat.exe utility), or by looking for **G83** in the G code display in the lower left of the MMI 600 run or manual screens.



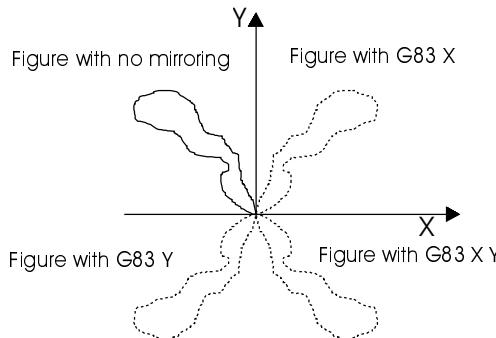
The **G83** command operates in both the relative and absolute mode (**G90/G91**). In relative mode, the origin or point in the plane at the moment mirroring takes place is the current position in the plane. In absolute mode, the origin is the home position.

While in the absolute mode, the user must be at the software home established by the **G92** command when activating the mirror mode, or unexpected results occur.



WARNING

Refer to Figure 5-32 and Figure 5-33 for mirror image examples.

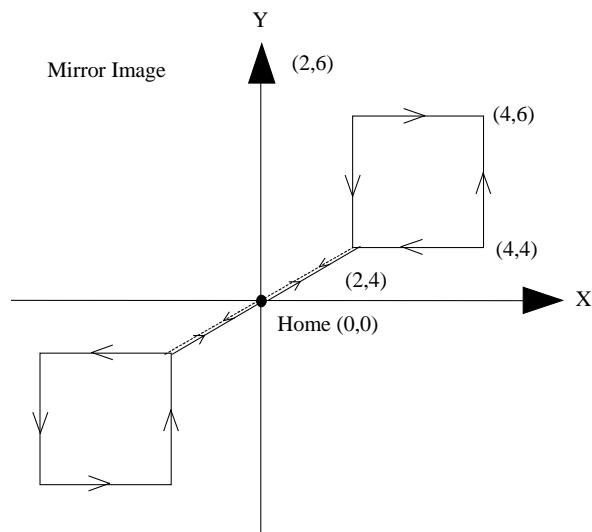
**Figure 5-32. G83 Mirror Image Example 1****EXAMPLE PROGRAM:**

```

G91      ; Set relative programming mode
G70      ; English programming units
HOME X Y ; Move to reference (0, 0)
G83      ; Disable mirroring
CLS BOX1 ; Call subroutine BOX1 (draws box in 1st quadrant in Figure 5-33)
G83 X Y ; Activate mirror image function for X and Y axes, changing positive
          ; X and Y values to negative values
CLS BOX1 ; Call subroutine BOX1 (draws box in 3rd quadrant in Figure 5-33)
G83      ; Disable mirroring, REMEMBER TO DO THIS !
M2       ; Stop the program

DFS BOX1 ; Define subroutine BOX1
F100.
G1 X2. Y4. ; Initiate a positive linear move for the X and Y axis
X2.        ; Initiate a positive linear move for the X axis
Y2.        ; Initiate a positive linear move for the Y axis
X-2.       ; Initiate a negative linear move for the X axis
Y-2.       ; Initiate a negative linear move for the Y axis
X-2. Y-4. ; Initiate linear move of X and Y axis to start position
ENDDFS   ; end of subroutine

```

**Figure 5-33. G83 Mirror Image Example 2**

5.23.2. Parts Rotation

G84

SYNTAX: **G84~<axisMask>~F<fExpression>**

except: the axis mask must contain exactly two axis letters.

EXAMPLE: **G84 X Y F\$VAR1 ; Part rotation angle from X-Y plane is set to
; VAR1 value**

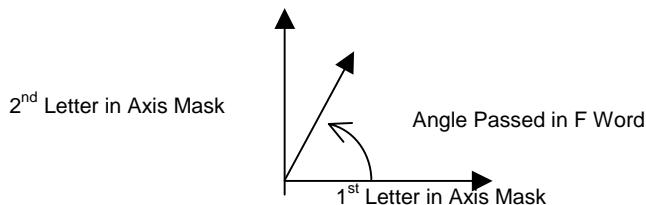
G84 X Y ; Part rotation is turned off



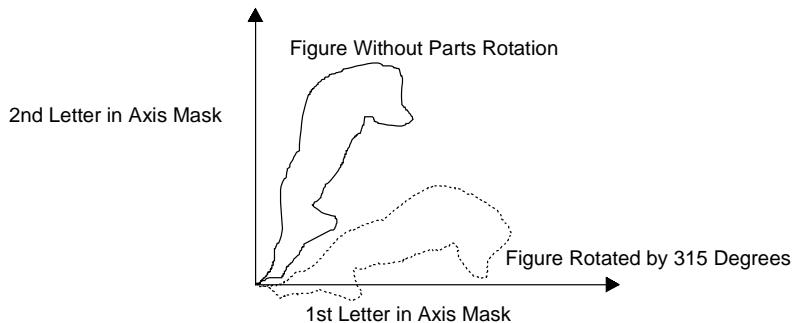
The **G84** command defines the plane and angle of parts rotation. The parts rotation programming feature permits the changing of orientation (in a plane) of a sequence of moves without changing the move coordinates or changing permanent coordinate reference frames.

The angle must be specified in positive degrees. The angle is relative and is based on the last angle specified in a **G84** command (see the picture below). A positive angle produces a CCW angle as referenced from the axes' plane. If no angle is specified, parts rotation will be disabled for the specified axes.

This statement block must occupy its own line (no other blocks on the line). **G84** is modal so rotation is in effect until deactivated. You can verify the current state of the Parts Rotation mode, by viewing the 'RotationActive' bit of the Status3 task variable (use the AerStat.exe utility), or by looking for 'G84' in the active G code display in the lower left of the UNIDEX 600 MMI run or manual screens. The active state of this command is indicated by bit 0 of the Status3 task parameter. Parts rotation is not disabled by the HOME command.



Parts rotation may be activated in either **G90** or **G91** mode, but you may not change the mode while parts rotation is active, or unexpected results may occur. **G0** commands may not be used while parts rotation is active or unexpected results may occur.



You may activate parts rotation at non-zero coordinates, causing coordinate rotation around an offset point. However, when rotating about an offset point, be sure that when you disable parts rotation, you are physically at the same point in space at which parts rotation was activated if you will be doing any movement after parts rotation is deactivated, or unexpected results may occur.

EXAMPLES (see Figure 5-34):

```
$GLOBAL0 = 0;
G71
N1      G91          ; Incremental programming mode
        G1 X5. F500.    ; Move X axis five units in plus direction
        G84 X Y F45    ; Set X, Y part rotation angle to 45
N3      X5.          ; Move X axis five units in plus direction
        G84 X Y F270    ; Set the X, Y part rotation angle to -45
N5      X5.          ; Move X axis five units in plus direction
        G84 X Y F630    ; Set the X, Y, part rotation angle to 225
N7      X2.5         ; Move X axis 2.5 units in plus direction
        G84 X Y F$GLOBAL0 ; No change in parts rotation angle
N9      X2.5         ; Move X axis 2.5 units in plus direction
        G84 X Y          ; Disable part rotation
N11     X5.          ; Move X axis five units in plus direction
        G84 X Y F330    ; Reset the part rotation angle to -30
N13     X5.          ; Move X axis five units in plus direction
        G84 X Y F0      ; Disable part rotation
N15     X5.          ; Move X axis five units in plus direction
```

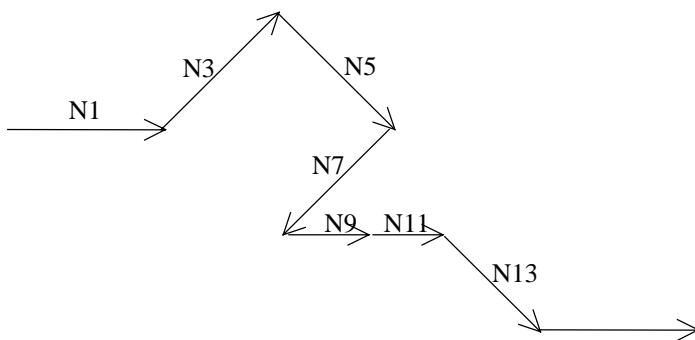


Figure 5-34. G84 Parts Rotation Example

5.24. Positioning Modes (G90, G91)

The user can change the interpretation of the position/distance constants or variables they provide. When a **G90** or **G91** is on the same line as motion, then the **G90/G91** executes before the motion.

5.24.1. Absolute Dimension Programming Mode (Distance) **G90**

SYNTAX: **G90**

EXAMPLE: **G90**

Prior to the execution of motion commands, the UNIDEX 600 Series controller must be told whether programmed dimensional data is to be interpreted as absolute coordinates, or as an offset from the current axis position. The **G90** command specifies that all move values be interpreted as absolute coordinates. This command is the **default**. The absolute/incremental programming mode state is indicated by bit 1 of the *Model* task parameter.



If a rotary Type axis is moved in **G90** mode, then its target position is first modulo'd to 360 degrees. For example, if A is a rotary axis and **G90 A500** is executed, A moves to 140 degrees.

EXAMPLE PROGRAM:

Assume the axes start at (0,0). Figure 5-35 illustrates the results of this example.

G90	;Set absolute programming mode
F100.	;Establish feedrate for subsequent moves
G1 X10.0 Y10.0	;Move X and Y axes to absolute coordinate 10,10
G1 X15.0 Y25.0	;Move X and Y axes to absolute coordinate 15,25
G1 X15.0 Y10.0	;Move X and Y axes to absolute coordinate 15,10

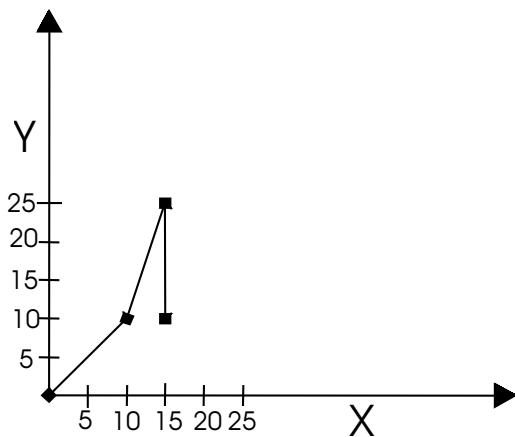


Figure 5-35. Absolute Mode Programming



5.24.2. Incremental Position Programming (Distance)

G91

SYNTAX: G91

EXAMPLE: G91

Prior to the execution of motion commands, the UNIDEX 600 Series controller must be told whether programmed dimensional data is to be interpreted as absolute coordinates or as an offset from the current axis position. The **G91** command specifies that all positions be interpreted as incremental distances from the current position. The absolute/incremental programming mode state is indicated by bit 1 of the *Mode1* task parameter.

If a rotary Type modulo axis is commanded in G91 mode, then its target position is NOT modulo'd to 360 degrees. For example, if A is a rotary axis and G91 A900 is executed, the A axis moves a full two and one half revolutions.

EXAMPLE PROGRAM:

Assume that the starting position is (0,0). Figure 5-36 illustrates the result of this example.

G91	'Set incremental programming mode
F100.	'Establish feedrate for subsequent moves
G1 X10.0 Y10.0	'Move the X and Y axes a distance of 10.0
G1 X15.0 Y25.0	'Move X axis 15.0 and Y axis 25.0 from current position
G1 X15.0 Y10.0	'Move the X axis 15.0 and Y axis 10.0 from current position

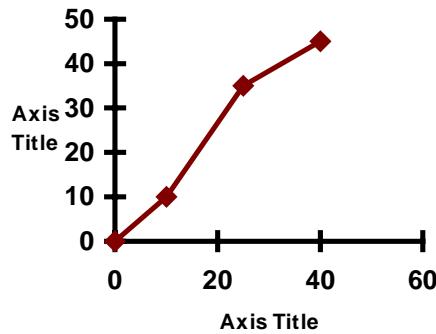


Figure 5-36. Incremental Mode Programming

5.25. Preset Positions (G92)

5.25.1. Software Home (Set Preset Positions) **G92**

SYNTAX: **G92~<axisPoint>**

EXAMPLE: **G92 X8.9 Y\$GLOB0 z9**

This command defines a ‘preset position’ or a software home position. This is useful when the user wants to program in absolute coordinates measured from a non-zero position. The **G92** command has no effect on motion commands specified in relative coordinates (**G91** mode). The **G92** command does not cause any axis movement. A **G92** command within a blended motion CNC program block will force a deceleration to zero velocity.

G92 must be followed by an axis point list. The **G92** command will be executed for all axes in the list, using the associated values. If an axis is not in the list, then its preset will not be changed. For example, a “**G92 X-1 Y-1**” then a “**G92 X-2**” is equivalent to the single command: “**G92 X-2 Y-1**. Axes that are not specified in the axis point, will not be affected.

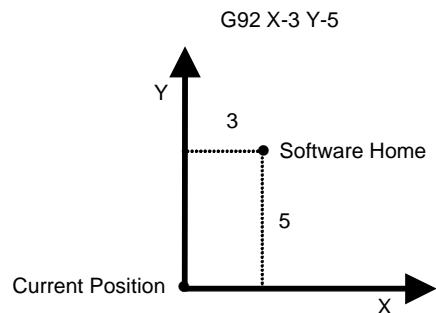
Upon execution of the **G92**, the PresetCmdUnits machine parameters will be assigned to the coordinates provided in the **G92**. No movement is executed. In other words, the coordinates provided in the **G92** specify the current position relative to the new “software home” position. All subsequent motion commands (in absolute coordinates) are then measured from the software home, not {0,0}. For example, if you execute a **G92X0Y0** while the current position is {2,3}, a subsequent **G90G0X1Y1** would go to {3,4}.

You can also provide non-zero coordinates to the **G92**, to declare a software home at a position other than the current position. For example, if the current position (with no presets active) is {2,3}, and you execute a “**G92 X-1, Y-1**”, then the new software home is at {3,4}. A subsequent execution of the command “**G1 X0 Y0**” would move to software home, or the coordinate {3, 4}.

Note that, at this point in the above example, the PresetCmdUnits machine parameter values will read: {-1,-1}, while the PositionCmdUnits machine parameters will still be {2, 3}. The **G92** command changes the value of the PresetCmdUnits machine parameter, but not the values of the PositionUnits and PositionCmdUnits machine parameters. The latter two always reflect the distance from the hardware home position (the zero position when no presets are active), while the former (PresetCmdUnits) always reflects the distance from the preset position set with **G92**. The presets also do not change the value of the POS or POSCMD axis parameters, which specifies the machine counts from the hardware home position.

The **G82** command clears the preset positions of a **G92** command, thereby restoring software home to the hardware home position.

The **G92** has some special considerations, when used in conjunction with a *ScaleFactor*.

**EXAMPLE PROGRAM:**

```
;Where X,Y positions are expressed as {x,y}
G1 X10 Y10      ;Move to PositionCmdUnits={10,10} PresetCmdUnits={10,10}
G92 X-1.0 Y-2.0 ;No move PositionCmdUnits={10,10} PresetCmdUnits={-1,-2}
G1 X0 Y0        ;Move to PositionCmdUnits={11,12} PresetCmdUnits={0,0}
G82             ;No move PositionCmdUnits={11,12} PresetCmdUnits={11,12}
```

5.26. Feedrate Modes (G93, G94, G95)

5.26.1. Inverse Time Feedrate Programming (FeedrateMode) G93

SYNTAX: G93

EXAMPLE: G93

The UNIDEX 600 Series controller provides the flexibility to specify feedrates (the F and E words) in either user units/minute (**G94**) or 1/minutes (**G93**), or user-units/spindle-revolution (**G95**). In all three of the above cases, user units are revolutions in the case of the E word, and inches or millimeters (for **G70** and **G71** modes respectively) for the F word. Note that the actual feedrates reported (the LinearFeedrateActual or RotaryFeedRateActual task parameters) will always be in the same units as the F and E words respectively.



The **G93** command specifies that feedrates should be interpreted as 1/minutes. The comments below concern the F word, but apply equally as well to the E word.

In **G93** mode the actual feedrate used is calculated from:

$$(F \text{ value given in the F word}) = (\text{distance of move}) / (\text{Actual feedrate used in move})$$

Where the distance of the move is measured along the actual path of the move (it is an arc distance for a **G2/G3**), and is in user units while the feedrate is in user-units/minute.

If the actual feedrate in the move is the actual speed during the entire move, (which is not true during automatic acceleration, feedrate limiting , (see Actual Feedrates on page 5-18), then during **G93** mode, the above formulae simplifies to:

$$(F \text{ value given in the F word}) = 1 / (\text{Actual duration of move in minutes})$$

The active **G93/G94/Gx95** mode is indicated by bits 16-17 of the *Mode1* task parameter.
to .7 minutes per revolution

5.26.2. Feed Per Minute Feedrate Programming (FeedrateMode) G94

SYNTAX: G94

EXAMPLE: G94



The UNIDEX 600 Series controller provides the flexibility to specify feedrates (the F and E words) in either user units/minute (**G94**) or 1/minutes (**G93**), or user units/spindle revolution (**G95**). In all three of the above cases, user units are revolutions in the case of the E word, and inches or millimeters (for **G70** and **G71** modes respectively) for the F word. Note that the actual feedrates reported (the *LinearFeedrateActual* or *RotaryFeedRateActual* task parameters) will always be in the same units as the F and E word respectively. The G94 command specifies that feedrates be interpreted as user units per minute. The active G93/G94/Gx95 mode is indicated by bits 16-17 of the *Mode1* task parameter. The formula below assumes that the feedrate is the actual speed during the entire move, which is not true during automatic acceleration, feedrate limiting, etc. (see Actual Feedrates on page 5-18). The feedrate is calculation as:

$$F = \frac{\text{SquareRoot}(X^2 + Y^2 + \dots + a^2)}{\# \text{ of minutes to complete move}}$$

where: X is the move distance for the X axis
 Y is the move distance for the Y axis
 a is the move distance for the a axis
 etc.

When performing circular interpolation on two axes, the sum of the squares for those axes is replaced by the product of the arc radius and the arc angle squared. For example, if circular interpolation is being performed on the X and Y axes, the feedrate would be set as follows:

$$F = \frac{\text{SquareRoot}((R * \Theta)^2 + Z^2 + \dots + a^2)}{\# \text{ of minutes to complete move}}$$

where: R is the arc radius
 Theta is the arc angle (in radians)
 Z is the move distance for the Z axis
 a is the move distance for the a axis.

5.26.3. Feed Per Spindle Revolution Feedrate Programming G95

SYNTAX :

- G95** ; F feedrate now in feed per spindle 1 rev
- G295** ; F feedrate now in feed per spindle 2 rev
- G395** ; F feedrate now in feed per spindle 3 rev
- G495** ; F feedrate now in feed per spindle 4 rev



EXAMPLE:

- G95** ; Feedrate now based on spindle 1 speed
- G395** ; Feedrate now based on spindle 3 speed

The UNIDEX 600 Series controller provides the flexibility to specify feedrates (the F and E words) in either user units/minute (**G94**) or 1/minutes (**G93**), or user units/spindle-revolution (**G95**). In all three of the above cases, user units are revolutions in the case of the E word, and inches or millimeters (for **G70** and **G71** modes respectively) for the F word. Note, that the actual feedrates reported (the LinearFeedrateActual and RotaryFeedRateActual task parameters) will always be in the same units as the F and E word respectively. The active **G93/G94/Gx95** mode is indicated by bits 16-17 of the Mode1 task parameter.

The **G95** command specifies that the feedrates be interpreted in units of user-units/spindle-revolution. If the spindle speed varies during the machining process, the speed of the associated axis also vary.

The **G95** command specifies that the value of the S word (speed of the spindle) is to be used to determine the desired velocity of the other axes in motion. The feedrate keyword (F) then contains the vectorial distance which the axes are to be moved for each revolution of the spindle axis.

The formula below assumes that the feedrate is the actual speed during the entire move, which is not true during automatic acceleration, feedrate limiting, etc. (see Actual Feedrates on page 5-18). This feedrate can be calculated as follows:

$$F = \frac{\text{SquareRoot}(X^2 + Y^2 + \dots + a^2)}{\# \text{ of spindle revs to complete move}}$$

where: X is the move distance for the X (linear) axis
 Y is the move distance for the Y (linear) axis.

When generating circular motion on two axes, the sum of the squares for those axes is replaced by the product of the arc radius and the arc angle squared. For example, if circular interpolation is being performed on the X and Y axes, the feedrate would be set as follows:

$$F = \frac{\text{SquareRoot}((R * \Theta)^2 + Z^2 + \dots + a^2)}{\# \text{ of spindle revs to complete move}}$$

where:

R	is the arc radius
Theta	is the arc angle (in radians)
Z	is the move distance for the Z axis
a	is the move distance for the a axis.

When in the **G95** mode, the unsigned spindle speed is used as the reference, that is, the direction of the spindle motion has no effect on the speed of the contoured motion. The spindle speed that is used as the basis for the slaved axes, is the commanded not the actual. That is, real-time variations in the spindle actual velocity will not cause any variations in the speed of the contoured move.

Actual feedrate in user units is: $\left[\text{ipm} \sqrt{\text{mmpm}} \right] F \left[\frac{\text{user units}}{\text{rev}} \right] * S \left[\frac{\text{rev}}{\text{min}} \right]$

EXAMPLE PROGRAM:

```
G95      ; Set spindle based feedrate mode
G70 F100 ; Define a linear feedrate of 100 in. per spindle revolution
G71 F500 ; Define a linear feedrate of 100 mm.. per spindle revolution
E5.      ; Define a rotary feedrate of 5 revs. per spindle revolution
```

5.26.4. Surface Speed Spindle Feedrate Programming

G96

SYNTAX : **G96 <AxisMask>** ; Use surface speed prog. on Spindle 1
G296 <AxisMask> ; Use surface speed prog. on Spindle 2
G396 <AxisMask> ; Use surface speed prog. on Spindle 3
G496 <AxisMask> ; Use surface speed prog. on Spindle 4



EXAMPLES: **G96 Y** ; Y axis is radial axis to measure surface speed
G396 Y ; Y axis is radial axis to measure surface speed

The UNIDEX 600 Series controller provides the flexibility to specify spindle feedrates (the S word) in either revolutions per minute (**G96**) or surface units (**G97**). The **G96** mode allows the spindle feedrate for the specified spindle to be specified in *user units* of surface (in./mm.) per minute. The active **Gx96/Gx97** mode is indicated by bits 18-21 of the *Mode1* task parameter.

In **G96** mode, every 10 milliseconds, the position of the “radial axis” (the axis provided in the **G96** command) is read, and the spindle is directed to a speed given by the following formulae:

$$\text{RPM} = v / (2\pi R)$$

Where, RPM is the resultant spindle rotational speed, v is the surface speed (the S word value) converted to user units per minute, and R is the current value on the radial axis (in user units). Note, that since this formulae is evaluated every 10 milliseconds, the actual RPM will trail 10 milliseconds behind the current radial axis position.

The “R” value shown in the formulae is in “preset units”, which means it is after all transformations, including presets and fixture offsets have been taken in account. For example, if the center of a lathe is at X5, you can execute a **G92 X0** when at X5, and now the center of the lathe, as far as the spindle is concerned (the point where the spindle speed goes infinite) is at X5.

When in **G96** mode, the normal accel/decel parameters of the spindle are suspended (allowing for instantaneous acceleration) and the velocity and acceleration are completely under the control of the motion of “radial axis”. Therefore, you may easily accelerate the spindle too fast, by moving the radial axis too fast, or by moving it to close to the center of the spindle. If the spindle is under closed loop control, this can result in position or current errors on the spindle.

However, in either **G97** or **G96** mode, the spindle speed will be limited by the *MaxFeedRateIPM* machine parameter value. Therefore, in **G96** mode, as the specified radial axis approaches zero, the *MaxFeedrateIPM* parameter may be used to prevent the spindle from being commanded to an infinite speed.

Note that in either **G97** or **G96** mode, the S word always specifies the spindle speed. If you change modes with the spindle running, the S word value will be reinterpreted by the new mode, and the spindle will accelerate/decelerate to the new speed immediately. Furthermore, if you change from **G97** to **G96** mode with the spindle running, this can cause a sudden “jerk” on the spindle (if it is not currently running at the speed appropriate for the given radial axis value) because, as mentioned above, in **G96** mode the spindle accel/decel parameters are suspended. Also note that you must have the *ExecuteNumSpindles* task parameter set properly to utilize more than one spindle.

The active **Gx96/Gx97** mode is indicated by bits 18-21 of the *Mode1* task parameter.



5.26.5. RPM Spindle Feedrate Programming

G97

SYNTAX :	G97	; Spindle 1 feedrate in RPM
	G297	; Spindle 2 feedrate in RPM
	G397	; Spindle 3 feedrate in RPM
	G497	; Spindle 4 feedrate in RPM

EXAMPLES:	G97	; Spindle 1 feedrate in RPM
------------------	------------	-----------------------------

The UNIDEX 600 Series controller provides the flexibility to specify spindle feedrates (the S word) in either revolutions per minute (**G96**) or surface units per minute (**G97**). **G97** allows the spindle feedrate for the given spindle to be specified in RPM.

The active **Gx96/Gx97** mode is indicated by bits 18-21 of the *Mode1* task parameter.

However, in either **G97** or **G96** mode, the spindle speed will be limited by the *MaxFeedRateIPM* machine parameter value.

Note, that in either **G97** or **G96** mode, the S word always specifies the spindle speed. If you change modes with the spindle running, the S word value will be reinterpreted by the new mode, and the spindle will accelerate/decelerate to the new speed immediately. Also note that you must have the *ExecuteNumSpindles* task parameter set properly to utilize more than one spindle.

5.27. Dominant Feedrate Overview (G98, G99)

A complex move contains both linear and rotary axes moving simultaneously in a contoured (**G1,G2,G3**) move. These types of moves create a number of special problems, in which one or more of the axes appear to be traveling at the wrong speed or acceleration, and the programmer must read and understand the following in order to properly perform complex moves. The following description and **G98/G99** mode applies only to complex moves. The first problem is that both the **F** and the **E** word speeds cannot be obeyed at once. Refer to the following example code fragment:



In all examples here, X, Y, Z are linear axes, while B is a rotary axis. Suppose, for all three examples, we begin at X=0, B=0, in **G90** mode.

```
G90 G0 X0 B0      ;Goto {0,0}; use absolute coordinates from now on
G1 X10 B90 F10 E1
```

If the F word is obeyed, then the **G1** move finishes in 1 minute. However, if the E word is obeyed, the move finishes in 15 seconds. The solution the Aerotech controller uses for this problem is to have a mode that specifies which commanded velocity is dominant. In the **G99** mode (linear dominant) the E word is ignored during moves involving both rotary and linear axes. Conversely, in the **G98** (rotary dominant) mode, the F word is ignored in complex moves. In each case, the type of axis whose speed word is ignored has its velocity calculated based upon the move time of the other axis type, so that all axes complete their moves simultaneously.

Unfortunately, due to the fact that the feedrate of the non-dominant axis is dictated by the dominant axis motion, the speed of the non-dominant axis may be exceeded. Examine the following code fragment, where X is a linear axis and B is a rotary axis.

```
G90 G0 X0 B0      ;Goto {0,0} use absolute coordinates from
                   ;now on
G99               ;Set linear axis dominant
G1 X0.00016666 B180 E1 F1  ;(0.00016666= 1/6000)
```

Here the user specified linear dominance for the **G1** command line, so the linear part of the move must travel at the linear feedrate. However, since the distance the X axis travels is small (relative to the B axis distance), the move must complete in a relatively short time (1/100 of a second in this case). This forces the B axis to travel at 3000 RPM, which may be too fast. The solution to this dilemma is to use the *MaxFeedRateIPM* / *MaxFeedRateRPM* machine parameters to limit the speed of the motion.

A related problem is the acceleration during complex moves. The dominant type of movement (rotary or linear) uses the acceleration instructed by the codes **G63** through **G68**. However, the non-dominant move is a slave to the dominant movement and may not obey its acceleration parameters. If the acceleration ramping is time based (see **G67**), then both types obey the acceleration time. However, if it is rate based (**G68**), then the non-dominant axis does not follow any specified acceleration.

Keep in mind that regardless of the **G99/G98** mode, if a CNC block moves only linear or rotary axes, then the F and the E words, respectively, will be used. **G98/G99** apply only to complex CNC blocks (ones that move rotary and linear axes simultaneously).

See the example below:

```
G99 G90 X1 B1 ; Here we use the F feedrate, because we are in G99 mode.  
G99 G90 B1 ; Here we use the E feedrate, even though we are in G99 mode.  
G99 G90 X0 B1 ; Here we use the E feedrate, even though we are in G99 mode.
```

The second and third examples are equivalent, even though the X axis is specified in the third example, because the X target is the same as our current location, so no X motion occurs. The active linear/rotary dominant mode is indicated by bit 4 of the *Mode1* task parameter.



5.27.1. Rotary Feedrate Dominant

G98

SYNTAX: **G98**

EXAMPLE: **G98**

The **G98** command specifies that the rotary feedrate (E) be considered dominant in coordinated motion commands moving both linear and rotary axes. When operating in this mode, the value of the E keyword determines the move duration, and the corresponding feedrate for the linear axis is computed from the duration of rotary axes motion. The active linear/rotary dominant mode is indicated by bit 4 of the *Mode1* task parameter.

Refer to the Dominant Feedrate Overview (Section 5.27) for more information.

EXAMPLE PROGRAM:

```
G98, G91          ; Make E-feedrate dominant  
G1 X10. Y72. F100. E10. ; Assuming X is linear and Y is rotary, use E-feedrate to calculate  
; move time of 0.2 minutes. Linear feedrate used will be 50  
; units/minute.
```



An axis must be designated as a linear or rotary axis by the *Type* machine parameter.

5.27.2. Linear Feedrate Dominant**G99****SYNTAX:** **G99****EXAMPLE:** **G99**

The **G99** command specifies that the linear feedrate (F) be considered dominant in coordinated motion commands involving both linear and rotary axes. When operating in this mode, the value of the F keyword determines the move duration, and the corresponding feedrate for the rotary axis is computed. This command is the **default**. The active linear/rotary dominant mode is indicated by bit 4 of the *Mode1* task parameter.



Refer to the Dominant Feedrate Overview (Section 5.27) for more information.

EXAMPLE PROGRAM:

```
G99          ;Make F-feedrate dominant.  
G1 X10. Y72. F100. E10.      ;Assuming X is linear and Y is rotary, use F-feedrate to  
                            ;calculate a move time of 0.10 minutes. Rotary feedrate used  
                            ;will be 2 RPM.
```

An axis must be designated as a linear or rotary axis by the *Type* machine parameter.



5.28. Spindle Shutdown Modes (G100, G101)

5.28.1. Disable Spindle Shutdown Mode

G100

SYNTAX: G100

EXAMPLE: G100

This G code disables the spindle shutdown mode, which is activated by the **G101** command. This command is the **default**. The spindle shutdown mode is indicated by bit 7 of the *Mode1* task parameter.

5.28.2. Enable Spindle Shutdown Mode

G101

SYNTAX: G101

EXAMPLE: G101

This command enables spindle shutdown mode. The spindle shutdown mode status is indicated by bit 7 of the *Mode1* task parameter.

When the spindle shutdown mode is disabled (**G100** is active), the spindle(s) will behave independently of other motion in the following ways:

Spindle motion will not be affected by a *feedhold* or *Abort* (spindle will keep rotating).

When the spindle shutdown mode is enabled, the spindle will react to feedhold and motion aborts like any other axis, except that a feedhold release, subsequent to a feedhold will not accelerate the spindle back up to speed.

5.29. Modal Velocity Profiling (G108, G109)

5.29.1. No Deceleration to Zero Velocity Between Moves G108

SYNTAX: G108

EXAMPLE: G108



The **G108** command is the inverse of a **G109** command. Meaning, it forces the controller to accelerate or decelerate between moves smoothly, as opposed to decelerating to zero velocity between consecutive contoured (**G1**, **G2**, **G3...**) moves. **G108** is a modal command. This command is the **default**. **G108** applies only to contoured motion. The active no-decel/force-decel mode is indicated by bit 5 of the *Mode1* task parameter.

When using **G108**, it is quite easy to construct sequences of moves that jerk one or more axes by commanding instantaneous acceleration/deceleration. See Corners on page 5-37 for details.



When using **G108**, if you want to blend two non-consecutive (other commands lie in-between) moves, there are some issues the CNC programmer needs to be aware of. See 5.40.3



5.29.2. Force Deceleration to Zero Velocity Between Moves G109

SYNTAX: G109

EXAMPLE: G109



The **G109** command performs the same function as a **G9** command, but operates modally. It causes the controller to decelerate to zero velocity between consecutive contoured (**G1**, **G2**, **G3 ...**) moves. Please see the **G9** command for more details. **G109** applies only to contoured motion. The active no-decel/force-decel mode is indicated by bit 5 of the *Mode1* task parameter.



5.30. Circular Direction Codes (G110, G111)

5.30.1. Normal Circular Interpolation

G110

SYNTAX: G110

EXAMPLE: G110

The UNIDEX 600 Series controller provides the user with the flexibility to change the orientation of the axes used for circular interpolation. By default, circular interpolation occurs as described in the description of the **G2/G3** and **G12/G13** commands, Plane Select (**G17/G18/G19** and **G27/G28/G29**) G-code groups, and the I/J/K keywords. Table 5-9 defines these relationships. This command is the **default**. The active inverse/normal circular orientation mode is indicated by bit 6 of the *Mode1* task parameter.

Table 5-9. G-Codes to Change Axes Used for Circular Interpolation

G-code	Description in G110 Mode
G2	Circular Clockwise Rotation - Plane 1
G3	Circular Counterclockwise Rotation - Plane 1
G12	Circular Clockwise Rotation - Plane 2
G13	Circular Counterclockwise Rotation - Plane 2

The **G110** command restores these default associations if a **G111** command has been previously executed.

5.30.2. Inverse Circular Interpolation**G111****SYNTAX:** G111**EXAMPLE:** G111

The UNIDEX 600 Series controller provides the flexibility to invert the direction of circular motion. By default, circular interpolation occurs as described in the **G2/G3** and **G12/G13** commands, the Plane Select (**G17/G18/G19** and **G27/G28/G29**) G-code groups, and the I/J/K keywords. The active inverse/normal circular orientation mode is indicated by bit 6 of the *Mode1* task parameter.



However, when operating in the inverse circular interpolation mode, the arc direction, plane and circle center point keyword associations are reversed. Table 5-10 summarizes the relationships that exist.

Table 5-10. Relationship of Arc Direction, Plane, & Circle Center point

G-code	Description in G111 Mode
G2	Circular Counterclockwise Rotation - Plane 1
G3	Circular Clockwise Rotation - Plane 1
G12	Circular Counterclockwise Rotation - Plane 2
G13	Circular Clockwise Rotation - Plane 2

The **G111** command sets these associations. These relationships are reset by the **G110** command.

5.31. Block Delete Mode (G112, G113)

The Block Delete mode may be toggled via the Shift F9 key. This allows a program to selectively skip CNC command lines which begin with the Block Delete operator ‘/’, when Block Delete is active. For example, in the following program fragment,

```
/ HOME X Y      ; skipped if block delete is active  
G1 X5 Y2 F200.  
/ G92 X0 Y0      ; skipped if block delete is active
```

the HOME and **G92** command lines in the above example, will not execute if the Block Delete mode is active. The second line containing the **G1** command line will always execute regardless of the state of the Block Delete mode state. Block delete may be toggled within a program via the **G112** and **G113** commands.



If you are single stepping through a CNC program, the controller will always stop prior to executing a line, even if that line is “skipped” due to a block delete.

5.31.1. Set Block Delete Mode

G112



The **G112** command sets the `TASKMODE1_BlockDelete` bit in the *Mode1* task parameter to 1. If this bit is true, the controller ignores subsequent CNC program lines having the block delete symbol at the beginning of the line. The **G113** command clears this mode. The block delete mode state is indicated by bit 8 of the *Mode1* task parameter.



5.31.2. Clear Block Delete Mode

G113

The **G113** command clears the `TASKMODE1_BlockDelete` bit in the *Mode1* parameter. This command is the **default**. The block delete mode state is indicated by bit 8 of the *Mode1* task parameter.

5.32. Optional Stop Mode (G114, G115)

5.32.1. Set Optional Stop Mode

G114

This command sets the TASKMODE1_OptionalStop bit in the *Mode1* task parameter to 1. If this bit is on, all subsequent **M1** codes are equivalent to **M0**. Otherwise, an **M1** does nothing. The **G115** command clears this mode. The optional stop mode state is indicated by bit 9 of the *Mode1* task parameter.



5.32.2. Clear Optional Stop Mode

G115

This command clears the TASKMODE1_OptionalStop bit in the *Mode1* task parameter. Refer to **G114** command for more details. This command is the **default**. The optional stop mode state is indicated by bit 9 of the *Mode1* task parameter.



5.33. Dry Run Mode (G116, G117)

When Dry Run mode is active the axes are moved at the feedrate specified by the *DryRunLinearFeedrateIPM* (or *DryRunRotaryFeedrateRPM*) task parameters regardless of the feedrates specified in the program. This mode has no effect on G0 commands or asynchronous motion commands. The Dry Run mode is enabled/disabled via the G116 / G117 commands, or via the Window List menu on the Manual or Run Pages. The Dry Run state is indicated by bit 13 of the *Mode1* task parameter.

This mode is normally used for verifying the movement of a tool over an area in which the workpiece (or fixture) has been removed.

The *MaxFeedRateIPM* and *MaxFeedRateRPM* machine parameter limits still apply and will limit the feedrates.



5.33.1. Dry Run Mode Enabled

G116

This command enables the *Dry Run Mode*. The dry run mode state is indicated by bit 13 of the *Mode1* task parameter.



5.33.2. Dry Run Mode Disabled

G117

This command disables the *Dry Run Mode*. The dry run mode state is indicated by bit 13 of the *Mode1* task parameter.



5.34. Servo Update Rate (G130, G131)**5.34.1. 4 Kilohertz Servo Update Rate** G130**SYNTAX:** G130**EXAMPLE:** G130

The **G130** command causes the UNIDEX 600 Series controller to update the servo loop 4000 times/sec (every .25 msec). This includes sampling the feedback device(s) and calculating a new command for the motor. When switching from 1K to 4K update, (**G131** to **G130**) the user must adjust the gains as shown below. This command is the **default**. This G code may be defined as a default on the *Task Initialization Page* of the U600MMI-NT/95.

$$\begin{aligned} K_P &= K_P * 4 \\ PGAIN &= PGAIN / 4 \end{aligned}$$

5.34.2. 1 Kilohertz Servo Update Rate G131**SYNTAX:** G131**EXAMPLE:** G131

The **G131** command sets the UNIDEX 600 Series controller to update the servo loop 1000 times/sec. (every 1 msec.). The normal process of sampling the feedback device(s) and updating the command 4000 times per second is reduced. When switching from 4K to 1K, the user must adjust the gains as shown below. This mode may be preferred on systems having a machine step size greater than 1 micron. The **G130** command sets the 4 kilo-hertz servo loop update mode. This G code may be defined as a default on the *Task Initialization Page* of the U600MMI-NT/95.

$$\begin{aligned} K_P &= K_P / 4 \\ PGAIN &= PGAIN * 4 \end{aligned}$$

5.35. Cutter Tool Offset Compensation Overview (G143, G144, G149)

Cutter tool Compensation will compensate for the length of your tool. The G143 adds the tool offset to the length and the G144 command subtracts the offset. The G149 command will cancel all offsets.

5.35.1. Activate Positive Cutter (Tool) Offsets G143

The **G143** command activates cutter offsets and cutter length compensation, and adds these values to the appropriate move targets. If you are already in cutter offsets compensation mode, this command is ignored. The lead-on move, is defined as the next contoured move on the same line as the **G143**, or on the line following the **G143**. You must be in **G1**, **G2** or **G3** mode to execute this command. The active state of this command is indicated by bit 12 of the *Status3* task parameter.



Normally, when entering (**G143**, **G144**) and exiting (**G149**) Cutter Offset Compensation, you provide a contoured move on the same line. For **G143**, **G144**, this move is called the “lead-on” move, for **G149** this is called the “lead-off” move. The lead-on and lead-off moves must be a contoured move.

Lead-on and lead-off moves must be carefully constructed, so as not damage the tool or part. Lead-on moves will ‘blend in’ the tool offsets gradually during the move. When a lead-on move is completed, the cutter offsets are fully applied.

If you do not provide a “lead-on” move on the same line as the **G143/G144**, then, it will use the next contoured move it executes, as the lead-on move. Any number of CNC statements that are not contoured moves (except **G41**, **G42**, **G40**, **G143**, **G144**, and **G149**) can be placed in-between the **G143/G144**, and the “lead-on” move. If it cannot find a next contoured move to use as a lead-on, it will not enter cutter offset compensation.

Cutter compensation is only intended for contoured moves (**G1**, **G2**, **G3**, **G12**, **G13**). You may command non-contoured motion on axes that are not part of the cutter compensation plane while in cutter compensation mode, however, this will lead to undesirable results, since cutter compensation has no effect on the other types of motion.



Cutter Offsets Compensation will not operate over multiple lines entered in MDI mode, as after each MDI line, the cutter compensation is canceled.



HOMING disables cutter offset compensation.





5.35.2. Activate Negative Cutter (Tool) Offsets

G144

G144 is identical to **G143**, except that the offsets (X and Y) and the tool length, as read from the tool file are subtracted from the actual coordinates. See **G143** for more details on this code. The active state of this command is indicated by bit 15 of the *Status3* task parameter.



5.35.3. Deactivate Cutter (Tool) Offsets

G149

The **G149** command exits tool offset mode, which deactivates the two tool offsets, and the tool length compensations. These three values are stored in the task parameters: CutterXOffset, CutterYOffset, and CutterLength. However, normally one assigns to these parameters via the Tool word.

The **G149** command has no effect on tool radius compensation, please see G40 to deactivate tool radius compensation.

Normally, when entering exiting Cutter Offset Compensation, you provide a contoured move on the same line, this is called the “lead-off” move.

Lead-off moves must be carefully constructed, so as not damage the tool or part.

Lead-off moves will ‘blend out’ the tool offsets and tool length gradually during the lead-off move, in order to remove compensation for the tool offsets. If you do not provide a “lead-off” move on the same line as the **G149**, then, it will use the next contoured move it executes, as the lead-off move. Any number of CNC statements that are not contoured moves (except **G41**, **G42**, **G40**, **G143**, **G144**, and **G149**) can be placed in-between the **G40**, and the “lead-off” move. If it cannot find a next contoured move to use as a lead-off, it will not remove the tool offsets compensation.

5.36. Scale Factor (G150, G151)

5.36.1. Clear Scale Factor

G150



SYNTAX: **G150**

EXAMPLE: **G150**

Cancels the **G151** Scale Factor command

5.36.2. Set Scale Factor

G151



The **G151** Scale Factor command will decrease or increase (scale) the part by the indicated value. Scaling is modal, **G150** disables scaling. A scale factor greater than 1 will increase the size of the part, a scale factor less than 1 will decrease the size of the part. The same scale factor will be applied to all axes bound to the task, except spindles, which will not be affected at all. You may also change each axes scale factor independently via its ScaleFactor machine parameter, however, this has serious limitations.

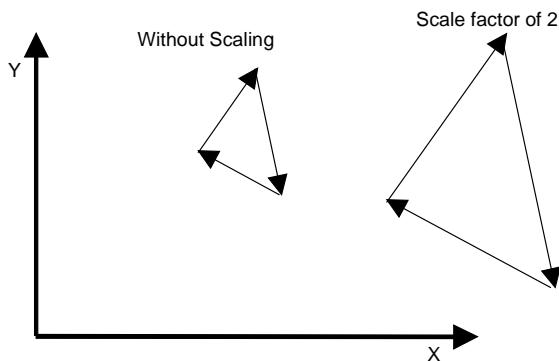


Figure 5-37. Scale Factor Example

G150 and **G151** must occupy their own line (no other CNC blocks on the line) You may verify the current state of Scaling, via the 'ScalingActive' bit of the *Status3* task parameter as viewed with the *AerStat.exe* utility, or by looking for '**G151**' in the active G code display in the lower left of the UNIDEX 600 MMI run or manual screens.

The **G151** Scalefactor command is not cumulative. That is, setting ScaleFactor=3, then immediately setting Scalefactor=2 is equivalent to setting Scalefactor=2.

The programmed velocities are unaffected by scaling. The position registers as displayed on the U600MMI, or reflected in any positions parameter (see Positions Overview) reflect the positions after scaling has taken place. After scaling, all target positions connected with motion, (X, Y and other axis names, as well as I, J, K and R) will be affected. Coordinates provided in asynchronous motion like MOVETO and INDEX are also affected by the scale factor. However, all other coordinates and parameters are unaffected by the scale factor, including those provided in G codes: **G92**, **G56** through **G59**, **G84**, and **G43**. Also unaffected by the scale factor are the coordinates provided in the TRACK and HANDWHEEL commands, and all tool lengths and diameters

Scaling of rotary axis coordinates is allowed also, however, this may cause rollover beyond 360 degrees.

5.36.3. The Scaling Center

Scaling using absolute coordinates (**G90**) has an extra complication: the scaling center. The following section elaborates on the scaling center. Keep in mind the scaling center is only relevant when programming in absolute coordinates, therefore this section can be ignored when programming in **G91**.

Scaling always occurs “around” the current point. So parts generated in absolute coordinates will be centered around the point at which the scaling was activated. This point is called the scaling center as illustrated below.

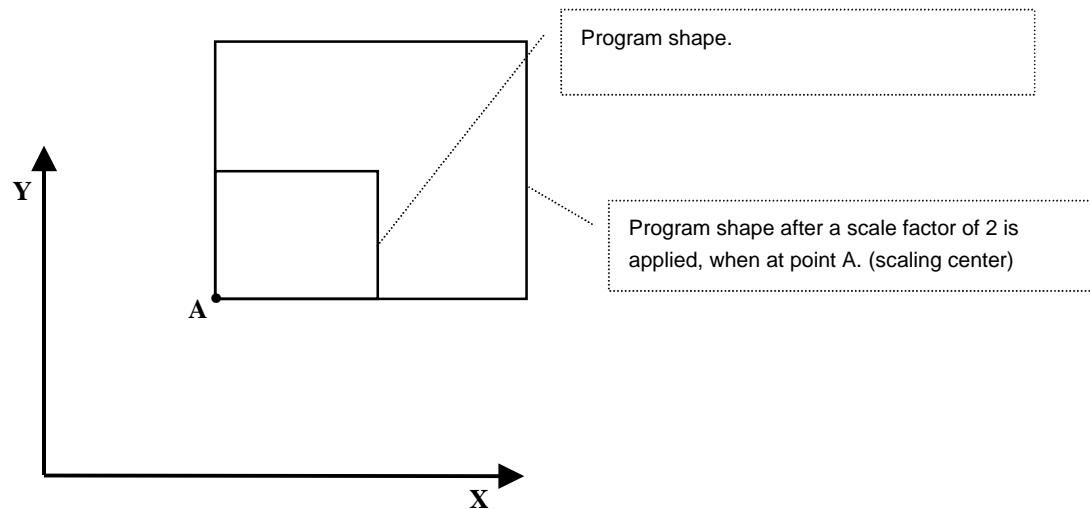


Figure 5-38. Scaling Center Illustration

When using fixture offsets or presets (**G92**) with absolute coordinates (**G90**) and scaling, the scaling center is measured in preset coordinates, as illustrated below.

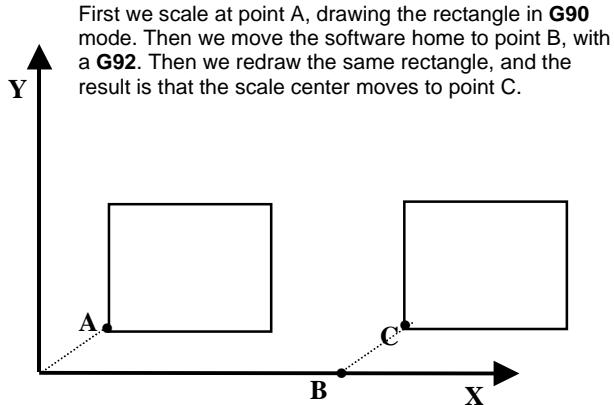


Figure 5-39. Scaling Center Illustration 2

Scaling while in G90 mode with software presets may be confusing, due to the scaling center. It is recommended that, in this situation you scale while at the absolute coordinate {0,0}. Then the scaling center will always be at the software home. You may change the scaling center by repeating a G151 with the same scale factor at a different point. Note that, when scaling has been applied, it should be removed via the G150 command. Rescaling via another G151 command, with a factor of one will not remove scaling properly because it will not reset the scaling center to {0,0}.

5.37. Suspend All Fixture Offsets

G153

SYNTAX: **G153~< CNC Block >**

EXAMPLE: **G153 G1 X10**



The **G153** command suspends fixture offsets (i.e. performs a **G53**) for one CNC block, the move on the same line as the **G153**. A **G153**, with no move on that line, has no effect. Therefore, a CNC block containing a **G153** command will ignore all active fixture offsets.



5.38. Rotary Axis Acceleration Rates (G165, G166)

5.38.1. Set Acceleration Rate (for Rotary Type Axes) G165

SYNTAX: **G165~F<fExpression>** ; Where the expression is acceleration rate

G165~P<fExpression> ; Where the expression is acceleration rate

EXAMPLE: **G165 F20.0** ; Sets acceleration rate to 20 deg/sec/sec

When accelerating a rotary type axis, using the rate based parameters (**G68**), the UNIDEX 600 Series controller uses the *AccelRateDPS2* task parameter as the acceleration value. This parameter can also be set (for the current task) with a **G165** command. This rate is specified in degrees/second/second. The **G165** setting is ignored if the acceleration is time based (**G67**). The accel rate may follow either an F or P keyword, as shown in the syntax definition above.

Note that the rate is applied over the vectorial distance of the move. For example, if a **G90 B1 C1** executes, then the specified rate is correct over the distance of 1.414 inches, not 1 inch.

G165 only applies when all the axes in the move are of rotary type or if the move contains both rotary and linear axes and the dominance of the move is rotary (refer to **G98/G99** Overview in section 5.27 for details on axes dominance). See **G65** if the axes are linear or if the dominance is linear.



5.38.2. Set Deceleration Rate (for Rotary Type Axes) G166

SYNTAX: **G166~F<fExpression>** ; Where the expression is deceleration rate

G166~P<fExpression> ; Where the expression is deceleration rate

EXAMPLE: **G166 F20.0** ; Sets deceleration rate to 20 deg/sec/sec

When decelerating a rotary type axis, using the rate based parameters (**G68**), the UNIDEX 600 Series controller uses the *DecelRateDPS2* task parameter as the deceleration value. This parameter can also be set (for the current task) with a **G166** command. This rate is specified in degrees/second/second. The **G166** setting is ignored if the deceleration is time based (**G67**). The decel rate may follow either an F or P keyword, as shown in the syntax definition above.

Note that the rate is applied over the vectorial distance of the move. For example, if a **G90 B1 C1** executes, then the specified rate is correct over the distance of 1.414 inches, not 1 inch.

G166 only applies when all the axes in the move are of rotary type or if the move contains both rotary and linear axes and the dominance of the move is rotary (refer to **G98/G99** Overview in section 5.27 for details on axes dominance). See **G66** if the axes are linear, or if the dominant axes type is linear.

5.39. Block Delete2 Mode (G212, G213)

The Block Delete2 mode may only be activated under CNC program control. This allows a CNC program to selectively skip CNC command lines which begin with the Block Delete 2 operator '/2, when Block Delete2 is active. For example, in the following program fragment,

```
/2 HOME X Y      ; skipped if block delete 2 is active  
G1 X5 Y2 F200.  
//2 G92 X0 Y0    ; skipped if block delete or block delete 2 is ; active
```

the HOME and **G92** command lines in the above example, will not execute if the specified Block Delete mode is active. The second line containing the **G1** command line will always execute regardless of the state of the Block Delete mode state. Note, that the **G92** CNC program line will not execute if either Block Delete or Block Delete 2 is on. Block Delete2 may be toggled within a CNC program via the **G212** and **G213** commands, but cannot be toggled from the U600 MMI.

If you are single stepping through a CNC program, the controller will always stop prior to executing a line, even if that line is “skipped” due to a block delete.



5.39.1. Set Block Delete2 Mode

G212

The **G212** command sets bit 22 in the *Mode1* task parameter to 1. If this bit is true, the controller ignores subsequent block delete2 lines. See *Block Delete2* for more details. The **G213** command clears this mode.

MODAL

5.39.2. Clear Block Delete2 Mode

G213

The **G213** command clears bit 22 in the *Mode1* parameter. See *Block Delete2* for more details. This command is the **default**.

MODAL

5.40. CNC Block Look-Ahead (G300, G301)

Look-ahead is the situation where the controller is about to execute a contoured (**G1 / G2 / G3**) move, but needs information on the next contoured move (**G1/ G2 / G3**) before it can begin the first move. Look-ahead occurs only in the five cases shown below (Table 5-11), the first of which is by far the most common. Reference each G code for more information on how look-ahead is used in each particular case. Normally, the controller will look-ahead only one move, but multiple CNC block look-ahead may be enabled via the **G301** command. The user is strongly advised to read and understand the following, particularly the situations concerning decel forcing and look-ahead failure, in order to achieve the desired results.

Table 5-11. The Five Look-Ahead Cases

Case	Reason Required	Next Move Found	If Decel Forced	Failed to Find Next Move
1. G108 mode, and no G8 / G9 on line	Decel at end of move?	No decel	Yes decel	Yes decel
2. <i>BlendMaxAccelLinearIPS2</i> or <i>BlendMaxAccelCircleIPS2</i> not zero	Decel at end of move?	No decel	Yes decel	Yes decel
3. <i>ChordicalSlowDownMsec</i> not zero	Decel at end of move?	No decel	Yes decel	Yes decel
4. In G21 / G22 mode (Normalcy)	Norm. align. needed ?	Yes align move	Yes align move	No align move
5. In G41 / G42 mode (Cutter comp.)	Link move needed?	Yes link move	Yes link move	No link move

In all cases listed in column one in the table above, the controller will look-ahead to the next CNC program block(s) for the next **G1/G2/G3** move, in order to answer the question shown in the second column above: “Reason Required.” There are three possible answers, as shown in columns 3, 4 and 5 in the table above. The paragraphs below provide more details.

It is important to keep in mind that if none of the cases described in the first column are true, the controller will not look-ahead. The programmer should keep in mind that look-ahead takes CNC execution time, and if that is in demand (see Motion Queue Starvation) that these situations should be avoided.

Look-ahead does not require that the next move immediately follow the first move, in order for it to find the second move. It will look past non-motion statements such as assignments, to find the second move (and the third and fourth move, and so on, if in **G301** mode, and cases 1, 2 or 3 above apply). However, there are definite limitations on which statements the controller is able to look past.

There are situations where the look-ahead process can “fail” to find a next move (for example, if the current move is the last move in the program). In these cases the controller follows the action shown in the last column of the table above (no warning or error messages will be generated). In addition, there are cases where although there is a next move, there exists a intervening decel forcing condition between the current move and the next move (for example if a G4 lies between the two moves. For the most common uses of look-ahead (cases 1,2 and 3 above) decel forcing” cases are identical to look-ahead failure: the controller forces a deceleration at the end of the move. However for cases 4. and 5, “decel forcing” cases are not considered failures (see column three in table above). For example, Normalcy (case 4) demands that a normalcy alignment move be inserted in-between any two moves at a right angle. If the current move is the last move in the program, this is a “look-ahead failure”, and there is no normalcy alignment after the current move. However if a **G4** lies between the two moves, a normalcy alignment move will still takes place.

5.40.1. Disable Multi-Block Look-Ahead G300

The **G300** command disables the multiple CNC block look-ahead process enabled by the **G301** command.



5.40.2. Enable Multi-Block Look-Ahead G301

The **G301** command enables the multiple CNC block look-ahead process. Normally, the controller will look ahead only one block. However, when in **G301** mode, the number of CNC program blocks that will be used within the look-ahead process is determined by the *MaxLookAheadMoves* task parameter. The current status of multi-block look-ahead is reflected by bit #24 of the *Mode1* task parameter.



CNC block look-ahead is the process of examining future CNC program lines to see if certain conditions will occur and to take appropriate actions when these conditions do occur. It must be emphasized that these conditions, and the actions taken when they are detected are the same, whether you are looking one move ahead (**G300**) or multiple moves ahead (**G301**). **G301** just increases the scope of the controllers pre-processing. Furthermore, it must be emphasized that the look-ahead process has certain limitations, that may not be obvious.

It is recommended that **G311** be used with **G301**. Multi -block look ahead increases the number of computations made by the controller each move, which can lead to Profile Queue Starvation. However, **G311** can be used to speedup multi-block look-ahead, so as to prevent starvation. **G311** does have two limitations.

5.40.3. CNC Block Look-Ahead Conditions that Force (G9) Deceleration

Even if the user requests profile blending (does not place a G9 or G8 on the line, and is in G108 mode), there are some exceptions where the controller forces deceleration at the end of the move (and subsequent acceleration at the beginning of the next move).

Multiple CNC block look-ahead may be enabled via the G301 command. These cases are detailed below:

- A. Perhaps the simplest example of an enforced G9 move is when the user is running in single step mode. In this case, clearly the motion must decelerate to zero velocity between the two contoured moves.
- B. The controller will force a G9 when it detects any time consuming activity that occur in-between the two moves to be blended. If the controller tried to blend moves in such situations, then a sudden break or drop in the velocity command would occur, so to avoid this, the controller forces deceleration. The controller will also make other statements force deceleration that might cause inconsistent motion, if you tried to blend them through them. A list of all statements that force deceleration are shown below:

Dwell Commands	- G4, WAIT, M00, M01 (but only if optional stop is on)
Any callback command	- DATAxxx, FILExxx, MSGxxx, LOADCAMTABLE, CALLBACK, CALLDLL, etc.
Other motion codes	- G0, G45, G46, G47, G84, G93, G94, G95, Home, or any Asynchronous motion commands
Parameter sets	- POS, RotateX, RotateY, RotateAngleDeg, IgnoreAxesMask, DryRunLinearFeedRateIPM, DryRunRotaryFeedRateRPM
Other	- G83, G84, G51, PROBE, all PSO commands
Statements causing lookahead failure (5.40.4).	

- C. The following modal commands will also force deceleration, but only if they change the current mode. For example, if a G108 command is executed between the two moves, then it forces deceleration if and only if the current mode is not G108.

Blended Motion	- G108, G109
Accel/Decel	- G63, G64, G67, G68

5.40.4. CNC Block Look-Ahead Failures

- A. The controller gives up after searching a predefined number of lines, if it can not find the next move (look-ahead fails). This is determined by the MaxLookAheadMoves task parameter.
- B. The controller gives up looking for the next move, if it encounters any of the following statements. As noted below, the controller can not look-ahead through FARCALLs or FARJUMP's.

Program control	- M02, M30, M47, FARCALL, FARJUMP
Axis Control Commands	- MAP, BIND, CAPTURE, RELEASE, FREE

Camming Commands - TRACK, SYNC, CFGMASTER, SLEW, AFCO

Other - G53 – G59, G92, M41, M42

Dominant Feedrate Mode - G98, G99 (only if the mode changes)

- C. The controller does not evaluate global, task or IO variables, when looking ahead, therefore, if the next move is based upon these variables, the move may be in error. This obscure situation affects only cases 2, 3, 4 and 5 in the table. The following code fragment illustrates this potential error:

```
$GLOBO = 100
G90 G1 X$GLOBO      ; First move
$GLOBO = 200
G90 G1 X$GLOBO      ; Second move
```

There are many other similar examples in which the look-ahead will error when contoured moves use global or task variables as the move target. The way to avoid these situations is by using program variables. Look-ahead does correctly evaluate these, while looking for the next move.

- D. When in the Program Queue Mode, look ahead will fail if it runs out of downloaded CNC program lines before finding the next move.
- E. The controller will not execute canned functions, ON's or ONGOSUBS while doing look-ahead.

In some situations the look-ahead process will find an error on a line that is ahead of the current line (the most common problem is an improperly written **G2/G3**). This can be confusing, because the line the controller is on is not the line where the error is seen (see example below).

```
G108
G1 X 10      ; controller generates error on this line: "no circular offsets"
G2 X5      ; this is the bad line, no circular offsets
```

In this simple program fragment, the controller will generate the error on the second line. However, it will indicate line 3 in the TaskWarning task parameter, indicating the third line is really the culprit.

Multiple block look-ahead may be enabled via the **G301** command.

5.41. High Speed Machining (G310, G311)

5.41.1. Disable High Speed Machining

G310



The **G310** command disables the High Speed Machining mode enabled by the **G311** command.

5.41.2. Enable High Speed Machining

G311



The **G311** command enables high speed buffering, which can be used to speed-up the multiple CNC block look-ahead process. In this mode, the controller will store results of previous move's look-ahead computations, and use them for future computations. The speedup can be such that the controller will operate as fast as it would with only single block look-ahead (no **G301**).

However, this mode has limitations.

The current status of high speed buffer mode is reflected by bit #26 of the *Model* task parameter.

G311 has no effect, unless **G301** is active.

5.41.3. High Speed Machining Limitations

If the MFO is changed during program execution, the speedup process is impaired, and **G311** may have little or no speedup effect.

If a “jump” is encountered (includes if, goto, while, repeat, farcall, and call statements) the controller will abandon all pre-computed results, since it can no longer be aware of what might have changed between compute time and usage time. This will cause **G311** to have little or no speedup effect, during and for some time after the “jump” is encountered.

If a move uses variables in the target position or speed, and the value of these variables is “externally” changed during program execution, then the look-ahead process may produce wrong results (force slowdowns when none necessary, or not forcing them when they are necessary). This caveat applies only when the variable values are “externally” changed, i.e. from the variable inspector, or from another task. Variable values changed WITHIN the current CNC task are OK.

5.41.4. Continue when Velocity command is Zero

G360

SYNTAX: G360

EXAMPLE: G360



In G360 mode the controller will never wait for in-position, before proceeding to the next CNC block, it will continue on with the next CNC program line when the velocity command is zero. The modal status of the G360 command is indicated by bit 27 of the Mode1 task parameter.

5.41.5. Wait till In-Position

G361

SYNTAX: G361

EXAMPLE: G361



Synchronous motion commands that decelerate to zero velocity do not allow CNC program execution to continue to the next program block until motion is done. This applies to the all of the following types of motion:

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 slowdown to zero velocity at the end of the move

ENDM

These motion commands are done when the velocity command reaches zero (the axes are not being commanded to move). However, when G361 mode is active, the controller will also wait for all axes to be in position, before continuing on with the next CNC program block. The modal status of the G361 command is indicated by bit 27 of the Mode1 task parameter.

5.42. M-codes

In general, M-codes perform miscellaneous I/O functions from within parts programs. The function associated with each M-code vary significantly. However, in most cases they either control program execution, spindle or feedrate operation. Custom M codes may be assigned.



This section does not address the use of M-codes for input or output functions. Refer to Chapter 7 for this functionality. This section only details those M-codes used for program and spindle control.

5.42.1. Program Stop

M0

SYNTAX: M0

EXAMPLE: M0

Pauses program execution. Pressing the cycle start button restarts program execution.

5.42.2. Optional Stop

M1

SYNTAX: M1

EXAMPLE: M1

The functionality of this command depends upon the current state of the TASKMODE1_OptionalStop bit in the *Mode1* task parameter (see **G114 / G115**). If this bit is on, the task responds to this M-code as described in the **M0** command. Otherwise, this M-code is ignored. The default condition is off, meaning **M1** is ignored.

Use of this command permits the user to interrupt program execution at a specific point when an abnormal condition occurs, but does not require operator intervention under normal circumstances.

5.42.3. End of Program

M2

SYNTAX: M2

EXAMPLE: M2

This command ends the CNC program. The user cannot continue after a **M2**, they must restart the program.

5.42.4. Spindle On Clockwise

M3, M23, M33, M43



SYNTAX: **M3** ; Spindle #1 On Clockwise
M23 ; Spindle #2 On Clockwise
M33 ; Spindle #3 On Clockwise
M43 ; Spindle #4 On Clockwise

EXAMPLE: **M3** ; Spindle #1 On Clockwise

This command causes the specified spindle to begin rotating clockwise (CW). It will accelerate (as defined by the *ACCELMODE* axis parameter) up to the target speed. The target speed of the spindle is determined by the S keyword, as well as the current operational mode or MDI mode in respect to the Spindle Speed G-code group. Up to four spindles may be defined. If the spindle is rotating CW no action will take place, if it is rotating CCW then it will decelerate to zero and accelerate CW up to the S word. This M code will enable the drive if it is not already enabled. M codes **M3/M23/M33/M43** set bits 0 through 3 of the *Status2* task parameter, respectively. This command is complete only when the spindle has accelerated up to the commanded speed.

Note that you must have the *ExecuteNumSpindles* task parameter set properly to utilize more than one spindle.



5.42.5. Spindle On Counterclockwise

M4, M24, M34, M44



SYNTAX: **M4** ; Spindle #1 On Counterclockwise
M24 ; Spindle #2 On Counterclockwise
M34 ; Spindle #3 On Counterclockwise
M44 ; Spindle #4 On Counterclockwise

EXAMPLE: **M4** ; Spindle #1 On Counterclockwise

This command causes the spindle specified to begin rotating counter-clockwise (CCW). It will accelerate (as defined by the *ACCELMODE* axis parameter) up to the target speed. The target speed of the spindle is determined by the S keyword, as well as the current operational mode or MDI mode in respect to the Spindle Speed G-Code group. There are four spindles available. If the spindle is rotating CCW no action will take place, if it is rotating CW then it will decelerate to zero and accelerate CCW up to the S word. This M code will enable the drive if it is not already enabled. M codes **M4/M24/M34/M44** set bits 0 through 3 of the *Status2* task parameter, respectively. This command is complete only when the spindle has accelerated up to the commanded speed.

Note that you must have the *ExecuteNumSpindles* task parameter set properly to utilize more than one spindle.



**5.42.6. Spindle Off****M5, M25, M35, M45**

SYNTAX: **M5** ; Spindle #1 Off
M25 ; Spindle #2 Off
M35 ; Spindle #3 Off
M45 ; Spindle #4 Off

EXAMPLE: **M5** ; Spindle #1 Off

This command causes the spindle to decelerate to zero velocity when this command is executed. (as defined by the *DECELMODE* axis parameter). The spindle stops moving by being disabled (spindles with large inertial loads will free-wheel to a stop). There are four spindles available, per task. If the spindle is not rotating, no action takes place. M codes **M5/M25/M35/M45** clear bits 0 through 3 of the *Status2* task parameter, respectively. This command is complete when the commanded spindle velocity has reached zero, independent of the in-position limit.

**5.42.7. Spindle Off/Reorient****M19, M219, M319, M419**

SYNTAX: **M19** ; Spindle #1 Off/Reorient
M219 ; Spindle #2 Off/Reorient
M319 ; Spindle #3 Off/Reorient
M419 ; Spindle #4 Off/Reorient

EXAMPLE: **M19** ; Spindle #1 Off/Reorient

This command causes the spindle to move to the zero position at the current S feedrate. This M code will decelerate the spindle to zero, then move it back to the zero position in the shortest distance. It will not disable the axis. This command should only be used for servo driven (closed loop) spindle axes. M codes **M19/M219/M319/M419** set bits 0 through 3 of the *Status2* task parameter, respectively.

5.42.8. Restart Program Execution and Wait for Cycle Start M30

SYNTAX: **M30**

EXAMPLE: **M30**

Program execution immediately returns to the first executable line of the first program (if this is encountered in a nested, called program, execution begins at the top of the first program and all nested calls, while loops, etc., are canceled) and awaits a cycle start command to resume program execution. All modal information remains unchanged and user defined (DVAR) variables are not redefined; their value remains unchanged.

5.42.9. Machine Lock Mode

The Machine Lock mode sets the SIMULATION axis parameter to 1 for all axes bound to the task. This mode differs from simulation mode, in that whenever the machine is unlocked, the position registers are updated with the current simulated positions. In essence, a preset (or offset) is introduced into the system. The Machine Lock mode is enabled/disabled via the **M41 / M42** M codes, or via the Window List menu on the Manual or Run Pages. The Machine Lock state is indicated by bit 25 of the *Mode1* task parameter.

Note: When a HOME command is executed in this mode, the position is set to the value of the *HomeOffsetInch* (*HomeOffsetDeg* for rotary axes) machine parameter, and the “homed” bit of the SERVOSTATUS axis parameter will be set. Upon exiting this mode the homed bit will be cleared if the axis was homed while in this mode.

5.42.10. Machine Lock Enabled

M41

This command enables the *Machine Lock* mode. The active state of the machine lock mode is indicated by bits 25 of the *Mode1* task parameter.



5.42.11. Machine Lock Disabled

M42

This command disables the *Machine Lock* mode. The active state of the machine lock mode is indicated by bits 25 of the *Mode1* task parameter.



5.42.12. Restart Program Execution

M47

SYNTAX: **M47**

EXAMPLE: **M47**

Program execution immediately returns to the first line of the first program (if this is encountered in a nested, called program, execution begins at the top of the first program, and all nested calls, while loops, etc., are canceled) and begins execution from there. All modal information remains unchanged.

**5.42.13. Feedrate Override Lock** M48**SYNTAX:** M48**EXAMPLE:** M48

This command disables usage of feedrate override controls, located within the U600 MMI. It sets the TASKMODE1_MFOLock bit (11) in the *Mode1* task parameter.

**5.42.14. Feedrate Override Unlock** M49**SYNTAX:** M49**EXAMPLE:** M49

This command is intended to enable usage of feedrate override controls; the ones located within the U600 MMI. It clears the TASKMODE1_MFOLock bit (11) in the *Mode1* task parameter. This command is the **default**.

**5.42.15. Spindle Feedrate Override Lock** M50**SYNTAX:** M50**EXAMPLE:** M50

This command disables usage of spindle feedrate override controls; the ones located within the U600 MMI. It sets the TASKMODE1_MSOLock bit (12)in the *Mode1* task parameter. It disables the MSO for all spindles.

**5.42.16. Spindle Feedrate Override Unlock** M51**SYNTAX:** M51**EXAMPLE:** M51

This command enables usage of spindle feedrate override controls, the ones located within the U600 MMI. It clears the TASKMODE1_MSOLock bit in the *Mode1* task parameter. It enables the MSO for all spindles. This command is the **default**.

5.42.17. Loop over Near Call to Subroutine**M97****SYNTAX:** **M97 L<expression>~P<integer>****EXAMPLE:** **M97 L10 P2000**

M97 calls a subroutine, a specified number of times. The call must be a near call (the subroutine must be defined with a DFS, and in the same CNC program the **M97** is used in). See the **M98** command for looping over far calls. You cannot pass parameters to the subroutine with the **M97** command. See the CALL command to call subroutines with parameters.

The L word specifies the number of times to call the subroutine. If the L word is omitted, the subroutine will be called once.

The P word identifies the subroutine. The beginning of the subroutine must be defined by an N code label that matches the integer specified in the P word (see the example below). Furthermore, you must specify the `#makencodeslabels` statement at the top of your program. You must end the subroutine with a RETURN statement, or the last statement of the subroutine must be the last statement in the CNC program.

EXAMPLE:

```
#makencodeslabels           ; You must have this statement at top of program !
...
...
$GLOBAL0 = 0
M97 L10 P2000           ; at this point, $GLOBAL0 is now 20
M02
...
N2000 $GLOBAL0 = $GLOBAL0 + 2
N2001 RETURN
```

5.42.18. Loop over Far Call to Subroutine**M98****SYNTAX:** **M98 L<expression>~P<integer>****EXAMPLE:** **M98 L10 P2000**

M98 calls a subroutine, a specified number of times. The call is a far call, and consequently the subroutine may be within any CNC program. However, see **M97** for a simpler way of looping over subroutines that in the same file as the calling statement. You cannot pass parameters to the subroutine with the **M98** command. See the FARCALL command to call subroutines with parameters.

The L word identifies the number of times to call the subroutine. If the L word is omitted, the subroutine will be called once.

The P word identifies the subroutine. The beginning of the subroutine must be defined by an N code label, that matches the integer provided in the P word (see the example below). Furthermore, you must have a `#makencodeslabels` statement at the top of the CNC program containing the called subroutine. You must end the subroutine with a RETURN statement, or the last statement of the subroutine must be the last statement in the CNC program.

EXAMPLE:

```

; File1.pgm
...
$SIRITASK1 = "file2.pgm"
$GLOBAL0 = 0
M98 L10 P2000
; at this point, $GLOBAL0 is now 20

; File2.pgm

#makencodeslabels ; You must have this statement at the top of program !
...
N2000 $GLOBAL0 = $GLOBAL0 + 2
N2001 RETURN

```

**5.42.19. Spindle On Clockwise Asynchronously****M103, M123,
M133, M143**

SYNTAX: **M103** ; Spindle #1 On Clockwise
M123 ; Spindle #2 On Clockwise
M133 ; Spindle #3 On Clockwise
M143 ; Spindle #4 On Clockwise

EXAMPLE: **M103** ; Spindle #1 On Clockwise

This command causes the specified spindle to begin rotating clockwise (CW). The speed of the spindle is determined by the S keyword, as well as the current operational mode or MDI mode in respect to the Spindle Speed G-code group. Up to four spindles may be defined. If the spindle is rotating CW no action will take place, if it is rotating CCW then it will decelerate to zero and begin accelerating CW up to the S word. It will not wait for the spindle to accelerate up to full speed. This M code will enable the drive if it is not already enabled. M codes **M3/M23/M33/M43** set bits 0 through 3 of the *Status2* task parameter, respectively.

**5.42.20. Spindle On Counter-Clockwise Asynchronously****M104, M124,
M134, M144**

SYNTAX: **M4** ; Spindle #1 On Counter-Clockwise
M24 ; Spindle #2 On Counter-Clockwise
M34 ; Spindle #3 On Counter-Clockwise
M44 ; Spindle #4 On Counter-Clockwise

EXAMPLE: **M4** ; Spindle #1 On Counter-Clockwise

This command causes the spindle specified to begin rotating counter-clockwise (CCW). The speed of the spindle is determined by the S keyword, as well as the current operational mode or MDI mode in respect to the Spindle Speed G-Code group. There are four spindles available. If the spindle is rotating CCW no action will take place, if it is rotating CW then it will decelerate to zero and begin accelerating CCW up to the S word. It will not wait for the spindle to accelerate up to full speed. This M code will enable the drive if it is not already enabled. M codes **M4/M24/M34/M44** set bits 0 through 3 of the *Status2* task parameter, respectively.

▽ ▽ ▽

CHAPTER 6: EXTENDED COMMANDS

In This Section:	Page
• Introduction.....	6-1
• Motion with Extended Commands	6-2
• Host vs. Axis Processor Based Commands	6-2
• RS-447 Extended Commands.....	6-4

6.1. Introduction

In addition to G-code programming, the UNIDEX 600 Series controller provides a set of commands that permit the user to control program flow and perform other miscellaneous functions. These commands are the RS-447 extended command set, or the extended command set. This chapter contains a discussion of each extended command.

Table 6-3 is an alphabetical listing and summary of all extended commands. In addition, all of the commands within this chapter are in alphabetical order. Many commands have alternate forms, for example, the GOTO and JUMP keywords refer to the same command. In the syntax and examples only one form is shown, but the implication is, the user can use the other form as well.

We strongly recommend the user read Chapter 5, Sections 5.1.1. through 5.1.9., before undertaking any motion from the CNC.



The user can freely mix extended command code lines and G-code lines in a program, but not within the same CNC block (program line).

All “PSOx” commands (e.g., PSOD, PSOP, etc.) require Aerotech’s optional PSO-PC, Position Synchronized Output Board (Laser Firing Card).



The reader should note that a number of basic CNC language elements used in extended commands are described in Chapter 3, not here. Table 6-1 directs the user on where to find descriptions of these items.

Table 6-1. Where to Find Details

Term	An Example	Another Example	Reference
<fExpression>	7*6+\$GLOB0	7.8	Section 3.7
<CNCMask>	X Y z	x	Section 3.4
<axisPoint>	X55.6 Z5	Z\$GLOB0	Section 3.6.1.2
Motion Details	n/a	n/a	See 5.1.1 – 5.1.9

Table 6-2 details the extended command categories.

Table 6-2. Extended Command Categories

Controller Based	Examples
Asynchronous Motion	STRM, ENDM, INDEX, OSC, HOMEASYNC, MOVETO
Synchronous Motion	HOME, REF, (All motion G codes)
Program Control	IF, ELSE, WHILE, FARCALL, GOTO, DVAR, RETURN ...
Axis Control	MAP, BIND, FREE
Continuous Monitoring	ON, ONGOSUB
Miscellaneous	PROBE, HANDWHEEL
Callback Based	Examples
File Handling	FILEOPEN, FILEWRITE, FILECLOSE, FILEREADINI...
Data Display	MSGxxx Commands
External .exe Execution	EXE, CALLDLL
Data Collection	DATASTART, DATASTOP
PSO-PC Card Based	Examples
Laser Firing, etc.	PSOD, PSOF, PSOP, PSOS, PSOT

6.2. Motion with Extended Commands

The UNIDEX 600 controller can perform two types of motion: synchronous motion and asynchronous motion. All extended motion commands except **HOME** use asynchronous motion. Please see Chapter 5, for information on the differences between the two types of motion.

6.3. Host vs. Axis Processor Based Commands

There are two types of extended commands: axis processor (controller) based and callback based. All G-codes and most extended commands are controller based. Only a small number of extended commands are callback based.

6.3.1. Axis Processor Based Extended Commands

Axis processor (controller) based commands execute exclusively on the controller. After the host PC compiles, downloads, and begins program execution, the program runs independently on the controller. These include commands in the following categories: asynchronous motion, synchronous motion, program control, and continuous monitoring. All G-code and extended commands except for callback commands execute this way.

6.3.2. Host Based Extended Commands

Callback commands are special commands that require assistance from the host PC. For example, the controller cannot access files on the PC, so when a file access command is encountered on the controller, it passes the command back to the host PC to execute.

Another example is the extended commands that display text. Since the controller cannot write to the display, it must pass the data back to the host PC to display on the screen.

The callback command is initiated by an interrupt to the PC. The process can be summarized as follows:

1. The UNIDEX 600 begins execution of a callback command.
2. An interrupt is generated by the UNIDEX 600 card.
3. The device driver detects the interrupt and signals the host application (normally the MMI600-NT).
4. A host application (U600MMI-NT/95) retrieves the callback data and carries out the appropriate function.
5. Upon completion of the callback command, the host application (MMI600-NT) tells the UNIDEX 600 to continue.
6. The UNIDEX 600 executes the next program line.

For a more detailed description of callback commands, refer to the Interrupt and Event Handling section in the UNIDEX 600 Series User's Guide, P/N EDU157. For information on implementing a custom callback command, see TN0004, in the online help file.

Some callback commands are: MSGxxx, EXE, CALLDLL, and DATASTART.

All Callback commands will set the ErrCode task parameter, if an error occurs during the commands execution.



6.3.2.1. Time-outs

A host application program must be running on the PC in order to respond to the callback. The host program normally used for this purpose is the UNIDEX 600 MMI. If there is no host program running, there is no response to the interrupt and the controller times out after a specific time. The *CallbackTimeoutSecs* task parameter can adjust the time.

Another common occurrence is the interrupt on the card does not agree with the interrupt registered in the registry in Windows NT/95. This also causes a time-out.

6.3.2.2. Error Returns from the CallBack Commands

The host PC must have a method of returning error information to the controller. For example, if the **FILEOPEN** failed because a file did not exist, the CNC program must have a way to recognize this and the *ErrCode* task parameter serves this purpose. If the *ErrCode* value is non-zero after a callback command, then an error occurred during the execution of the callback command on the host PC. Refer to the specific callback command to find out what the error codes (non-zero) mean.

It is important to understand that the controller is not inherently aware of the *ErrCode* value. In other words, the CNC program continues after a **FILEOPEN** command, regardless of whether the **FILEOPEN** failed and set the *ErrCode*. It is the CNC programmer's responsibility to check the *ErrCode* in the CNC program and direct the proper error action. See the example in the following section for clarification.

6.3.2.3. Return Values from the Callback Commands

In addition to the error status, some callback commands return information in a floating point value. For example, a **FILEOPEN** command must return the fileHandle used by any subsequent **FILEWRITE** commands. This information can be captured by assigning the extended command to a variable of the programmer's choice. The example below shows two methods of writing the syntax for a callback command; the second syntax captures the return variable.

SYNTAX: <*callbackCommand*>

<*fVariable*> = <*callbackCommand*>

Some callback commands require the return variable, in others the user cannot have a return variable, and in others it is optional. If the return value of a callback command is optional and the user does not use the assignment form of the extended command, it discards the value of the return variable.

Refer to the specific callback command for the meaning of return values. See the example in Section 6.3.2.4 for an example use of return variables.

6.3.2.4. Parameters to a Callback Command

SYNTAX: <*parameterList*> is (<*fVariable*> or <*s32Variable*> or <*axisMask*>)1

EXAMPLE: “stuff” \$GLOB6 “more stuff” X Y Z (6+\$GLOB9)

Syntactically, many callback functions take a *parameterList*. An arbitrary number of axis masks, strings, floating point variables, or constants may be in the *parameterList* in any order, separated by whitespace. There is no limit to the number of objects in the *parameterList*, instead there is a limit to the total size of the *parameterList*. The user is allowed 320 bytes of data, where string constants and variables occupy 34 bytes apiece; axis masks, floating point constants, and strings occupy 12 bytes apiece. Host PC functions have many special considerations, foremost is what process must be running on the PC, in order to respond to the callback. The UNIDEX 600 MMI is primarily used for this purpose.

6.4. RS-447 Extended Commands

RS-447 extended commands may optionally be enclosed within parentheses "()". This provides compatibility with the RS-447 standard as well as previous command sets used by Aerotech's OS/2 based applications. Table 6-3 summarizes and references the extended commands within this chapter.

Table 6-3. Extended Command Summary

Extended Command	Page	Category	Description
+ - AND EQ > = != <=	Ch. 3	Operators	Mathematical Operators
:	6-44	Controller	Line label (see the GOTO/JUMP command)
ABS	Ch. 3	Function	Absolute value of a number
ACOS	Ch. 3	Function	Inverse cosine
AFCO	6-9	Controller	Auto Focus, move an axis based upon an analog input
\$AI	Ch. 3	Controller	Read the value of the specified analog input
ALIGN	6-10	Controller	Align a slave axis to its master axis
ASIN	Ch. 3	Function	Inverse sine
ATAN	Ch. 3	Function	Inverse tangent
#AXISNAMES	Ch. 4	Compiler	Redefine the standard axisnames
\$BI	Ch. 3	Controller	Read the state of a binary input
BIND	6-12	Controller	Allocate axis to task
\$BO	Ch. 3	Controller	Set the state of a binary output
CALL	6-13	Controller	Call a subroutine in same program, and return upon completion
CALLDLL	6-14	Controller	Allows a function to be called from within a dynamic link library
CAPTURE	6-14	Compiler	Gain ownership of an axis bound to another task
CFGMASTER	6-15	Controller	Configure a slave axis to track a master axis
CHANGECONFIG	6-15	Callback	Configure or Re-Configure an axis from within a CNC program
COMMINIT	6-17	Callback	Configure PC's serial port parameters
COMMSETTIMEOUT	6-18	Callback	Define PC's serial port read time out
CLS	6-13	Controller	Call a subroutine in same program, and return upon completion
COS	Ch. 3	Function	Trigonometric cosine
DATASTART	6-19	Callback	Data acquisition start
DATASTOP	6-25	Callback	Data acquisition stop
DBLTOSTR	6-125	Controller	Convert a (double precision) number to a string
#DEFINE	Ch. 4	Compiler	Define a string (or lines) to be substituted for a string
.DEFINED	Ch. 3	Function	Call argument existence testing
DFS	6-25	Controller	Define subroutine

Table 6-3. Extended Command Summary Cont'd.

Extended Command	Page	Category	Description
DISABLE	6-26	Controller	Disable an axis drive
DISPLAY	6-27	Callback	Display message within the custom display window
DVAR	6-27	Controller	Define program variable or array
ENABLE	6-29	Controller	Enable an axis drive
ENDM	6-30	Controller	Ends the motion on a single axis
EXE	6-30	Callback	Execute a DOS, Win NT/95 program and return completion code
EXECANNEDFUNCTION	6-32	Controller	Execute a canned function
EXEMODAL	6-32	Callback	Execute a DOS, WINDOWS NT/95 program, wait for completion and then return completion code
EXP	Ch. 3	Function	Raise e to a power
FARCALL	6-33	Controller	Call a subroutine in another program, and return upon completion
FARGOTO	6-34	Controller	Jump to another program
FARJUMP	6-34	Controller	See the FARGOTO command
FEDM	6-35	Controller	Async. infeed an axis
FILECLOSE	6-36	Callback	Closes the specified user data file
FILEEXISTS	6-36	Callback	Test for existence of a file
FILEOPEN	6-36	Callback	Opens the specified user data file for writing
FILEREAD	6-37	Callback	Read data from a file
FILEREADINI	6-39	Callback	Read .ini file
FILEWRITE	6-40	Callback	Writes the information into the specified data file
FILEWRITEINI	6-42	Callback	Write a single parameter or complete parameter .ini file to disk
FRAC	Ch. 3	Function	Fractional part of a number
FREE	6-43	Controller	De-allocates an axis from a task
FREECAMTABLE	6-43	Controller	Free memory allocated to a cam table
GOTO	6-44	Controller	Jump, or go, to the specified entry point
HAND	6-45	Controller	Allows positioning of an axis with a handwheel
HANDWHEEL	6-45	Controller	See the HAND command
HOME	6-46	Controller	Synchronously home an axis to its absolute reference point
HOMEASYNC	6-47	Controller	Asynchronously home an axis to its absolute reference point
IF-THEN-ELSE-ENDIF	6-47	Controller	Conditional execution operator
#INCLUDE	Ch. 4	Compiler	Include a file within another

Table 6-3. Extended Command Summary Cont'd.

Extended Command	Page	Category	Description
INDEX	6-50	Controller	Starts relative move while continuing with next program line
INT	Ch. 3	Function	Integer portion of a number
ISAVAIL	6-50	Function	Is the specified axis available for use by the current task
JUMP	6-44	Controller	See the GOTO command
LOADCAMTABLE	6-62	Callback	Load a cam table and configure the master and slave axes
#MAKENCODESLABELS	6-63	Compiler	Convert N codes to program labels, for M97 and M98 M codes
MAP	6-64	Controller	Assign a physical axis to a task with a given axis name
MASKTODOUBLE	6-64	Function	Convert a task axis mask to an integer value
MOVETO	6-65	Controller	Moves a specified axis to a specific position at a specified speed
MSET	6-66	Controller	Output the specified vector to a brushless motor
MSGBOX	6-67	Callback	Display a user message in a pop-up box
MSGCLEAR	6-70	Callback	Clear the messages from the user display list
MSGDISPLAY	6-71	Callback	Display a user message in the user display list
MSGHIDE	6-72	Callback	Hide the user message display list
MSGINPUT	6-72	Callback	Input data from the user via a pop-up box
MSGLAMP	6-74	Callback	Display status or warning info. in the message lamps
MSGMENU	6-75	Callback	Display a user option menu
MSGSHOW	6-76	Callback	Show the user message display list
MSGTASK	6-76	Callback	Clear Task fault messages from the MMI task fault display area
ON	6-77	Controller	Monitor and act upon a condition
ONGOSUB	6-79	Controller	Conditional branch on error conditions
OSC	6-88	Controller	Causes a specified axis to oscillate (cycle) a specified distance and speed
#PARMNAME\$		Compiler	Redefine the call stack parameter names
POPMODES	6-89	Controller	Restore the modal G Code states
PROBE	6-90	Controller	Initialize touch probe
PROGRAMDOWNLOADFILE	6-91	Callback	Download a CNC program to the controller
PROGRAMEXECUTE	6-92	Callback	Execute a CNC program on a specified task on the controller
PROGRAMEXECUTEFILE	6-92	Callback	Execute a CNC program from a file on a specified task on the controller
PROGRAMTASKRESET	6-93	Callback	Change the execution mode of a CNC program on another task
PROGRAMUNLOAD	6-94	Callback	Unload a CNC program from the controllers memory
PSOD	6-97	PSO Card	Firing distance entry
PSOF	6-100	PSO Card	Specify tracking axes and/or begin tracking

Table 6-3. Extended Command Summary Cont'd.

Extended Command	Page	Category	Description
PSOP	6-102	PSO Card	Laser pulse output definition
PSOS	6-105	PSO Card	Scaling of axes
PSOT	6-106	PSO Card	Digital and analog output control
PGM	6-33	Controller	See FARCALL
PRG	6-33	Controller	See FARCALL
PUSHMODES	6-89	Controller	Save the modal G Code states
REF	6-46	Controller	See HOME
RELEASE	6-112	Compiler	Release ownership of an axis to its bound task
RPT/ ENDRPT	6-112	Controller	Repeat blocks specified number of times, then end
RETURN	6-117	Controller	Return from subroutine
\$RI	Ch. 3	Controller	Read the state of a register input
\$RO	Ch. 3	Controller	Set the state of a register output
SETCANNEDFUNCTION	6-113	Controller	Define a canned function
SETPARM	6-119	Controller	Set a parameter within a CNC program
SIN	Ch. 3	Function	Trigonometric sine
SLEW	6-119	Controller	Allows the user to position the axis manually with a mouse/trackball or joystick
STRCHAR	6-123	Controller	Look for a set of characters within another string
STRCMP	6-122	Controller	Compare the length of two strings
STRFIND	6-122	Controller	Look for a string within another string
STRLEN	6-121	Controller	Find the length of a string
STRLWR	6-124	Controller	Convert a string to lower case characters
STRM	6-120	Controller	Begins motion on a single axis without stopping at any given target position
STRMID	6-125	Controller	Remove a string from a larger string
STRTOASCII	6-124	Controller	Find the ASCII value of a character
STRTODBL	6-123	Controller	Convert a string to a (double precision) number
STRUPR	6-124	Controller	Convert a string to upper case characters
SQRT	Ch. 3	Function	Square root
SYNC	6-126	Controller	Synchronizes a slave to a master axis
SUB	6-25	Controller	See the DFS command
TAN	Ch. 3	Function	Trigonometric tangent
TRACK	6-128	Controller	Establish master/slave relationship

Table 6-3. Extended Command Summary Cont'd.

Extended Command	Page	Category	Description
VOLCOMP	6-131	Callback	Calculate the compensation factor for the velocity of light
WAIT	6-132	Controller	Will hold position until a specified condition is met
WHILE / ENDWHILE	6-132	Controller	Execute blocks while condition is true

6.4.1. Auto Focus - AFCO

SYNTAX: `AFCO, AXIS, ADC_CHANNEL, SPEED, DEADBAND,`
`ANALOG_SETPOINT, ANTIDIVE`

EXAMPLE: `AFCO Z, 0, 60, .1, 0, 5`

Where:

- `AXIS` : axis to track the analog input
- `ADC_CHANNEL` : analog input channel (0 to 7)
- `SPEED` : Speed of `AXIS` when the difference between the current analog input and `ANALOG_SETPOINT` is 10 volts (IN/MIN or MM/MIN). `ANALOG_SETPOINT` is 10 volts
- `DEADBAND` : Minimum deviation from `ANALOG_SETPOINT` for motion to occur
- `ANALOG_SETPOINT` : Desired analog target value, range +/- 10 volts
- `ANTIDIVE` : Value of the analog input beyond which no motion will occur, range +/- 10 volts

This command is typically used to maintain a constant height above the part surface using a position transducer, which returns an analog voltage proportional to distance. A desired height above the surface is specified by the `ANALOG_SETPOINT` variable. The speed at which the axis will respond to a changing analog input is defined in the following pseudo code:

```

DIFFERENCE = CURRENT_ANALOG_INPUT - ANALOG_SETPOINT
IF ABS(DIFFERENCE) > DEADBAND THEN
    IF ANTIDIVE < 0 THEN
        IF CURRENT_ANALOG_INPUT > ANTI_DIVE THEN
            OUTPUT_SPEED = SPEED * DIFFERENCE/10
        ELSE
            OUTPUT_SPEED = 0
        ENDIF
    ELSE
        IF CURRENT_ANALOG_INPUT < ANTI_DIVE THEN
            OUTPUT_SPEED = SPEED * DIFFERENCE/10
        ELSE
            OUTPUT_SPEED = 0
        ENDIF
    ENDIF
ELSE
    OUTPUT_SPEED = 0
ENDIF
OUTPUT_SPEED_COUNTS = OUTPUT_SPEED * CNTSPERINCH

```

The *ANTIDIVE* parameter is used to specify a maximum analog value above which, no motion will occur. This parameter will prevent the axis from “diving” if the auto-focus head passes over a hole, or moves off of the edge of the material whose thickness it is sensing. This implies that the auto-focus head will have to initially be positioned to an analog input value, which does not exceed the value specified by *ANTIDIVE*, otherwise no motion will occur. Note that the direction the axis will move, is equivalent to the sign of the difference between the current analog input and the set-point value and the sign of the CntsPerInch machine parameter. To reverse the direction of movement, specify a negative speed command or invert the sign of the CntsPerInch machine parameter. Also, the *ANTIDIVE* parameter is unidirectional, as indicated by its sign. In other words, an *ANTIDIVE* parameter of 2 implies no motion when the analog input is greater than 2 volts, but there will be motion if the analog input is more negative than 2 volts.

The auto-focus command may be disabled by specifying the auto focus axis in the ENDM command.

6.4.2. ALIGN Command

SYNTAX: **ALIGN AXIS, MASTER_TARGET, SPEED**

AXIS specifies the axis to move into alignment relative to its master axis.

MASTER_POSITION specifies the master axis position to align to.

SPEED is the speed at which the alignment move is to occur at.

The master target is an absolute position, and is specified in user units. The speed is specified in user units/minute.

This command will generate a move that will be added to the current synchronized motion enabled via a previous TRACK command. For this command to function correctly the following items must be observed:

Both the master and slave axis must have their CntsPerInch machine parameter set correctly. These parameters are used to convert units (IN/MM) in the master coordinates into machine counts on the slave axis.

The current master position is referenced relative to the zero position of the slave axis. Note, that the zero (home) position of the slave axis can be changed through the HOMEOFFSET parameter or by setting the axis POS parameter directly.

The distance the axis will travel to align with the specified master position is defined by:

CURRENT_MASTR_POS - MASTR_TARGET - CURRENT_SLAVE_POS

Monitoring the POSTOGO axis parameter allows the completion of the move to be tested. When the POSTOGO axis parameter reaches zero, the ALIGN move is complete.

Refer to Figure 6-1 for clarification on how the ALIGN command functions.

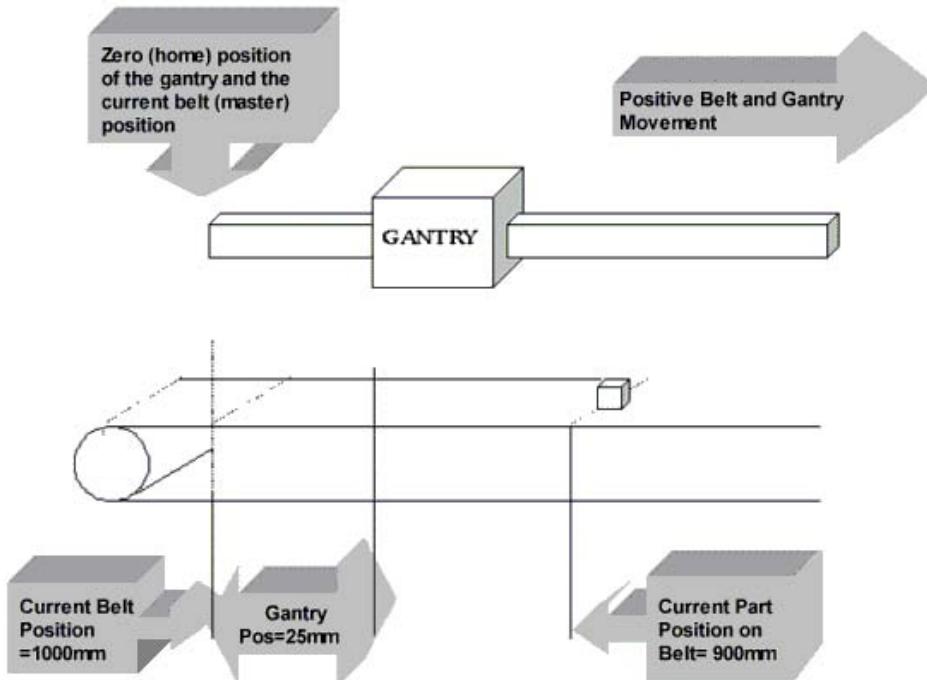


Figure 6-1. Align Command Function Illustration

In Figure 6-1, the current belt position is 1000mm, the gantry is displaced 25mm from its home position and the position to align to (the current part position on the belt) is 900mm. Note that this illustration represents the relative positions of all the elements when the ALIGN command is executed. In actuality, the gantry is tracking the belt as it moves from left to right. The execution of ALIGN GANTRY 900 1000 for the illustration above will cause a move of 75mm (MASTER_POS – GANTRY_POS – ALIGN_POS) at 1000 mm/min to be added to the tracking motion of the belt. When the move completes, the gantry will be located above the part on the belt.

6.4.3. BIND Axis Command

SYNTAX: **BIND <axisMask>**

EXAMPLE: **BIND X Y Z**



This command should not normally be used. Axes should be bound to a task within the axis configuration wizard and captured by another task and then released by that task for the original (or other) task to command motion on.

Since there are four independent tasks running in parallel, there must be some way of arbitrating situations where more than one task tries to move the same axis. The **BIND** command serves this purpose. Once an axis is bound to a task, no other task can move that axis, however, you can modify most parameters of an axis without having ownership of that axis. If a task tries to move an axis bound to another task, the “physical axis is controlled” fault appears (on the task attempting to move the axis). The BIND command is not typically necessary, if you are operating from the U600MMI-NT/95, which automatically binds axes within the “task-axis” group box of the second page of the Axis Configuration Wizard.

A **BIND** command automatically clears all *fixture offsets* on the axes being bound.

If a task tries to move an axis without binding it, then the “axis is not bound” fault occurs.

If a task is moving an axis and another task tries to bind that axis, then the “physical axis is controlled” fault appears.

The **FREE** command unbinds an axis from a task. If the user frees an axis that is not bound, the controller ignores the command.

EXAMPLE PROGRAM:

```
BIND X Y  
G1 X50 Y50 F100  
FREE X Y
```

6.4.4. Call Subroutine Command

CALL / CLS

SYNTAX: CALL <label> [[callArgument>]]

EXAMPLE: CALL MYROUTINE X78 Y\$GLOB8

The **CALL** command, along with the DFS and ENDDFS commands execute subroutines within a program. Please refer to the DFS command for details on subroutines and their behavior.

Please refer to Call Arguments (Section 3.11.7.) and Call Argument Existence Testing (Section 3.11.7.1.) for details on program/subroutine call arguments (sometimes called parameters). The number of nested subroutines is limited by the MaxCallStack task parameter.

When exiting the subroutine the modal settings are not restored. For example, if **G90** is active before the subroutine call, and the subroutine executes **G91**, then just after returning from the subroutine, **G91** will still be active!



EXAMPLE PROGRAM:

```
DVAR $theCubedValue
$GLOBO = 1
REPEAT 10          ; For values 1 to 10
    CALL doCube      ; Performs the cube
    $GLOBO[$GLOBO] = $theCubedValue ; Stores it in global vars 1 to 10
    $GLOBO = $GLOBO + 1 ; Go on to the next one
ENDRPT
M2                ; End of program

;;; NOTE without the M02 above, the
;;; doCube subroutine would be executed one more time.

DFS doCube
    $theCubedValue = $GLOBO**3
ENDDFS

$GLOBAL0 = $GLOBAL0 * $a
ELSE IF ($p.DEFINED == 1) THEN
    $GLOBAL0 = $GLOBAL0 * $p
ENDIF
ENDDFS
```

6.4.5. CallDLL Command

SYNTAX: **CALLDLL** “*UserDLL*”, “*UserFunction*” [,arbitrary data]

EXAMPLE: **CALLDLL** “AerCBack.dll”, “AerCBackFileWrite”, \$hFile,
“;comment/data”

This CNC statement allows a function to be called from within a dynamic link library (.DLL). The *User.DLL* and the *UserFunction* may be specified. The specified .DLL must be found within the system environment PATH variable or an absolute path must be specified. For detailed information on writing a .DLL that may be called by this command, see TN0004 in the online help file.

EXAMPLE PROGRAM:

```
DVAR $hFile

CALLDLL "AerCBack.dll", "AerCBackExe", "CMD.Exe /C AerDebug.Exe"
CALLDLL "AerCBack.dll", "AerCBackExe", "Notepad.Exe"
CALLDLL "AerCBack.dll", "AerCBackExe", "Notepad.Exe"
e:\u600\programs\test.pgm", 2

$hFile = CALLDLL "AerCBack.dll", "AerCBackFileOpen",
"e:\u600\programs\test.pgm", 2

DISPLAY 0, $hFile

IF $hFile != 0 THEN
    CALLDLL "AerCBack.dll", "AerCBackFileWrite", $hFile, ";Useless
comment"

    CALLDLL "AerCBack.dll", "AerCBackFileClose", $hFile
ENDIF
```

6.4.6. Capture Axis

SYNTAX: **CAPTURE** <axisMask >

EXAMPLE: **CAPTURE X Y**

This command allows a task to borrow ownership of an axis from another task that owns the axis (another task that has performed a BIND command on the axis). Suppose, for example that task 1 has bound an axis. Then as long as that axis is not moving at the time when another task issues a CAPTURE command on that axis, that task takes ownership of the axis, and can command motion on that axis as if it owned it. Once an axis is CAPTURED, the task that originally bound it (task 1 in the above example) cannot command motion on the axes until the other task releases it.

While an axis is captured, it behaves in every way as if it were bound to that task.

A CAPTURE command automatically clears all fixture offsets on the axes being bound.

6.4.7. CFGMASTER Command (Configure Master Axis)

SYNTAX: CFGMASTER <SlaveAxis>, <Type>, <MasterAxis>

Where:

<SlaveAxis> is the axis whose master axis is being configured.

<Type> = 0, Slave off of the master axe's commanded position.

= 1, Slave off the masters actual position.

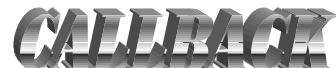
<MasterAxis> is the master axis.

This command configures a slave axis to track a master axis for camming motion. It is required prior to using the TRACK or GEARING commands. It is not required prior to the HANDWHEEL or LOADCAMTABLE command, which automatically configures the master axis. This command assumes that the master axis has already been configured for the appropriate type of feedback via the axis configuration wizard.

Gearing Example Program

```
; The following program will configure the X axis to follow the Y axis at
; 2 times its commanded speed.
;
CFGMASTER X 0 Y          ; X is slave, Y is master
GEARSLAVE.X = 2
GEARMASTER.X = 1
GEARMODE.X = 1
G0 Y100000               ; Starts master moving
;
; slave axis follows at 2 times its speed
```

6.4.8. Change Axis Configuration from within a CNC Program



SYNTAX: CHANGECONFIG <axis>, <Cfg_number>, <IniFileName>

EXAMPLE: CHANGECONFIG 3, 1, ENCODER_CFG_FILE

<axis> is the axis index of the axis to reconfigure.

<Cfg_number> is the configuration number to read from the ini file.

<IniFileName> is the name of the ini file to read the configuration from.

This command allows you to change the configuration of an axis from within a CNC program. The DRIVE will be disabled by this command before the axis is configured. The specified Ini file must be of the format used by the AxisCfg.Ini file, as shown below.

All Callback commands will set the ErrCode task parameter, if an error occurs during execution.



Sample configuration file:

```
[AxisConfig.1]
FBType=EncoderHall
FBType.EncoderHall.Channel=4
FBType.EncoderHall.Lines=100000
FBType.EncoderHall.HallLines=2000
FBType.EncoderHall.CommOffset=0
FBType.EncoderHall.CommChannel=3
FBType.EncoderHall.Bounded=1
IOType=D2A
IOType.D2A.Channel=3
Sp1Type=Encoder
Sp1Type.Encoder.Type=3
Sp1Type.Encoder.Channel=3
Sp1Type.Encoder.Lines=160000
Sp1Type.Encoder.VelHomeFlag=0
Sp2Type=NULL
Name=X
Task=0
TaskAxis=2

[AxisConfig.2]
FBType=EncoderHall
FBType.EncoderHall.Channel=5
FBType.EncoderHall.Lines=100000
FBType.EncoderHall.HallLines=2000
FBType.EncoderHall.CommOffset=0
FBType.EncoderHall.CommChannel=3
FBType.EncoderHall.Bounded=1
IOType=D2A
IOType.D2A.Channel=3
Sp1Type=Encoder
Sp1Type.Encoder.Type=3
Sp1Type.Encoder.Channel=3
Sp1Type.Encoder.Lines=160000
Sp1Type.Encoder.VelHomeFlag=0
Sp2Type=NULL
Name=X
Task=0
TaskAxis=2

[AxisConfig.3]
FBType=EncoderHall
FBType.EncoderHall.Channel=3
FBType.EncoderHall.Lines=100000
FBType.EncoderHall.HallLines=2000
FBType.EncoderHall.CommOffset=0
FBType.EncoderHall.CommChannel=3
FBType.EncoderHall.Bounded=1
IOType=D2A
IOType.D2A.Channel=3
Sp1Type=NULL
Sp2Type=NULL
Name=X
Task=0
TaskAxis=2
```

6.4.9. COMMINT



SYNTAX: COMMINT \$FileHandle, modeString

EXAMPLE: COMMINT \$FileHandle, "baud=9600 parity=N data=8 stop=1"

\$FileHandle is a valid comport handle returned by the FILEOPEN command.

modeString is used to define the configuration of the serial port. It may contain a format specifier.

The modeString parameter has the same format as the command-line arguments for the operating systems MODE command. For example, the following string specifies a baud rate of 9600, no parity, 8 data bits, and 1 stop bit:

"baud=9600 parity=N data=8 stop=1"

For further information on the mode command syntax, refer to your operating system documentation.

All Callback commands will set the ErrCode task parameter, if an error occurs during execution.



EXAMPLE PROGRAM:

```
DVAR $hFile
DVAR $var1
DVAR $var2

; get a communications handle to COM2
$hFile = FILEOPEN "COM2", 2

; Initialize comport
COMMINT $hFile, "baud=9600 parity=N data=8 stop=1"

; Define 1 second timeout for data at serial port
COMMSETTIMEOUT $hFile, -1, -1, 1000

FILEWRITE $hFile, "Hello !"
FILEWRITE $hFile, "How are you ?"

FILEREAD $hFile 0 $var1, $var2
MSGDISPLAY 1, $var1, " ", $var2

FILECLOSE $hFile
```



6.4.10. COMMSETTIMEOUT

SYNTAX: **COMMSETTIMEOUT** \$FileHandle, readIntervalTimeOut,
readTotalTimeoutMultiplier, readTotalTimeoutConstant

EXAMPLE: **COMMSETTIMEOUT** \$FileHandle, -1, -1, 1000

See the COMMINIT command for a complete example program.



All Callback commands will set the ErrCode task parameter, if an error occurs during execution.

The COMMSETTIMEOUT function defines the time-out parameter for all read operations on a specified communications port.

\$FileHandle is a variable containing a valid comport handle returned by the FILEOPEN command.

ReadIntervalTimeOut – Specifies the maximum time, in milliseconds, allowed to elapse between the arrival of two characters on the communications line. During a FILEREAD operation, the time period begins when the first character is received. If the interval between the arrival of any two characters exceeds this amount, the FILEREAD operation is completed and any buffered data is returned. A value of zero indicates that interval time-outs are not used.

A value of -1, combined with zero values for both the ReadTotalTimeoutConstant and ReadTotalTimeoutMultiplier parameters, specifies that the read operation is to return immediately with the characters that have already been received, even if no characters have been received.

ReadTotalTimeoutMultiplier – Specifies the multiplier, in milliseconds, used to calculate the total time-out period for read operations. For each read operation, this value is multiplied by the requested number of bytes to be read.

ReadTotalTimeoutConstant – Specifies the constant, in milliseconds, used to calculate the total time-out period for read operations. For each read operation, this value is added to the product of the ReadTotalTimeoutMultiplier member and the requested number of bytes.

A value of 0 for both the ReadTotalTimeoutMultiplier and ReadTotalTimeoutConstant members indicates that total time-outs are not used for read operations.

The most common combinations of timeouts are as follows:

; If the data is waiting at the serial port, then immediately read the data and return.

COMMSETTIMEOUT \$hFile, -1, 0, 0

; If the data is not already at the serial port, then each read will wait up to 1 sec (1000 msec) for data

COMMSETTIMEOUT \$hFile, -1, -1, 1000

6.4.11. Data Acquisition Start

DATASTART

SYNTAX: DATASTART handle nsamps rate [[<axisMask>]] [[datamask]] [[mode]] [[waitTime]]

Where:

<i>handle</i>	is a < <i>iExpression</i> >	the handle of a file to write to
<i>nsamps</i>	is a < <i>fExpression</i> >	the number of samples to collect
<i>rate</i>	is a < <i>fExpression</i> >	the rate at which to collect data
<i>axismask</i>	is a < <i>iExpression</i> >	that specifies that axis to collect the data for
<i>datamask</i>	is a < <i>iExpression</i> >	what data to write to the file (default=3)
<i>mode</i>	is a < <i>iExpression</i> >	how to acquire data (default=0)
<i>waitTime</i>	is a < <i>iExpression</i> >	how long to wait between samples (default=100 milliseconds)



All Callback commands will set the ErrCode task parameter, if an error occurs during execution.



General Overview

The UNIDEX 600 Series controller is equipped with a data acquisition feature used to capture real-time information relevant to motion, and I/O. The data is collected and stored on the controller; and at appropriate intervals of time, blocks of data are uploaded from the controller to the PC, and written to the specified file on the PC. This data may be read from the file and displayed using the AerPlot utility.

Some of the data collected is related to particular axes (i.e., position), while other data is related to the system as a whole (i.e., clock). Table 6-4 specifies what data can be collected. The datamask parameter allows the user to specify the items to collect. For items that are axis related, that data will be collected for all axes specified in the axisMask parameter.

The user must use the **DATASTOP** command to terminate the data acquisition. The **FILEOPEN** and **FILECLOSE** commands must be used in conjunction with the **DATASTART** and **DATASTOP** commands to open and close a file to write the acquired data. Data may be acquired into a single data file from multiple acquisitions by either starting another acquisition with the **DATASTART** command before closing the file, or by re-opening an existing data file with the **FILEOPEN** command in mode 2, for appending to the end of the existing data file.

Timing Performance

Data collection has no detrimental effect on the speed of the update rate of the servo loop (does not effect the motion). However, it does interfere with any controller responses to other requests, and may significantly slow down the MMI600 or other library calling applications. It takes up memory on the controller card; 24*nsamps bytes for each axis requested and 24*nsamps for the system data. Data is collected as part of the servo loop interrupt, which is either 4 or 1 kilo-hertz, limiting the worst case accuracy to +/- 1 servo loop interrupt (+/- .25 msec or +/- 1 msec.).

Obviously, the longer the waitTime between samples, the less time taken up on the controller at collection. Similarly, smaller numbers of points take up less space on the controller.

All possible data is always collected on the controller and delivered to the PC. So there is no saving of controller time or space by specifying smaller amounts of data. However, in the infinite collection modes, be careful requesting more data than needed. If the user requests too much data (too large a datamask parameter), or makes the nsamps parameter too small, the controller may overwrite the data before it is uploaded to the PC. There is no warning generated for this. If this happens, the data acquisition will continue, but an error code will be returned in the ErrCode task parameter when the DATASTOP statement is executed. Either, lower the amount of requested data; kill the other process on the PC, or request a larger queue size.

If the controller collects data too fast, it will overwrite the circular queue before the Data Collection thread can retrieve the samples and write them into the file. The controller detects this “overwrite” condition, but will continue to collect data. If an overwrite condition occurs during data collection, then the DATASTOP command generates the task fault: “Data Collection Overflow. When an overflow does occur, data will be lost. Each data line that is returned will be consistent (each line is correct by itself), however blocks of lines will be missing. To correct for overflow conditions, either lower the rate of collection, kill the other process’s on the PC, or otherwise speed up the PC. Requesting a larger queue size will, in some cases help, as it will delay the onset of the overflow condition. You can also use the waitTime parameter, to try to speed up the PC polling. However, the waitTime only indicates the frequency of generation of WM_TIMER messages posted to the PC thread, and the actual collections will occur only as fast as the PC processor can dispose of other tasks of the same priority.

Errors

Like all callback commands, the DATASTART command may return errors in the ErrCode task parameter. Overflow problems (see the paragraph below) are only reported when the DATASTOP command is executed. The programmer should test the ErrCode task parameter after each DATASTART and DATASTOP command to insure that it is zero.

In the queued modes; if the PC data collection thread created by the DATASTART command can not keep up with the controller, the controller will overwrite data items that have not been uploaded to the PC. The result is; data points will be periodically missing from the file. In order to ensure that the PC can keep up, the user must make the queue buffer, nsamps, large enough to insure that the PC uploads all the points before they are overwritten. This does not apply to non-queued collection modes.

Parameters

Data Acquisition handle Parameter

The first parameter (*handle*) to the DATASTART command must be a variable containing the file handle to write the ASCII (text based) data to. A previous FILEOPEN command provides this file handle.

Data Acquisition nsamps Parameter

The second parameter is the number of samples (*nsamps*). The interpretation of this parameter is dependent upon the mode parameter (see below).

Data Acquisition rate Parameter

The third parameter is the acquisition rate (*rate*). The interpretation of this parameter is dependent upon the mode parameter (see below).

Data Acquisition axismask Parameter and datamask Parameter

The fourth and fifth parameters are optional and specify what data to collect. The fourth parameter, “<*axisMask*>”, is an optional list of axes to collect their data. If this is not supplied, then it only collects system data items, regardless of whether the *datamask* specifies axis data items.

The fifth parameter, “*datamask*”, is a 32-bit integer mask. Each bit specifies a data item in Table 6-4 available for collection. Any axis data items specified in the *datamask* will always be collected for all the axes specified in the *axisMask*. If the *datamask* is not supplied, it defaults to 3 (CLOCK and POSITION only).

DATA_CLOCK
DATA_POSITION
DATA_POSITION_CMD
DATA_POSITION_RAW
DATA_POSITION_MASTER
DATA_VELOCITY
DATA_VELOCITY_CMD
DATA_ACCELERATION
DATA_TORQUE
DATA_IN
DATA_OUT
DATA_ANL1
DATA_ANL2
DATA_ANL3
DATA_ANL4
DATA_ANL5
DATA_ANL6
DATA_ANL7
DATA_ANL8



The analog values are floating point, while the rest are integers.

Position is sampled from the primary (position) feedback device, except when in the latch based collection modes. In which case, the position is latched via the high speed position latch input. The analog values are floating point (6 digits beyond the decimal point), while the rest are integers. Velocity is not from the feedback device, rather it is the derivative of the position feedback, computed every millisecond. Similarly, acceleration is the derivative of that velocity, also computed every millisecond.

Data Acquisition mode Parameter

The sixth parameter, “*mode*”, is the optional data acquisition mode. If not provided, it defaults to zero. It determines the triggering of the data, and the interpretation of the *nsamps* and *rate* parameters.

Mode 0 = Finite size rate-based collection. The *rate* parameter specifies the interval between acquisitions in milliseconds. Collects only *nsamps* samples, then data acquisition stops. However, the user still needs to do a **DATASTOP** afterwards.

Mode 1 = Infinite size rate-based collection. The *rate* parameter specifies the interval between acquisitions in milliseconds. Collects data continuously until performing a **DATASTOP**. The queue size is set at *nsamps* samples.

Mode 2 = Finite size latch-based collection. Like mode 0, but collects only when an edge is detected in the Position Latch Input (see *U600 Hardware Manual, EDU154*, Technical Details, under the P10 pinout). The *rate* parameter is not used in this mode. The position latch input is checked every millisecond. Therefore, the maximum possible rate of data collection is one sample per millisecond. Collects only *nsamps* samples, then data acquisition stops. However, the user still needs to do a **DATASTOP** afterwards.

Mode 3 = Infinite size latch-based collection. Same as Mode 2, but uses a queue like mode 1. The queue size is set at *nsamps* samples.



In queue mode, if the PC data collection thread created by the **DATASTART** command cannot keep up with the controller, the controller overwrites data items not yet uploaded by the PC. The result is data points will be periodically “skipped” in the file. There is no error or warning delivered in this case. In order to ensure that the PC can keep up, the user must make the queue buffer *nsamps* large enough so the PC uploads all the points before they are overwritten. This does not apply to non-queue-mode collections.

Data Acquisition waitTime Parameter

The optional seventh parameter <waitTime>, is the time between data acquisition samples. If not provided, it defaults to 100 millisecond.

Table 6-4. Data Available for Collection

Bit	Hexadecimal Value	Data	Type	Data type	Label	Description / Elaboration
0	0x1	Clock	Sys	32 bit integer	CLOCK	Milliseconds
1	0x2	Position	Axis	32 bit integer	POS	Counts
2	0x4	Position Command	Axis	32 bit integer	POSCMD	Counts
3	0x8	Raw Position	Axis	32 bit integer	POSRAW	Counts
4	0x10	Master Position	Axis	32 bit integer	POSMAS	Counts
5	0x20	Velocity	Axis	16 bit integer	VEL	Counts/millisecond
6	0x40	Velocity Command	Axis	16 bit integer	VELCMD	Counts/millisecond
7	0x80	Acceleration	Axis	16 bit integer	ACC	Counts/millisec/millisec
8	0x100	Torque	Axis	16 bit integer	TORQUE	-32768 to 32768
9	0x200	Bit Inputs	Sys	16 bit integer	BYINP	Bit Mask (virtual inputs 0-15)
10	0x400	Bit Outputs	Sys	16 bit integer	BYOUT	Bit Mask (virtual outputs 0-15)
11-18	0x800 - 0x40000	Analog inputs 1 through 8	Sys	32 bit Floating Point	ANL1 - ANL8	Volts (-10 to 10) 6 digits past decimal point are always shown
19-31	0x80000-0x80000000	NOT USED				

The analog values are floating point, while the rest are integers.



Format of Output

The file written is always an ASCII text file. Each line is terminated with a carriage return and linefeed (ASCII codes 13 and 10, respectively). Data values on each line are always separated by a single space.

Lines beginning with a semicolon ";" as the first character are not data lines; these are comment lines written to the file in order to identify the file and/or errors in the collection process. These lines might appear anywhere in the file, not just in the beginning. However, the first three lines in the file are always comment lines and help identify the file content.

The first line of the file always provides the maximum width of any line in the file, followed by (separated with a space) the number of data values per data line. The maximum width is useful for allocating text for the data lines, since lines can potentially be over fifteen thousand characters long (16 axes, 8 data items per axis, plus 11 system data items).

The second line of the file always is a reflection of the **DATASTART** command that created it, with all the parameters resolved into constants (except the return variable assignment, if any, is not shown, and the handle is not shown). Even if the user did not

supply certain parameters, and allowed them to default, all parameter values are shown in the first line of the file.

DATASTART nsamps rate datamask <axisMask> mode waitTime

The third line of the file identifies the units of time.

The fourth line of the file is always a title line, identifying the contents of a line of data. It essentially serves to identify the columns of data in the file, although no effort is made to line these labels up with the actual data. The line is a series of five character labels, each label separated by a space (see Table 6-4). The axis related labels are the three characters of their label, followed by a two digit axis number. For example, the axis 2 position is labeled POS02, while the clock time is labeled CLOCK.

Normally, the data collection terminates with the DATASTOP command. However, if not in a queued mode, data collection can terminate due to the controller completing the requested number of samples. In this case, terminating the file, is the line “; POINTS COLLECTED”.

EXAMPLE PROGRAM:

```
; This program is the ExData.Pgm found in the \U600\Programs directory.  
;  
; This example will produce the file: \U600\PROGRAMS\test.dat which file will look like:  
(except for the first preceding semicolons)  
;; 80 5  
;; DATASTART 500 10 3 35 0 100  
;; CLOCK POS01 VEL01 POS02 VEL02  
;58586159 0 0 0 0  
;58586169 0 0 0 0  
;58586179 0 0 0 0  
;58586189 0 0 0 0  
; ...  
;  
DVAR $filenum  
  
$filenum = FILEOPEN "\U600\PROGRAMS\test.dat" 0 ; open the file to write to  
IF ErrCode NE 1 GOTO fail1 ; Goto fail routine if the command was unsuccessful  
DATASTART $filenum 500 10 X Y 35 ; Acquire 500 samples of clock, position, veloc  
; data for X, Y axes, one sample every 10 msec  
IF ErrCode NE 1 GOTO fail2 ; Goto fail routine if the command was unsuccessful  
; <program block>  
Any number of program blocks  
;...  
Typically, these block do motion  
; <program block>  
  
DATASTOP ; Stop acquiring data  
  
:fail1  
FILECLOSE $filenum ; close the data file  
:fail2 ; error handler  
M02
```

6.4.12. Data Acquisition Stop

DATASTOP

SYNTAX: **DATASTOP**

The **DATASTOP** command is the opposite of a **DATASTART** command. See the **DATASTART** example. The **DATASTOP** command can generate the task fault: “Data Collection Overflow”. See the **DATASTART** command under “Timing Performance” for details on this error.



All Callback commands will set the ErrCode task parameter, if an error occurs during execution.



6.4.13. Define Subroutine

SYNTAX: **DFS <label >**
 <NCBlock >
 ...
 <NCBlock >
ENDDFS

The **DFS** command, along with the **ENDDFS** and **CALL** extended commands define and call subroutines. A subroutine is a group of CNC program blocks. Typically, this group of blocks perform a particular task multiple times during the execution of a parts program. A call can be made to the appropriate subroutine (using the **CALL** extended command), at each point in the program requiring the performance of this task. The subroutine executes and upon completion, program flow returns to the point where the subroutine was called. Call arguments may be passed to subroutines.

The advantage to using a subroutine is, it can decrease the parts program size. Multiple copies of program blocks that perform the same task can be replaced with one subroutine call. However, CNC subroutines unlike “C” language subroutines, do not have local variables. CNC subroutines have the same local program variable values that the calling program has.

Unless the user is careful, execution may fall into the subroutine without a call. If the compiler detects this condition, it delivers a warning. See the example below.



When exiting the subroutine the modal settings are not restored. For example, if G90 is active before the subroutine call, and the subroutine executes G91, then just after returning from the subroutine, G91 will still be active ! The **PUSHMODES** command may be used to store the states of the modal G codes.



Subroutines may call themselves or other subroutines, the MaxCallStack task parameter limits the number of nested calls that you may have. However, there is no limit to the MaxCallStack parameter, therefore there is no limit to the number of nested calls that can be made, except, that imposed by the amount of available memory on the controller.

EXAMPLE PROGRAM:

```
DVAR $theCubedValue
$GLOB0 = 1
REPEAT 10
    CALL doCube
    ; For values 1 to 10
    ; Performs the cube
    $GLOB0[$GLOB0] = $theCubedValue ; Stores it in global vars 1 to 10
    $GLOB0 = $GLOB0 + 1           ; Go on to the next one
ENDRPT
M2                                     ; End of program

;;; NOTE without the M02 above, the
;;; doCube subroutine would be executed one more time.

DFS doCube
    $theCubedValue = $GLOB0**3
ENDDFS
```

6.4.14. DISABLE Axes Command**SYNTAX:** **DISABLE** <AxisMask >**EXAMPLE:** **DISABLE X Y** ; Disable X axis and Y axis drives

This command sets the DRIVE axis parameter to 0, for all the axes in the specified AxisMask.

6.4.15. Displaying Text in the CDW Window

DISPLAY



The **DISPLAY** command is provided only for backward compatibility with previous versions of the UNIDEX 600MMI-NT/95. See the **MSGxxx** commands.

All Callback commands will set the ErrCode task parameter, if an error occurs during execution.



6.4.16. Define Program Variable or Array

DVAR

SYNTAX: **DVAR** (\$<pvName>) ; defines a program variable
DVAR (\$<pvName>[<integer>]) ; defines a program array of variables
<pvName> is \$<letter><letter>(<letter> or <underscore>)

The **DVAR** extended command defines a program variable name for use within a program. Like all other variables and parameters, a dollar sign must precede a program variable name. Program variable names must be at least three characters long and the first two characters must be letters. Program variables are the only CNC keywords where the difference between upper and lower case is relevant. For example, the variable “\$MYVARIABLE” is a distinctly different program variable than the program variable “\$myvariable.” Also, program variables are the only keywords that allow mixed case (i.e., \$MyVariable). One or more variables can be defined on the same **DVAR** line.

All **DVAR** statements in a given program must appear before any executable command in the program (non-executable commands include **DVAR**, and any compiler directive).

Also, the **DVAR** statement can define program variables or program variable arrays. Program variable arrays are simply lists of consecutive program variables, accessed by an index value.

All program variables are automatically initialized to zero upon program initialization. The maximum number of variables/array elements, is limited only by the amount of available memory on the controller.

Program variables are only accessible within the current program. Program variables defined in one program are not accessible by other programs running on the same Task (see Task variables). Also, program variables are not accessible by programs running under other Tasks (see Global variables). The UNIDEX 600 Series controller does not support variables local to subroutines.

6.4.16.1. Define Program Variable

SYNTAX: **DVAR (\$<pvName>) ;** Defines a program variable

EXAMPLE: **DVAR \$myVariable, \$andherVariable**

A variable is a location used by a program to hold a numeric floating point value. The value of this variable may be modified from within the CNC program and used in subsequent operations, such as numerical calculations. Another use for variables is directing program flow (refer to the **IF** command). The maximum number of variables, is limited only by the amount of available memory on the controller. Refer to *Expressions* for more details on floating point variable syntax.



This command may not be used in the Program Queue Mode.



All **DVAR** statements in a given program must appear before any executable command in the program (non-executable commands include **DVAR**, and any compiler directive).

6.4.16.2. Define Program Array

SYNTAX: **DVAR (\$<pvName>[~<integer>~]) ;** Defines a program array

EXAMPLE: **DVAR \$myArray[10]**

The **DVAR** command can also define an array. An array is a group of related variables. Each element of the array is one program variable.

In the **DVAR** command the user must specify the number of variables found in the array. To access individual variables, the user can specify the array name followed by a number stating the referenced variable.



This command may not be used in the Program Queue Mode.



All **DVAR** statements in a given program must appear before any executable command in the program (non-executable commands include **DVAR**, and any compiler directive).

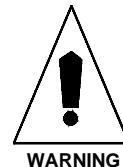
EXAMPLE PROGRAM:

```
DVAR $myArray[3]      ; Define an array which
; contains three variables.
; These variables can be accessed as follows:
;   myArray[0]
;   myArray[1]
;   myArray[2]
```

Also, a variable may hold the value of an array index (i.e., \$myArray[\$myVar]). The programmer can continue this kind of nesting indefinitely, as in \$myArray[\$anotherArray[\$thisArray[\$anotherOne[5]]]]. However, it is the programmer's responsibility to ensure that every index evaluates to an integer. If an index (a value within brackets) is not an integer (i.e., \$myArray[6.5]), the controller generates a task fault when it executes that command.

Any variable can be indexed, even variables not declared in a **DVAR** command and the index is used as an offset from the actual variable storage location. For example, \$GLOBAL[6] and \$GLOBAL1[5] refer to the global variable “\$GLOBAL6”.

A test of the index range validity is not performed. If the index is outside the range defined for that array (as defined in the **DVAR** command), the syntax specifies other program variables, based on the order of declaration and this may lead to unexpected results. See the example below.



WARNING

EXAMPLE PROGRAM:

```
DVAR $vara
DVAR $varb[5]
DVAR $varc)
$vara[0] = 1      ; Same as assignment to $vara
$vara[1] = 2      ; Actually assigns to $varb[0]
$varb[5] = 7      ; Actually assigns to $varc
$vara[10]= 11     ; WARNING: non-existent variable - results unexpected !
$varc[1] = 9      ; WARNING: non-existent variable - results unexpected !
```

6.4.17. ENABLE Command

SYNTAX: **ENABLE** <AxisMask>

EXAMPLE: ENABLE X Y ; Enable X axis and Y axis drives

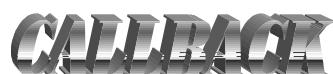
This command sets the DRIVE axis parameter to 1, for all the axes in the specified AxisMask.

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.

6.4.18. End motion (Asynchronous)**ENDM****SYNTAX:** **ENDM <axisLetter>****EXAMPLE:** **ENDM Y** ;End the Y axis motion

The **ENDM** asynchronous motion command halts motion on a single axis. If there is no motion, it has no effect. The ENDM command stops motion on the axis regardless of how the motion was initiated: asynchronous (STRM, INDEX, AFCO, jogging, etc.), synchronous motion (**G0**, **G1**, **G2 / G3**) or slave motion (LOADCAMTABLE). Except in the case of slave motion, the ENDM command will decelerate the axis smoothly to a stop using the same deceleration axis parameters as a **G0** command (DECEL, DECELMODE, and DECELRATE). However, in the case of slave motion, the axes will be stopped abruptly, therefore it is recommended that slave axes be stopped by an ENDM command on the master axis instead of the slave axis.

Although it is classified an asynchronous motion command, the ENDM command is actually synchronous, meaning that the task waits until the axis is “done” before proceeding to the next CNC program line.

**6.4.19. Execute DOS or Windows Program****EXE****SYNTAX:** **EXE <s32Expression> <parameterList>**

The *<s32Expression>* is the filename of the executable

The *<parameterList>* is the data for the executable



All Callback commands will set the ErrCode task parameter, if an error occurs during execution.

This command allows the operator to execute any .EXE file, .COM file, .CMD or .BAT file as a CNC command. The first parameter, a string, specifies the name of the executable. The remaining parameters on the line pass to the executable file when the CNC statement executes. The executable program’s return code sets the ErrCode task variable, as shown in the following example.

An arbitrary number of string or floating point variables, format specifiers, or constants may be included as the executable’s parameters. There is no limit to the number of the executable’s parameters, instead there is a limit to the total size of the parameters. The user is allowed 320 bytes of data, where string constants and variables occupy 12 bytes apiece, and floating point constants and strings occupy 34 bytes apiece.

Also, the operator can run .BAT (or .CMD) files, by specifying the executable name as “COMMAND.COM” for Windows 95 or “CMD.EXE” for Windows NT, and passing the

.BAT or .CMD file name as a parameter with the “/c” or “/ k” switches. Refer to your Windows operating system documentation for more details.

When passing parameters you MUST append a space to the end of the executables name, as shown in the example program.



EXAMPLE PROGRAM:

```

; Use of EXE without parameters
;

EXE "erase c:\u600\*.tmp"      ;Erases all .tmp files in the u600 directory on
;drive C.
IF ErrCode EQ 1 GOTO fail    ;Test return code

EXE "Cleanup.exe " $MYVAR "/a" ;Runs Cleanup.exe with $MYVAR and /a as
;parameters. For example, if $MYVAR was
;currently 10, the call would be:
;Cleanup.exe 10.00000 /a.

EXE "Cmd.exe /C typeit.bat"   ;Runs the batch file typeit.bat, with no
;parameters

M02                           ;end of program
:fail                          ;error handler routine
M02                          ;Handle error here or end program

; Use of EXEMODAL with parameters.
;

; NOTE: The space below within the string containing "Copye      ", this is
; required, when parameters such as $STRGLOB0 is appended to the string !
;

EXEMODAL "Cleanup.exe " $MYVAR "/a" ; Runs Cleanup.exe with $MYVAR and /a as
; parameters. For example, if $MYVAR was
; currently 10, the call would be:
; Cleanup.exe 10.00000 /a.

EXE "Cmd.exe /C typeit.bat"     ; Runs the batch file typeit.bat, with no
; parameters

M02                           ; end of program
:fail                          ; error handler routine
M02                          ; Handle error here or end program
end program

```

The executable file must exist somewhere within the system search path, unless a full path is specified.



6.4.20. EXECCANNEDFUNCTION Command

SYNTAX: EXECCANNEDFUNCTION <id>, <task>, [<callArgument>]

The EXECCANNEDFUNCTION command is used to execute a Canned Function. It allows parameters to be passed to the function (subroutine). CallArguments may be passed to the Canned Function called as a subroutine.

Where:

<id> - is the number of the canned function.

<task> - is the task number to execute the canned function on (0 - 3).

<callArgument>- allow parameters to be passed to the function.

Alternatively, a canned function not requiring parameters may be called by setting the CannedFunctionID task parameter, i.e.,

CannedFunctionID[<.Task#>] = id



6.4.21. Execute DOS or Windows Program and Wait for Completion

SYNTAX: EXEMODAL <s32Expression> <parameterList>

The <s32Expression> is the filename of the executable

The <parameterList> is the data for the .exe/.bat program

EXAMPLE: See the example program in 6.4.19.



All Callback commands will set the ErrCode task parameter, if an error occurs during execution.

This command allows the operator to execute any .EXE file, .COM file, .CMD or .BAT file as a CNC command. The first parameter, a string, specifies the name of the executable. The remaining parameters on the line pass to the executable file when the CNC statement executes. The CNC program waits until the executable program finishes before continuing to the next step in the CNC program. The executable program's return code sets the ErrCode task variable, as shown in the following example.

An arbitrary number of string or floating point variables or constants may be included as the executable's parameters. There is no limit to the number of the executable's parameters, instead there is a limit to the total size of the parameters. The user is allowed 320 bytes of data, where string constants and variables occupy 12 bytes apiece, and floating point constants and strings occupy 34 bytes apiece.

Also, the operator can run .BAT (or .CMD) files, by specifying the executable name as "COMMAND.COM" for Windows 95 or "CMD.EXE" for Windows NT, and passing the .BAT or .CMD file name as a parameter with the "/c" or "/ k" switches. Refer to your Windows operating system documentation for more details.

6.4.22. FARCALL

FARCALL / PGM / PRG

SYNTAX: **FARCALL** <filename> [[<label>]] [[callArgument>]]

EXAMPLES: **FARCALL** “prog2.pgm” “start”

FARCALL “prog2.pgm”

FARCALL “” “starthere”

FARCALL “prog2.pgm” “start” X(\$GLOBAL0) P5.5

This command calls a subroutine within another CNC program. When the specified subroutine completes its task, program execution continues with the next program block in the calling program. After the last statement in the called program executes, or if a return executes from the called program, program flow returns to the point from where the subroutine was called. An M2 within a called program will terminate execution and control will not be returned to the main (calling) CNC program.

Although the subroutine called can be in the same program as it is called from, the CALL statement is probably more convenient in this case. The FARCALL is intended for the case where the called subroutine is in a different program.

Programs may call themselves or other programs, there is no limit to the nesting of calls, except for the limits imposed by available memory and by the MaxCallStack task parameter

The programmer should note that once a **FARCALL** is made to another program, the program variables of the original program are no longer available. Each program has its own set of program variables, even if a program calls another version of itself. Call arguments or global variables may be used for passing data to called CNC program.



When exiting the subroutine the modal settings are not restored. For example, if **G90** is active before the subroutine call, and the subroutine executes **G91**, then just after returning from the subroutine, **G91** will still be active! The PUSHMODES command may be used to store the states of the modal G codes.



The program variables declared within a particular file will be initialized to zero on the first FARCALL into any subroutine within that CNC program. On subsequent calls into the subroutines within that file, the program variables values will be retained from the last FARCALL. You must re-download the file, if you would like the value of the variables re-initialized to zero on subsequent FARCALL's.



The filename and label arguments define the point to jump to. If the filename is specified as “”, then it is assumed that the label is in the same program that the FARCALL is in.

The label is optional, if it is omitted, then execution will begin at the first line in the specified program.

CallArguments may be passed to the CNC program called as a subroutine.

EXAMPLE PROGRAM:

```
;Main.Pgm
FARCALL "PROG2.PGM" "entry3" X1
; This example uses a string variable as the name of a program
;
$STRGLOB1 = "prog2" ".pgm"
FARCALL $STRGLOB1 "entry3"

;Prog2.Pgm
$GLOB0 = $X           ; This line is not executed
:entry3
$GLOB0 = $X           ; This line is executed
```

6.4.23. Jump to program

FARGOTO / FARJUMP

SYNTAX: **FARGOTO** <filename> <label>

Where:

<filename> is the CNC program

:<label> ; The user may equivalently use: <label>:

EXAMPLE: FARGOTO "Prog1.Pgm"

The **FARGOTO** command, along with the colon symbol, jumps or transfers execution to another location in another program. Instead of proceeding with the next sequential program block, execution immediately proceeds to the command immediately following the label command with the colon in the named program. The colon can appear before or after the label name.

Once the user jumps to another program, they will not return to the originating program, unless another **FARJUMP** referencing the original program executes. In most cases the programmer should use the **FARCALL** statement instead, it returns to the original program once the called program completes. The exception is, if an ONGOSUB command defined in the original program occurs, causing a jump back to the original program.

Also, the user can use the **FARGOTO** to transfer control to a label within the same program, but we recommend the GOTO statement for this.



Once the programmer **FARJUMPs** to another program, the program variables of the original program are no longer available.

There is no limit to the number of **FARGOTOS** that may reference the same label, but a label can only be defined (using the colon) once in a program. We strongly recommend that the user not jump into block structures, since this can make code very confusing. For

example, if a **FARGOTO** statement jumps into a DFS defined subroutine, then execution will not return (as in a normal DFS call) back to the point of the GOTO at the ENDDFS command. The ENDDFS command will simply be ignored and execution falls through to the statement immediately following the ENDDFS.

FARGOTO has a special role as a return from an ONGOSUB command. Refer to the ONGOSUB command for details.

6.4.24. FEDM Command

SYNTAX: **FEDM** <axisLetter> <distance><speed>

Where:

<axisLetter> is the axis to feed in on (the slave)

<distance> is the amount to feed in, relative to the current position in user units

<speed> is in user units per minute or RPM for a rotary axis.

This statement executes motion on a single slave axis, stopping after the specified increment. The motion is asynchronous to the CNC program execution, meaning motion begins and then the next CNC statement begins execution. The **FEDM** command exhibits asynchronous accel/decel like a **G0** command. This command is intended to be used during electronic camming (it is applied to the motion in addition to a master/slave driving the given axis).

EXAMPLE:

FEDM Y 4.5 6.7 ;Infeed the slave to 4.5 inches, at 6.7 inches per second

6.4.25. File and Serial Port Command Overview



The following commands are available for accessing files and/or the PC's serial ports:

FILECLOSE	Closes the file or serial port that the user opened.
FILEEXISTS	Test existence of a file.
FILEOPEN	Open a file or serial port.
FILEREAD	Read data from a file or serial port.
FILEREADINI	Read data from a formatted .Ini file.
FILEWRITE	Write data to a file or serial port.
FILEWRITEINI	Read data from a formatted .Ini file
COMMSETTIMEOUT	Define the time-out on the serial port.
COMMINIT	Initialize the communications parameters on the specified serial port.

All of these (Callback) commands will set the ErrCode task parameter, if an error occurs during execution.



EXAMPLES: See File Example Program 1 on page 6-41, or File Example Program 2 on page 6-42, or see the COMMINIT command for a serial port example program.

**6.4.25.1. File Close Command****FILECLOSE****SYNTAX:** **FILECLOSE** <*fVariable*>

The <*fVariable*> parameter is the file handle returned by **FILEOPEN**.

EXAMPLE: See **COMMINIT** for a serial port example program.

See the **FILEWRITE** command for a file example program.

The **FILECLOSE** command closes the file or serial port that the user opened. The <*fVariable*> parameter is the fileHandle assigned to the file or serial port that the user would like to close. The ErrCode task parameter is set to a non-zero following the unsuccessful completion of this command.

**6.4.25.2. File Existence Testing Command****SYNTAX:** <*fVariable*> = **FILEEXISTS** <*s32Expression*>

EXAMPLE: \$bExists = **FILEEXISTS** “C:\U600\Programs\Prog.Dat”

See the **FILEWRITE** command for a file example program.

The **FILEEXISTS** command will return TRUE if the specified file exists, FALSE otherwise.

**6.4.25.3. File Open Command****FILEOPEN****SYNTAX:** <*fVariable*> = **FILEOPEN** <*s32Expression*> [[<*iExpression*>]]

EXAMPLE: \$hFile = **FILEOPEN** “COM2”, 2 ;open serial port 2
\$hFile = **FILEOPEN** “\U600\Data\MyFile.dat”, 2 ;open MyFile

See **COMMINIT** for a serial port example program.

See the **FILEWRITE** command for a file example program.

The <*s32Expression*> is the name of the file or serial port to be opened. The *s32Expression* is limited as defined by *filenames*.

The <*iExpression*> is the mode (0 by default); specify 2 for serial ports.

The <*fVariable*> is the file handle of the file.

The **FILEOPEN** command opens a serial port or a user data file. The <*s32Expression*> is the name of the file or serial port to open. The filename specified may be any valid Windows NT/95 path\filename (31 chars max.) and may use either an absolute or relative path specification. If no path is specified, the file is assumed to be in the current program

directory (\U600\Programs by default). To open a serial port, specify a communications port specifier, “COM1”, “COM2”, “COM3”, etc.

The *<iExpression>* is the mode, and refers to the mode of file access. Mode = 0 opens a new file overwriting an existing file. Mode = 1 opens an existing file for reading. Mode = 2 opens an existing file only for appending records to the end of the file. Serial ports should always be opened in mode 2.

The *<fVariable>* is the handle to the file assigned when it is opened; this allows referencing of a particular file when multiple files are open. This variable is required, and must be used in any subsequent FILEREAD, FILEWRITE or FILECLOSE commands. The ErrCode task parameter is set non-zero to indicate error conditions.

Refer to the FILEWRITE command for an example program.

6.4.25.4. FILEREAD Command

FILEREAD

SYNTAX: **FILEREAD** *<fVariable>**<integer>**<parameterList>*

EXAMPLE: See **COMMINIT** for a serial port example program.

See the **FILEWRITE** command for a file example program.

Where: *<fVariable>* is the file handle returned by the **FILEOPEN** command.

<mode> is a mask indicating the file read mode.

<parameterList> is the data to write to the file; this must be a series of variables, each variable separated by white space.

FILEREAD will read data from the serial port or an ASCII file, into CNC variables on the controller.

The **FILEOPEN** command must be used prior to the **FILEREAD** command, to provide a file handle for the data file or serial port.

FILE FORMAT

The FILEREAD command reads one line of data at a time, where a line is terminated by a non-printable ASCII character (a character which is not a tab, and whose ASCII code is less than 32). The next data line begins at the next printable ASCII character. Therefore, <CR>, or <CR><LF> etc. are all valid line terminators.

The FILEREAD command will identify items on each line, where items are separated by whitespace.

Each item must be a value or a string (surrounded by double quotes). Values may be integer or floating point. If an integer value is preceded by “0x” then it is interpreted to be a hexadecimal number, otherwise, it is assumed to be a decimal number.



WHITESPACE CHARACTERS

A whitespace characters is a series of spaces, tabs or commas, except if in “European mode” of the FILEREAD command, where a comma is not whitespace. In European mode, the comma is interpreted as a decimal point, and thus cannot be used as a whitespace separator. The mode parameter allows the whitespace character definition to be defined.

VARIABLE ASSIGNMENT

The variables to contain the data, must follow the mode parameter on the command line.

Each item on the line read from the data file, is sequentially assigned to the variables on the FILEREAD command line (first data item is assigned to the first item read, etc.) unless, the “array-mode” is active (See the mode parameter). The variables receiving the data may be program, task or global. However, the type of variable must match the data read (string or numeric) must match the type of the variable or an error will be generated. For example, the line: 0.987, “dog” could be read by: “FILEREAD \$fil 0 \$GLOB0 \$STRGLOBO”, but not by: “FILEREAD \$fil 0 \$STRGLOBO \$GLOB0”.

RETURN VALUE

The **FILEREAD** command will return the number of data items read. If the end of the file (EOF) has been reached, it will return 0.

MODE PARAMETER

This parameter is a bit mask, where each bit activates a specific option (see the Table below).

Table 6-5. Mode Parameter Values

Bit #	Value	Meaning
0	0x1	Array mode ON
1	0x2	European mode ON

In the array mode, you provide a single variable on the **FILEREAD** command line, which is an array element. The controller will read all values on the line into the array, beginning at the given variable. Caution must be used when in array mode that the variable specified is in fact an array element, and that the number of values in the file line is never greater than the size of the array. Violation of this rule will cause unpredictable results.

The European mode defines the whitespace characters.

The data on the line read is assigned to the variables specified by the user as arguments to the **FILEREAD** command.

6.4.25.5. FILEREADINI Command



SYNTAX: **FILEREADINI** <fileName>, sectionName, valueName, defaultValue

```
FILEREADINI    <fExpression>,      sectionName,      valueName,
                defaultValue
```

EXAMPLES:

```
$value = FILEREADINI "c:\u600\ini\test.ini", "TestMe", "Test1", -1
```

Or to read from the axis parameter file

```
$value = FILEREADINI 2, "AxisParm.1", "KP", 10
```

where;

- <fileName> is the name of the .INI file
- <fExpression> is an integer indicating the \U600\Ini\U600MMI.Ini file
- sectionName is the name of the .INI section (the text in square brackets)
- valueName identifies the entry (the text on the left-hand side of the "=")
- defaultValue is the value to returned if the entry is not found

This command will read a value from a specified entry in an ASCII text file, which is formatted as a Windows .INI file. You can provide the name of the INI file in the first parameter, or you can provide an integer value, which indicates an .INI file used by the MMI. The table below indicates the integer values representing each of the MMI .INI files. See the MMI setup page (or the \U600\Ini\U600.Ini file) for the file specifications corresponding to the names in the second column below:

0	Axis Configuration File
2	Axis Parameter File
3	Machine Parameter File
4	Task Parameter File
5	Global Parameter File

By using these integer values you can read parameter values from the parameter files (see example below) that the MMI automatically loads on a soft reset.

If an absolute file specification is not provided, the file will be written to the operating systems directory. If the specified section or entry name does not exist in the file, the "defaultValue" parameter value will be returned. If the specified file does not exist, a task fault will be generated.

EXAMPLE PROGRAM:

```
file: "FILE.INI" contents:  
[Section1]  
Variable1 = 11  
Variable2 = 22  
[Section2]  
Variable3 = 33  
  
; Example program commands (results in comment at end of line):  
$GLOBAL0 = FILEREADINI "FILE.INI" "Section1" "Variable2" -1 ; sets $GLOBAL0 to 22  
$GLOBAL1 = FILEREADINI "FILE.INI" "Section1" "garbage" -1 ; sets $GLOBAL1 to -1
```

**6.4.25.6. File Write Command****FILEWRITE**

SYNTAX1: **FILEWRITE** <fVariable>~<parameterList>

SYNTAX2: **FILEWRITE** <fileName>~<parameterList>

EXAMPLE: See File Example Program 1 on page 6-41, or File Example Program 2 on page 6-42.

See the COMMINIT command for a serial port example program.

The <fVariable> is the file handle returned by the FILEOPEN command.

The <fileName> is the file to write the data to.

The <parameterList> is the data to write to the file or serial port.

The **FILEWRITE** command permits the user to write one line of data to a file or a serial port. The first parameter may be a variable specifying the file handle returned by a successful FILEOPEN command, or the filename to write the data to. If a filename is specified the file will be automatically opened, the data will be written, and the file will then be closed. In this mode, text is ALWAYS appended to the file. If the file does not exist it will be created.

The **FILEWRITE** command will by default, write 6 digits past the decimal point for each value. You may specify differently in the <parameterList>. Each value is separated by a space. Each line written by a **FILEWRITE** automatically terminates with a carriage return, linefeed pair, see below.

```
127.000000 4096.000000 65536.000000
```

```
256.000000 8192.000000 32768.000000
```

FILE EXAMPLE PROGRAM 1:

```

; The below example, if the file operations succeeded, will write the file
;"c:\test.dat". If any file operation fails, it will erase the c:\test.dat" file.
; The file will read as follows (except no semicolon as the 1st character)
;This is line 1
;5.000000 4.000000 3.200000 6.000000 9.000000
;Strings are allowed ! 4 along with variables
;Masks are allowed too ! 7 along with variables
;This line was appended
;
DVAR $fil $var1 $var2 $var3 $var4 $var5           ; Declare the variables
$fil = FILEOPEN "c:\test.dat"                      ; Open the file new
IF ErrCode GOTO fail; Goto fail routine if the command was unsuccessful
$var1=5
$var2=4
$var3=3.2
$var4=6
$var5=9

FILEWRITE $fil "This is line 1"                   ;write line 1
IF ErrCode GOTO fail; Goto fail routine if the command was unsuccessful
FILEWRITE $fil $var1 $var2 $var3 $var4 $var5      ;write line 2
IF ErrCode GOTO fail; Goto fail routine if the command was unsuccessful
FILEWRITE $fil "Strings are allowed !" $var2 "along with variables" ;write
                                                               ;line 3
IF ErrCode GOTO fail; Goto fail routine if the command was unsuccessful
FILEWRITE $fil "Masks are allowed too !" X Y Z "along with variables" ;write
                                                               ;line 4
IF ErrCode GOTO fail; Goto fail routine if the command was unsuccessful

$var1=9
$var2=6
$var3=3.2
$var4=4
$var5=5

FILECLOSE $fil

$fil = FILEOPEN "c:\test.dat" 2                  ; Open the file append
IF ErrCode GOTO fail; Goto fail routine if the command was unsuccessful
FILEWRITE $fil "This line was appended"          ; Write line 5
IF ErrCode GOTO fail; Goto fail routine if the command was unsuccessful

FILECLOSE $fil

M02          ; end of program
:fail        ; error handler routine
EXE "erase c:\test.dat"
M02          ; Handle error here or end program

```

FILE EXAMPLE PROGRAM 2:

```
; The example will write data, and close it. Text will be
; appended to the file. If the file does not exist it is created.

DVAR $var1

$var1=0
WHILE $var1 < 10 DO
    FILEWRITE "C:\U600\Test.Dat", "Count ", $var1
    $var1 = $var1 + 1
ENDWHILE
```

**6.4.25.7. FILEWRITEINI Command**

SYNTAX: **FILEWRITEINI** <fileName>, <sectionName>, <valueName>, <value>

FILEWRITEINI <fExpression>, <sectionName>, <valueName>, <value>

EXAMPLES: **FILEWRITEINI** “c:\u600\ini\test.ini”, “TestMe”, “Test1”, \$glob0

Or to write to the axis parameter file.

FILEWRITEINI 1, “AxisParm.1”, “KP”, 10

where

- <fileName> is the name of the .INI file, which may contain format specifiers.
- <fExpression> is an integer indicating the U600MMI.Ini file.
- <sectionName> is the name of the .INI section (the text in square brackets), which may contain format specifiers..
- <valueName> identifies the entry (the text on the left-hand side of the “=”), which may contain format specifiers..
- <value> is the value to be returned if the entry is not found.

This command will write a value to the specified entry in an ASCII file, formatted as a Windows .INI file. You can provide the name of the .INI file as the first parameter, or you can provide an integer value, which indicates an .INI file used by the MMI. The table below indicates the integer values representing each of the MMI .INI files. See the MMI setup page, (or the \U600\Ini\U600.Ini file) for the file specifications corresponding to the names in the second column below:

0	Axis Configuration File
2	Axis Parameter File
3	Machine Parameter File
4	Task Parameter File
5	Global Parameter File

By using these integer values you can save parameter values into the parameter files (see example below) so that the U600MMI will automatically reload the new value on a soft reset.

If an absolute file specification is not provided, the file will be written to the operating systems directory. If the section or entry name provided does not exist within the file, it will be appended to the file. If the specified file does not exist, the file will be created.

EXAMPLE PROGRAM:

```
FILEWRITEINI 3, "MachParm.3" "HomeType" 2 ; Set HomeType machine  
; parameter, for axis 3, to 2
```

6.4.26. Free axes**FREE**

SYNTAX: **FREE** <axisMask>

EXAMPLE: **FREE X Y**

This command should not normally be used. Axes should be bound to a task within the axis configuration wizard and captured by another task and then released by that task for the original (or other) task to command motion on.



This command is the opposite of the **BIND** command; it releases control of the axis from the current task. Refer to the **BIND** command for more details.

6.4.27. FREECAMTABLE Command

SYNTAX: **FREECAMTABLE** <tableNumber>

EXAMPLE: **FREECAMTABLE 1**

Where;

tableNumber is the table number from 1 to 99.

This command frees a cam table from the controllers memory. See the Camming Overview for more information.

6.4.28. Goto to a CNC block

GOTO / JUMP

SYNTAX: **JUMP** <label>
<NCBlock>
...
:<label> ; The user can equivalently use: <label>
<NCBlock>

The **GOTO** command, along with the colon symbol, will “go to” or transfer execution to another location in the program. Instead of proceeding with the next sequential program block, execution immediately proceeds to the command following the label command with the colon. The colon can appear before or after the label name. Also, the programmer can use the **IF-GOTO** form when conditionally executing a **GOTO**.

The named location can appear anywhere in the program, before or after the line containing the **GOTO**, or within a subroutine. There is no limit to the number of **GOTOS** that may reference the same label, but a label can only be defined (using the colon) once in a program.

In any circumstance, the block commands; DFS, IF, REPEAT, and WHILE, can alternately be used instead of the **GOTO**. We strongly recommend that the programmer avoid the use of the **GOTO** by using block commands, in order to improve the readability of the program.



There is no difference in program speed between code using **GOTOS** and code using block commands. All of these translate into “**GOTOS**” in the object code.

We strongly recommend that the user not jump into block structures, since this can make code very confusing. For example, if a **GOTO** statement jumps into a **DFS** defined subroutine, then execution will not return (as in a normal **DFS** call) back to the point of the go to at the **ENDDFS** command. The **ENDDFS** command will simply be ignored and execution falls through to the statement immediately following the **ENDDFS**.

EXAMPLE PROGRAM:

```
goto setIO
M02 ; This line is never executed, because of the goto above it

:setIO
$BO1 = 1
M02
```

6.4.29. HANDWHEEL Command**HAND / HANDWHEEL**

SYNTAX: **HAND** <encoderCh><axis><dist_per_encoder_count>

Where:

- encoder_ch* - specifies the encoder channel the handwheel is connected to (0-32).
- axis* - is the axis to be positioned by the handwheel.
- dist_per_encoder_count* - is the distance the axis will move for each count of the handwheel in user units.

The <dist_per_encoder_count> parameter may be specified as negative value to reverse the direction of the axis.

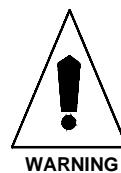


The **HANDWHEEL** command permits the user to manually position an axis with a hand wheel having a standard quadrature encoder output (or with any other device having the same). The hand wheel connects to the UNIDEX 600 Series controller via a spare encoder channel input, so X4 multiplication will be done on the hand wheel quadrature signal producing four times the number of counts per revolution specified by the handwheel manufacturer. Some hand wheels actually produce four counts per step of the hand wheel, so in these cases there will be 16 increments of the axis for each handwheel step. Large values for the distance parameter will produce jerky motion, since the axis will “jump”, the dist_per_encoder_count for each handwheel step, possibly causing one of the axis faults to disable the axis. Disabling the **VFF** and **AFFGAIN** axis parameters provides smoother hand wheel operation, then re-enable them after completion of the use of the handwheel.

The encoder channel parameter may specify any of the sixteen possible encoder channels, 1 through 16 (channels 5 through 16 are located on the encoder expansion cards 1 through 3 respectively). It may also specify the commanded positions of axes 1 through 16 by specifying an encoder channel of 17 through 32, respectively. Specify encoder channel zero (0) or distance of zero (0) to disable the hand wheel command for a particular axis.

The user may execute multiple handwheel commands sequentially to permit simultaneous multi-axis positioning, since this is an asynchronous command; see the example program.

If you repeatedly “handwheel” and “de-handwheel”, you may have to set the **MASTERPOS** axis parameter before enabling gearing to avoid 32-bit overruns and resultant jerky motion of the slave.



WARNING

EXAMPLE PROGRAM:

```
HAND 3 X .01      ;Position X axis via handwheel on encoder channel 3  
HAND 4 Y .1       ;Position Y axis via handwheel on encoder channel 4  
MSGDISPLAY 0 "Press OK when finished positioning"  
MSGSHOW  
HAND 0 Y .1       ;Disable Y positioning  
HAND 0 X .01      ;Disable X positioning  
MSGHIDE
```

6.4.30. Home Command**HOME / REF****SYNTAX:** **HOME** <axisMask>

The **HOME** or **REF** command will home the specified axes to their hardware zero reference point. There are several types of home cycles, and parameters that affect homing. Axes may be forced to be homed and a homing sequence may be defined via the Home Setup on the MMI Options Page of the Setup page. Axes may be manually homed via the Jog page.

Once issuing this command, parts program execution halts until all axes are “in-position” at their respective hardware home (absolute reference) positions.

If a virtual axis is homed, it will immediately set the position to the home position, rather than simulating any motion. If an axis is in the simulation mode or machine lock mode when it is homed, it will immediately set the current position to the value of the HomeOffsetInch (or HomeOffsetDeg) task parameter.



Homing will cancel all fixture offsets and presets.
Homing will disable normalcy, cutter offset and cutter radius compensation modes.



The **HOME** and **REF** commands may be used interchangeably.

EXAMPLE PROGRAM:

```
HOME X Y      ; The X and Y axes are sent home simultaneously.  
REF X Y       ; Initiate a homing sequence simultaneously on the X and Y axes.
```

6.4.31. HOMEASYNC Command**HOMEASYNC****SYNTAX:** **HOMEASYNC** <*axisLetter*>

The asynchronous Home command is the same as a synchronous home, except, it is asynchronous and the user may only home one axis at a time. There are several types of home cycles, and parameters that affect homing.

The move is asynchronous and program execution resumes immediately after the move starts. The controller does not wait for the home cycle to end before continuing on to the next command.

If a virtual axis is homed, it will immediately set the position to the home position, rather than simulating any motion. If an axis is in the Dry Run mode when it is homed, the home command will never complete, because the axis does not move in this mode.

The home speed is determined by the machine parameters, just like the synchronous home cycle. Refer to the HOME command for more details.

Homing will disable normalcy, cutter offset and cutter radius compensation modes.
Homing will cancel all fixture offsets and presets.

**EXAMPLE PROGRAM:**

HOMEASYNC X ;The X axis begins homing and the program continues.

6.4.32. IF Command**IF ... THEN ... ELSE ... ENDIF**

The **IF** command, useable in one of two contexts; as a directive to conditionally goto to a predefined program location, or as a directive to conditionally execute a block of commands. In either case, the user must supply a conditional expression. Refer to Chapter 3: *Expressions*, for descriptions on expressions.

6.4.32.1. IF ... GOTO command

SYNTAX: **IF** <conditionalExpression> GOTO <label>

EXAMPLE: **IF** (\$GLOBO > 5) GOTO exit
 IF \$GLOB5 GOTO exit

The IF command will first evaluate the expression. If the expression evaluates to non-zero (TRUE), then execution transfers to the CNC line immediately following the specified label (a line label has a colon before or after it). If the expression evaluates to zero (FALSE), then no action takes place and the line immediately after the IF statement executes.

Normally, a conditional operator is used in the expression, (see the first example above), but, this is not necessary (see the second example above). In the second example, the branch is taken if global variable five is non-zero.

EXAMPLE PROGRAM:

```
if ($B11 EQ 1) goto stopSpindle
M02                       ; Normal exit

:stopSpindle
M5
M02                       ; error exit
```

6.4.32.2. IF ... THEN Command

SYNTAX: **IF** <conditionalExpression> [[**THEN**]]

...

[[**ELSE IF** <conditionalExpression>]]

...

[[**ELSE IF** <conditionalExpression>]]

...

[[**ELSE**]]

...

ENDIF

The ellipses ‘...’ above indicate a series of zero or more CNC program lines. The user may have any number of **ELSE IF**s, but only one **ELSE**. The user cannot have any **ELSE IF**s after the **ELSE**. The **THEN** statement is optional.

This construct permits the user to execute a group of program blocks only if a specified condition is true. Using the **ELSE** statement, the user can execute an alternative group of program blocks if the condition is determined to be false. Finally, using the **ELSE IF**, the user can test multiple conditions in a sequence.

The CNC blocks in between the **IF** and the next **ELSE IF**, **ELSE**, or **ENDIF** (whichever comes first) are the blocks that execute when the **IF** conditional expression is true. If the **IF** conditional is false, then execution transfers to the next **ELSE IF**, **ELSE** or **ENDIF**, whichever comes first. If that command is:

1. **ENDIF**, then execution transfers to the command following the **ENDIF**.
2. **ELSE**, then it executes the CNC blocks between the **ELSE** and the next **ENDIF**.
3. **ELSE IF**, then it evaluates the **ELSE IF** conditional in exactly the same way as the **IF** conditional statement.

The **ENDIF** reserved word terminates the entire construct. **IF THEN** commands may be nested within each other or WHILE or REPEAT loops (see the following example). There is no limit to the level of nesting, except those imposed by available memory.

EXAMPLE PROGRAM:

```
; This example determines what quadrant the current XY coordinate is in.
; (quadrant 1 is X>0,y>0 and quadrant 4 is X<0,y<0 etc.)
; If X,Y, is on the origin, it sets the quadrant number to 0

IF (POSCMD.X > 0)
    IF (POSCMD.Y > 0)
        quadrant = 1
    ELSE IF (POSCMD.Y < 0)
        quadrant = 4
    ELSE
        quadrant = 1      ; Let points on +X axis be in quadrant 1
    ENDIF
ELSE IF (POSCMD.X < 0)
    IF (POSCMD.Y > 0)
        quadrant = 2
    ELSE IF (POSCMD.Y < 0)
        quadrant = 3
    ELSE
        quadrant = 2      ; Let points on -X axis be in quadrant 1
    ENDIF
ELSE                      ; X must be zero to get here
    IF (POSCMD.Y > 0)
        quadrant = 2      ; Let points on +Y axis be in quadrant 1
    ELSE IF (POSCMD.Y < 0)
        quadrant =        ; Let points on -Y axis be in quadrant 3
    ELSE
        quadrant = 0      ; This point is the origin, do not know the quadrant
    ENDIF
ENDIF
```

6.4.33. INDEX Command**INDEX****SYNTAX:** INDEX <axisLetter> <distance> <speed>Where: *distance* and *speed* are <*fExpressions*>**EXAMPLE:** INDEX X 50. 300.

The asynchronous motion command initiates a relative move on a designated axis at the specified speed, then continues with the next program line without waiting for the index to finish. Distance is specified as a signed parameter in user units and velocity is specified in user units/minute.

The move is an asynchronous motion command so program execution resumes immediately after the move starts. The controller does not wait for the move to end before continuing on to the next command.

The **INDEX** command uses the same acceleration/deceleration axis parameters as a G0 command. It does not use the acceleration/deceleration task parameters, like a G1 command.

6.4.34. IsAvail, Axes Available Command**SYNTAX:** ISAVAIL (*axismask*)**EXAMPLE:** IF(ISAVAIL(Zx)&&(SQRT(\$glob8==0))then

...

endif

The **ISAVAIL** command returns a TRUE (1), if the axes specified are all of the following:

1. The axis is not moving (the bms_moving bit is not set – i.e., the axis is not executing a **G0**, **G1**, **HOME** or asynchronous move command).
2. The axis is not synced up (the bms_sync bit is not set).

Otherwise, it returns FALSE (0), where the bms_bits are in the system status.

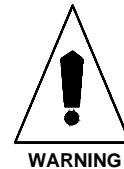
6.4.35. Camming Motion Overview

Electronic camming (master/slave) functions allow the user to command master/slave motion. Master/slave motion was originally developed for grinding cam shafts, but has many other uses, such as forcing an axis to follow a handwheel. For historical reasons master/slave motion is alternately referred to as "camming".

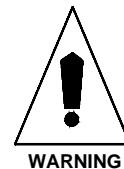
The camming mode allows you synchronize an axis (the slave) to another axis (the master) and command the master, thereby indirectly moving the slave axis. Master/slave motion is the most general form of motion allowing the user to command slave axes to move with virtually any position or velocity profile. Furthermore, by synchronizing multiple slave axes to a common master, or by using slave axes as masters to other axes, the user may move multiple axes in a fully synchronized manner. Another major advantage of camming motion is that the controller performs camming independently of other forms of motion. This allows the programmer to infed or command motion "on top off" the camming motion. Finally, camming is more efficient than regular contoured motion, since it is far simpler for the controller to process cam points than CNC lines. When you need to program a series of moves that are only a few milliseconds long, camming is the preferred choice.

However, there is a drawback to the master/slave motion, the programmer must understand and do more to achieve the proper results.

Camming motion is performed independently of other motions. It is not effected by the setting of any G code, global, machine or task parameter. Camming is only controlled by the camming commands, the parameters given to those commands, and a few related axis parameters. These commands and the parameters are described in the below sections.



Camming does not provide any acceleration limiting or feedrate limiting or automatic acceleration features, so it is easier to accidentally program "jerks" in the motion. The velocity and acceleration of the slave axis is defined solely by the master motion, the cam table, and the synchronization process, and is not limited in any fashion (SYNC mode 3 is the only exception, providing some acceleration limiting).



There are four types of camming motion that can be performed, listed here from the simplest to the most complex:

Handwheel Motion	: user controlled handwheel motion
Gearing	: ratioed to another axis
Tracking	: same as gearing, but with velocity ramping
File-Driven (typical master/slave camming)	: arbitrary velocity/distance from a file

Any or all of the following steps may be required to configure camming motion, based upon the type of camming motion being generated (Table 6-6).

Table 6-6. Configuring Camming Motion

Configuration Step	Handwheel	Gearing	Tracking	File-driven
0. Select the master axis	Master selection	Master selection	Master selection	Master selection
1. Configure the master	CFGMASTER	CFGMASTER	CFGMASTER	LOADCAMTABLE
2. Load the Cam table	Not Applicable	Not Applicable	Not Applicable	LOADCAMTABLE
3. Initialize Master position	MASTERPOS	MASTERPOS	MASTERPOS	MASTERPOS
4. Set Camming Parameters				
5 Synchronization	HANDWHEEL	GEARMODE	TRACK	SYNC
6. Monitor Camming Motion				

After step 5 the slave is activated and will respond to master motion. You can move the master using any other form of motion. The master can even be a slave to yet another master (for example, see Synchronizing multiple axes). Note that you may or may not be able to perform step 5 (and step 6 below) while the master is moving, see the particular step 3 command listed above for details. After the master/slave motion is complete you should perform the following cleanup activities:

Table 6-7. Configuring Camming Motion Cleanup

Configuration Step	Handwheel	Gearing	Tracking	File-driven
7. Un-Synchronize the axes	HANDWHEEL	GEARMODE	ENDM	SYNC
8. Free the Cam table	Not Applicable	Not Applicable	Not Applicable	FREECAMTABLE

After step 6 you may move the master axis without the slave moving.

6.4.35.1. Axis Parameters Affecting Camming

MASTERPOS, MASTERLEN

CAMOFFSET, CAMADVANCE, MAXCAMACCEL (file-driven/handwheel only)

CAMPOSITION, CAMPOINT (file-driven only)

GEARSLAVE, GEARMASTER, GEARMODE (gearing and tracking only)

6.4.35.2. Axis Parameters Used To Monitor Camming Motion

The following read-only parameters are rarely required, but, may be used to monitor the camming process.

MASTERPOS, MASTERRES

CAMPOSITION, CAMPOINT (file-driven only)

6.4.35.3. Camming Performance Tip

For best results, the master axis resolution (CntsPerInch / CntsPerDeg) should be equal or greater than the slave axes resolutions. The resolution of the slave motion will only be as good as the smaller of the two axes CntsPerInch values. This applies even if the master axis is a virtual axis.

In the case of virtual axes, the CntsPerInch (or CntsPerDeg) machine parameter may be arbitrarily scaled up to match the slave axes resolutions. However, if a virtual master is scaled up such that it exceeds 2^{16} counts/sec, then the plotting utilities will display a rolled over position (-2^{16}). This problem does not effect actual motion, as the actual velocity counter used is 32-bit.

6.4.35.4. 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 than 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.

6.4.35.5. Synchronizing Multiple Axes

You may synchronize multiple axes to a single master axis, or form a chain of master/slave relationships. All the master slave axes relationships are treated independently. The following example uses the HANDWHEEL command to synchronize multiple axes, but you may use any camming motion type, or combinations of types, in a similar manner.

For example, to have the Y and Z axes follow the X axis:

```
; synchronize the Y axis to the X  
HANDWHEEL <encoderCh> Y <dist_per_encoder_count>  
; synchronize the Z axis to the X  
HANDWHEEL <encoderCh> Z <dist_per_encoder_count>  
;  
; Command X axis Motion, G1, Joystick , etc.  
;  
; Disable slaving action below  
HANDWHEEL 0 Y 0  
HANDWHEEL 0 Z 0
```



The <dist_per_encoder_count> specified may be negative to reverse the direction of either slaved axis.

You must specify the encoder channel numbers of the Y and Z axes above. After doing so, any motion commanded on the X axis will cause corresponding motion on the Y and Z axes, based upon the value and sign of the <dist_per_encoder_count> specified for each slave axis.

6.4.35.6. Camming Motion from a File

For each master/slave relationship, the programmer must provide a table of coordinates (a cam table) that specify slave axis positions/velocities for each master axis position. These coordinates must be provided in an ASCII file.

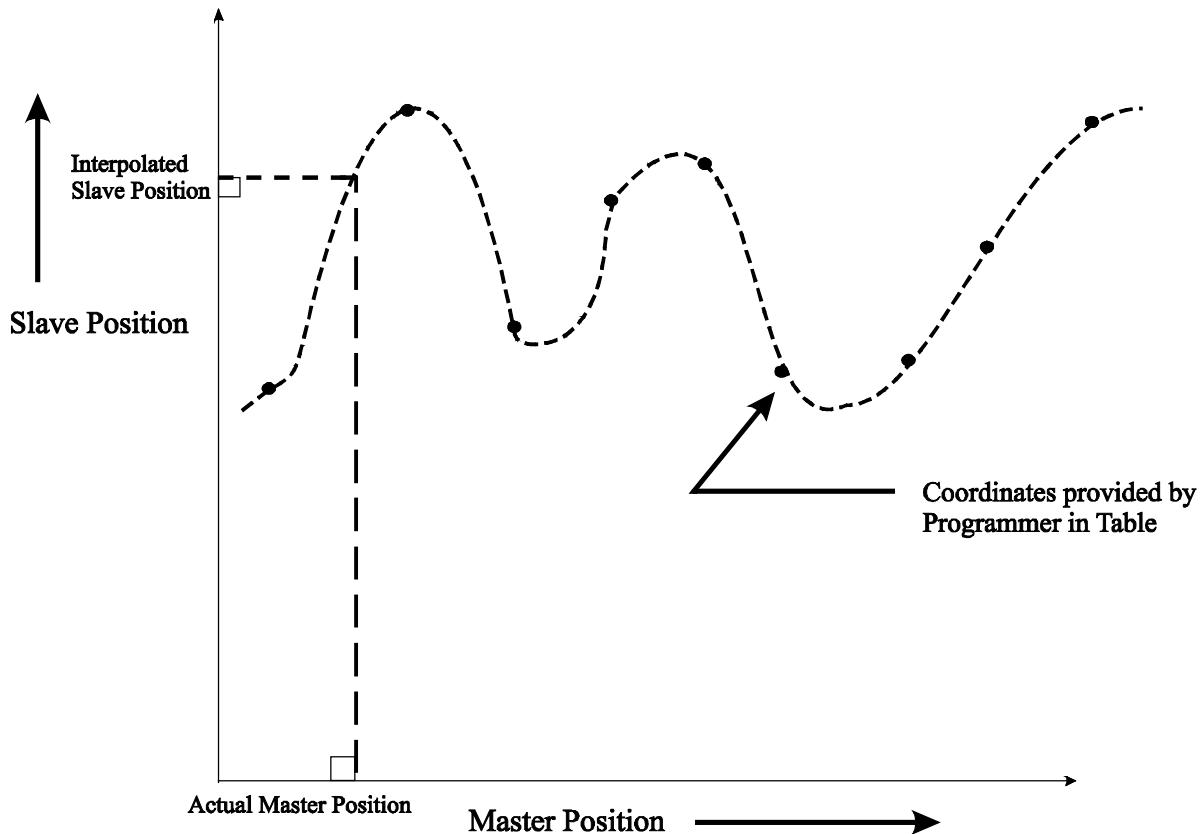


Figure 6-2. Master/Slave Profile

When the current position of the master axis is a value specified in one of the above points, then the slave axis position will take on the corresponding slave value of the point. If the current master position lies in between points specified in the table, then the slave position is found by either linear or spline interpolation of the adjacent points, (see the *SYNC* command). However, if the current position of the master is outside the range of the master values provided by the user, then the current master position will be "wrapped" so as to fall within the specified values in the table. For example, if the master points in the table range from 0 to 10, and the master axis is currently at position 12, then the master point for value 2 will be used. Therefore, if the master axis is not rotary, the user should provide points for the entire range of potential master motion.

The slave axis can act as a master to another axis, allowing the user to direct multiple axes in master/slave motion. Also, instead of slave axis positions, the programmer may specify slave axis velocities for given master axis positions (see *SYNC* mode 3). One can also add an "infeed" onto the slave while it is camming. The infeed is specified as a position and speed, (see the *FEDM* command) and the required motion produced by the infeed will be added onto the cam directed motion.

The basic steps needed to direct master/slave motion are listed below with the function(s) detailing that step.

1. Load the Cam table.
2. Optionally, offset the cam table via the slaves CAMOFFSET axis parameter.
3. Synchronize the axis.
4. Generate master axis motion.
5. Un-Synchronize the axes.
6. Free the Cam table.



The axis needs to be enabled before step 4 in the above procedure.



Care should be taken when engaging cam table motion as it's incorrect application may result in abrupt changes in slave velocity. Make sure that the master axis is stationary when engaging or disengaging cam table motion for best results.

6.4.35.7. Infeeding Overview

The UNIDEX 600 controller generates velocity commands that are translated into torque by the motors. These velocity commands the controller generates for each motor is actually the sum of two separate velocity commands:

$$\text{Velocity_cmd} = \text{Camming_Velocity_command} + \text{Non-camming_Velocity_command}$$
$$\text{command} + \text{AUXVELCMD_generated_Velocity_command}$$

The camming velocity command is computed based on the current (if any) camming motion, and the non-camming velocity command is computed based on the current (if any) synchronous and asynchronous motion. This command summation allows the programmer to independently direct two motions. For example, the camming motion can direct a tool to follow the contour of an irregular rotating piece (a cam) at a constant distance from its surface. The programmer can then use asynchronous motion to infeed or slowly move the tool closer to the surface (synchronous motion can not be done concurrently with camming motion). Furthermore, in complex configurations, the user may add more velocity command via the AUXVELCMD axis parameter. AUXVELCMD generated velocity command is more general than camming, and can be added to any other type of motion.

6.4.35.8. Asynchronous Motion Commands

Asynchronous motion commands, cause CNC program execution to continue on to the next program block immediately after the move starts. The controller does not wait for the move to end before continuing on to the next command, therefore the task will be executing other statements while the motion is proceeding. Asynchronous motion gives you the freedom to complete other actions while performing a time consuming move. However, it is the programmer's responsibility to insure that the asynchronous motion has actually completed, if a subsequent command depends upon the completion of this motion. For example, a slow carriage move might need to be completed before a part

loader can approach the carriage to avoid a collision. The two other available types of motion are synchronous and camming. Asynchronous motion commands also include the AerMovxxx library commands.

ENDM	- end motion
FEDM	- in-feed an axis, in addition to other motion in progress
HOMEASYNC	- home an axis without waiting for completion
INDEX	- move incrementally
MOVETO	- move to an absolute position
OSC	- oscillate (cycle at the specified distance and velocity)
STRM	- free run an axis at the specified velocity

If you start an asynchronous motion command on an axis before a previous asynchronous motion command on that axis has completed, it will immediately abandon the previous command, then perform the second command.

You cannot generate asynchronous motion while polar (**G46**) or cylindrical (**G47**) transformation modes are active.

Although they do not wait for the end of the move, asynchronous motion statements will wait until the following conditions are satisfied before proceeding onto the next statement: (these restrictions do not apply to the ENDM command):

1. FeedHold is not active.
2. No axis specified in the move is still moving in response to another non-camming motion command (**G0 /G1** or other asynchronous move)
3. The done bit of the SERVOSTATUS axis parameter is off for each axis in the move.
4. The “moving” bit of the MOTIONSTATUS axis parameter is set on for each axis in the move (The “Profile” bit in the STATUS axis parameter indicates if G1/G2/G3 commands are executing.

The last condition can also be used to test whether an asynchronous motion command has completed. For example, the following example program could be used to monitor motion initiated from an asynchronous motion command, setting binary output 1 on when it starts, and setting it off when its done::

EXAMPLE PROGRAM:

```
#include "\u600\programs\aeerstat.pgm" ; need this included at top of program
INDEX X 100 100 ; starts X moving
$BO1 = 1 ;
; You may complete other tasks (commands) here, while the X axis is moving !
;
WHILE( MOTIONSTATUS.X & MOTIONSTATUS_Moving ) ; Wait till X is done moving,
; this does not mean that it is In-Position!
ENDWHL
WHILE( SERVOSTATUS.X & SERVOSTATUS_InPosition ) ; Wait till x axis is in-position
ENDWHL
$BO1 = 0
```

Asynchronous motion commands have another important use, they can be used while camming motion is active. This allows the programmer to infeed.

6.4.35.9. Camming Example Program

```

; ExCam.Pgm (Ref. Jol_Cam.Cam also)
;
; Electronic Cam Table Example program
DVAR $TableNumber, $Interpolation
;
; Motion Setup (C is the master axis, Z is the slave axis)
;
E100                                ; define rotary axes RPM
G91 G70                               ; absolute, metric modes
DRIVE.C = 1                            ; enable the drives
DRIVE.Z = 1
$TableNumber = 3
CAMOFFSET.C = 0                         ; set zero offset into table

; Turn off camming (if it was on)
SYNC Z $TableNumber 0                  ; disable any active sync.

; Home the axes, as required.
REF Z C                                ; home the axes

; Activate camming
$Interpolation = 0                      ; use mode 1 for splines
LOADCAMTABLE C $TableNumber Z $Interpolation "\u600\programs\Jol_Cam.cam"

MASTERPOS.Z= POSCMD.C
SYNC Z $TableNumber 1                   ; Sync. Z axis to table 0, mode 1

; Here we move the master (C), which moves the slave (Z)
M0
G91                                    ; incremental programming mode
STRM C 1 100

; Disable camming when complete !
SYNC Z $TableNumber 0                  ; disable any active sync.

; Master axis motion will not command slave axis motion.
FREECAMTABLE $TableNumber             ; release memory for the cam table

M2

```

6.4.35.10. Cam Table Format

The cam table file consists of data lines and comment lines. A line is defined as a string of characters, terminated by any number of consecutive line terminator characters. A line terminator character is a carriage return or a line feed.

Data lines are defined as those with the first "non-whitespace" character as a numeric character. All other lines are comment lines. Whitespace characters are the tab, formfeed, comma, or space character. A numeric character is a digit, decimal point, or minus sign.

The format of the ASCII file is described in the paragraphs below:

COMMENT LINES:

The file must contain comment lines preceding the first data line, that provide three pieces of information:

The number of points in the file.

The units of the master position values.

The units of the slave position values.

The number of target points must be preceded by the keyword: "Number of Points" The first numeric character after this keyword will begin the number read in as the number of target positions. One data line must be read for each target position. The units of the position values must be preceded by the keyword: "Master Units". The first non-whitespace character which is not a "(" will begin the master position units descriptor. Position units descriptors may be "nanometers", "deg", "degrees", "mm", "in", "microns", "millimeters", or "inches". Identification of the master position units descriptor is not as sensitive.

The units and scale factor of the slave position values must be preceded by the keyword: "Slave Units(". The first non-whitespace character which is not a "(" will begin the slave scale factor. The slave scale factor is optional, and assumed to be one, if not specified. The first non-whitespace character after the slave scale factor must be the slave position units descriptor. Slave Position Units descriptors can be "nanometers", "deg", "degrees", "mm", "in", "microns", "millimeters", or "inches". Identification of the slave position units descriptor is not case sensitive.

DATA LINES:

Each data line must contain three numbers, the first of which must be an integer. All text following the first three numbers will be ignored. The numbers must be separated by one or more whitespace characters.

The first number is the point number. It must be a positive integer. These point numbers are not used.

The second number is the master position value. This value is assumed to be in the units specified in the comment lines at the beginning of the file.

The third number is the slave value that goes with the position value. It is assumed to be in the units specified in the comment lines at the beginning of the file, i.e. if the value read in is -2, and the slave axis scale factor compensation read in was .25 microns, then the actual compensation value is -.5 microns.

6.4.35.11. Cam Table Format Example

; Filename: J01_Cam.Cam (See ExCam.Pgm also)

Number of Points 206					
Master Units (deg)					
Slave Units (inch)					
0001	0.000	.3720	0048	44.619	.1769
0002	1.000	.3720	0049	45.615	.1698
0003	1.928	.3719	0050	46.612	.1627
0004	2.857	.3716	0051	47.611	.1556
0005	3.785	.3711	0052	48.611	.1485
0006	4.714	.3704	0053	49.614	.1414
0007	5.642	.3695	0054	50.618	.1343
0008	6.571	.3684	0055	51.626	.1272
0009	7.499	.3671	0056	52.637	.1201
0010	8.427	.3655	0057	53.653	.1130
0011	9.356	.3638	0058	54.674	.1059
0012	10.284	.3619	0059	55.703	.0989
0013	11.213	.3598	0060	56.739	.0919
0014	12.141	.3575	0061	57.785	.0850
0015	13.070	.3550	0062	58.842	.0782
0016	13.998	.3522	0063	59.911	.0715
0017	14.927	.3493	0064	60.993	.0649
0018	15.857	.3462	0065	62.088	.0586
0019	16.786	.3429	0066	63.198	.0525
0020	17.717	.3393	0067	64.320	.0468
0021	18.648	.3356	0068	65.454	.0414
0022	19.580	.3317	0069	66.595	.0364
0023	20.513	.3276	0070	67.740	.0320
0024	21.447	.3233	0071	68.884	.0280
0025	22.383	.3188	0072	70.024	.0245
0026	23.320	.3142	0073	71.159	.0215
0027	24.259	.3093	0074	72.284	.0189
0028	25.200	.3043	0075	73.400	.0168
0029	26.144	.2991	0076	74.504	.0150
0030	27.090	.2937	0077	75.594	.0135
0031	28.039	.2882	0078	76.670	.0124
0032	28.990	.2825	0079	77.733	.0115
0033	29.945	.2766	0080	78.781	.0107
0034	30.902	.2706	0081	79.816	.0101
0035	31.863	.2645	0082	80.840	.0096
0036	32.828	.2583	0083	81.854	.0091
0037	33.795	.2519	0084	82.860	.0087
0038	34.766	.2454	0085	83.862	.0083
0039	35.739	.2389	0086	84.862	.0079
0040	36.716	.2322	0087	85.862	.0075
0041	37.696	.2255	0088	86.862	.0071
0042	38.679	.2187	0089	87.862	.0067
0043	39.664	.2118	0090	88.862	.0063
0044	40.651	.2049	0091	89.862	.0059
0045	41.640	.1979	0092	90.862	.0055
0046	42.632	.1909	0093	91.862	.0051
0047	43.625	.1839	0094	92.862	.0047

0095	93.862	.0043	0151	306.347	.1130
0096	94.862	.0039	0152	307.363	.1201
0097	95.862	.0035	0153	308.374	.1272
0098	96.862	.0031	0154	309.382	.1343
0099	97.862	.0027	0155	310.386	.1414
0100	98.862	.0023	0156	311.389	.1485
0101	99.862	.0018	0157	312.389	.1556
0102	100.865	.0014	0158	313.388	.1627
0103	101.876	.0011	0159	314.385	.1698
0104	102.894	.0007	0160	315.381	.1769
0105	103.918	.0004	0161	316.375	.1839
0106	104.944	.0002	0162	317.368	.1909
0107	105.967	.0001	0163	318.360	.1979
0108	106.986	.0000	0164	319.349	.2049
0109	107.996	.0000	0165	320.336	.2118
0110	109.000	.0000	0166	321.321	.2187
0111	110.000	.0000	0167	322.304	.2255
0112	179.000	.0000	0168	323.284	.2322
0113	180.000	.0000	0169	324.260	.2389
0114	267.000	.0000	0170	325.234	.2454
0115	268.000	.0000	0171	326.205	.2519
0116	269.004	.0000	0172	327.172	.2583
0117	270.015	.0000	0173	328.137	.2645
0118	271.035	.0001	0174	329.098	.2706
0119	272.066	.0003	0175	330.055	.2766
0120	273.104	.0005	0176	331.010	.2825
0121	274.149	.0009	0177	331.961	.2882
0122	275.198	.0014	0178	332.910	.2937
0123	276.248	.0021	0179	333.856	.2991
0124	277.297	.0029	0180	334.800	.3043
0125	278.344	.0039	0181	335.741	.3093
0126	279.387	.0050	0182	336.680	.3142
0127	280.428	.0063	0183	337.617	.3188
0128	281.466	.0076	0184	338.553	.3233
0129	282.504	.0091	0185	339.487	.3276
0130	283.544	.0107	0186	340.420	.3317
0131	284.588	.0124	0187	341.352	.3356
0132	285.641	.0143	0188	342.283	.3393
0133	286.706	.0164	0189	343.214	.3429
0134	287.784	.0188	0190	344.143	.3462
0135	288.880	.0215	0191	345.073	.3493
0136	289.992	.0245	0192	346.002	.3522
0137	291.120	.0280	0193	346.930	.3550
0138	292.260	.0320	0194	347.859	.3575
0139	293.405	.0364	0195	348.787	.3598
0140	294.546	.0414	0196	349.716	.3619
0141	295.680	.0468	0197	350.644	.3638
0142	296.802	.0525	0198	351.573	.3655
0143	297.912	.0586	0199	352.501	.3671
0144	299.007	.0649	0200	353.429	.3684
0145	300.089	.0715	0201	354.358	.3695
0146	301.158	.0782	0202	355.286	.3704
0147	302.215	.0850	0203	356.215	.3711
0148	303.261	.0919	0204	357.143	.3716
0149	304.297	.0989	0205	358.072	.3719
0150	305.326	.1059	0206	360.000	.3720



6.4.36. LOADCAMTABLE Command

SYNTAX:

LOADCAMTABLE<masteraxis><tableNumber><slaveAxis><interpolationType><filename>

Where:

<masteraxis> is the master axis number.

<tableNumber> is the table number from 0 to 99.

<slaveAxis> is the slave axis number.

<interpolationType> is either 0 (linear) or 1 (cubic spline.)

<filename> is filename

<flag> 0 = slave tracks master position feedback

1 = slave tracks master position command

EXAMPLE: LOADCAMTABLE C 1 Z 0 “\U600\Programs\Jol_Cam.Cam”



For best results, the master axis resolution (CntsPerInch / CntsPerDeg) should be equal or greater than the slave axes resolutions. This applies even if the master axis is a virtual axis.

In the case of virtual axes, the CntsPerInch (or CntsPerDeg) machine parameter may be arbitrarily scaled up to match the slave axes resolutions.



All Callback commands will set the ErrCode task parameter, if an error occurs during execution.

This function will read an ASCII file containing master position (or master position command) and slave position (or slave velocity) information, download the table to the controller and configure the master and slave axes. The flag parameter is used to determine if the master position feedback or master position command is to be used. The mode argument of the SYNC command to determine if slave position or slave velocity is specified.

Each line in the file is one cam table point. After the first data line is read from the file, an EOF (end of file) or comment line, will terminate the file and it will be closed, and the cam table points downloaded to the controller. Each cam table point will occupy 24 bytes of memory on the controller and all points allocated will be filled with zeros after allocation. Table numbers can range from 0 to 99.

The interpolation type specifies either cubic or linear splining. A cubic spline is used for maximum "smoothness", and linear for minimum "rippling". The splines are 3rd order,

but are not true Bsplines. In true Bsplines, an entire n by n matrix (where n is the number of points) must be solved to obtain values for all the points. Instead, for speed and memory optimization a 2 by 2 matrix for each pair of points, carrying the resultant derivatives over to the adjacent pair as the boundary conditions for the next splining point. The results are nearly identical to true Bsplines and in some cases are preferable.

Once SYNCed, the controller will then use the current master axis' position as an index into the cam table, to obtain the slaves position value. This value is then used as the commanded position of the slave axis. The LOADCAMTABLE function must be executed before the SYNC command. The successful engagement of the cam table can be verified by testing the sync mode bit of the STATUS (AerStat) axis parameter for the specified slave axis after executing the SYNC command.

All cam table entries consist of a master position and a slave value, where no two table entries have the same master position (see Cam Table File Format or the AerCamTableSetPoint function in the Library Reference manual for more details on table entries). If the current master position is equal to a table entry's master position, then the value from that entry in the table is used as the current slave value. If however, the current master position lies in-between two master positions listed in the table, then either linear or spline (see InterpolationType parameter) interpolation is used to find a slave value between the two slave values listed in the table. Finally, if the current master position lies outside of the table, that is, if the current position of the master is outside the range of the master values provided by the user, then the current master position will be "wrapped" so as to fall within the values specified in the table. For example, if the master points in the table range from 0 to 10, and the master axis is currently at position 12, then the master point for value 2 will be used. Therefore, if the master axis is not rotary, the user should provide points for the entire range of potential master motion.

6.4.37. #MAKENCODESLABELS

SYNTAX: #MAKENCODESLABELS

The #makencodeslabels statement directs the controller to convert all N codes in the CNC program to labels, for use with the M97 and M98 M codes. Otherwise, the N codes are ignored. However, always converting N codes to labels can unnecessarily waste memory on the controller, which can be a concern for large programs with lots of N code labels.

This command must be within the main CNC program, it cannot be within an included file.



6.4.38. MAP Command**MAP****SYNTAX:** **MAP <axisPoint>****except:** the floating point expressions for each axis must evaluate to an integer.

Axis mapping is performed on a per task basis. The mapping relates a task axis (X, Y, Z,...) to a physical axis. All task related functions and commands use the task axes. At execution time the task axes are translated into physical axes. This unique mapping per task, allows the use of all axis identifiers for each task. Also, it allows the same program to run on different tasks, controlling different sets of axes, with each task having a set of axes by the same axis names.



The physical axis numbering is one-based (i.e., the first physical axis is 1).

If the expressions provided do not evaluate to integers, the UNIDEX 600 Series Controller generates a fault. All axes to be mapped, must be free, or unbound. If the user tries to MAP an axis that is bound, the UNIDEX 600 Series Controller generates a fault.

EXAMPLE PROGRAM:

```
MAP X1 Y2 Z3      ; X now means channel 1, Y channel 2, etc.  
MAP X3 Y2 Z1      ; X now means channel 3, Y channel 2, etc.
```

6.4.39. MaskToDouble Command**SYNTAX:** **<variable>=MASKTODBL <axisMask>****EXAMPLE:** **\$GLOB0 = MASKTODBL X Y Z** ;\$GLOB0 = 7(1+2+4)

The **MASKTODBL** command converts a task axis mask to its equivalent integer value.

6.4.40. MOVETO (Asynchronous Absolute Move) Command

SYNTAX:MOVETO <axisLetter> position speed

Where: *position* and *speed* are <*fExpressions*>

EXAMPLE:MOVETO X 50. 300.

The asynchronous motion command initiates the move of an axis to a specified absolute position at the specified speed, then continues with the next program line without waiting for the move to finish. The absolute position is specified in user units, velocity is specified in user units/minute. The move direction is determined via the commanded absolute position, which is in user units.

The move is an asynchronous motion command so program execution resumes immediately after the move starts. The controller does not wait for the move to end before continuing on to the next command.

The **MOVETO** command uses the same acceleration/deceleration axis parameters as a G0. It does not use the acceleration/deceleration task parameters, like a G1.

Example Program

;This program illustrates changing the target position and velocity on the fly with out deceleration

```
ENABLE X Y ;enable the axes

MOVETO X 10 10
MOVETO Y 20 10

;simulate a delay before target position change
G4 F1 ;dwell 1 second

MOVETO X 20 500
MOVETO Y -10 500
;simulate a delay before target position change
G4 F5

MOVETO X 150 50 ;direction reversal
MOVETO Y 150 50
;simulate a delay before target position change
G4 F10
ENDM ;stop the axes
```

6.4.41. MSET Command

SYNTAX: **MSET <AxisLetter> <vector>**

EXAMPLE: **MSET X 90 ; Set X axis to 90 electrical degrees**

This command is used with brushless motors to output the specified electrical vector to the specified axis, typically for alignment or debugging purposes. The peak current commanded will be limited by the IAVGLIMIT axis parameter.



Be sure the IAVGLIMIT axis parameter is set to the continuous current rating of the motor, so as not to damage it !

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 this command.

CALLBACK

6.4.42. MSGxxx Commands Overview

The UNIDEX 600 Series controller allows the CNC parts program to interface with the operator via message windows referred to as the Custom Display Window (CDW). The **MSGxxx** series of commands permit the operator to show and hide the window (behind the program tracking displays), display information to the display list window via the MSGDISPLAY command. As well as, display messages to the user with simple button selected responses, as well as solicit numeric data from the user, via the MSGBOX and MSGINPUT commands, respectively. The MSGLAMP# commands allow warning or informational text to displayed as illuminated text LAMP's within the G code display.

Each of the MSGxxx command indicates a failure by setting the ErrCode task variable to a non-zero value. This variable should be tested by the user after the MSGxxx commands. All of the MSGxxx commands may contain format specifiers for their string parameters.



All of these (Callback) commands will set the ErrCode task parameter, if an error occurs during execution.

EXAMPLE PROGRAM for MSGxxx commands:**See \U600\Programs\Display.Pgm also**

```
DVAR $WIDTH $HEIGHT
MSGDISPLAY 1 "Performing program startup" ; Type 1 message
MSGDISPLAY 2 "You can create long" " strings by concatenating them !"

MSGCLEAR 1 ; Clear all type 1 messages from display
$WIDTH = 1
$HEIGHT = 2
MSGDISPLAY 3 "The square is: " $WIDTH "millimeters. by " $HEIGHT "millimeters."
MSGCLEAR -1 ; Clear all messages from display
```

6.4.42.1. MSGBOX Command**SYNTAX:** **MSGBOX** <flags><parameterList>

The <flags> define the buttons and icons displayed within the **MSGBOX**.

The <parameterList> is the data to display.

The MSGBOX command allows the user to display a pop-up message box to the operator displaying a warning or informational message (Figure 6-3).

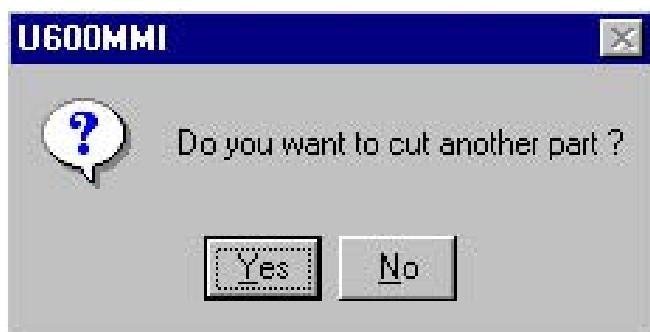


Figure 6-3. MSGBOX Pop-up Message Example

All Callback commands will set the ErrCode task parameter, if an error occurs during execution.



The flags specify the buttons and icons that will be displayed in the window to the operator. The icons provide added clarity for the event. The icons that may be displayed are an X (critical), ! (exclamation), ? (question mark) and an 'i' (information symbol), as shown in the table below. Also, a particular button may be set as the default button, so that if the return key is pressed that response will be selected. This is done by using the default button defines as shown in the table below. These defines allow you to select the first, second or third button as the default button when the return key is pressed by the operator. The first, second and third buttons are defined by the order they are specified in the MSGBOX button defines, in Table 6-8. For example, in the define DF_MSGBOX_ABORTRETRYIGNORE, the ignore button would be set as the default

using the DF_DEFBUTTON3 define, because it is the third button in the define. These buttons will also appear on the screen from left to right, once again the ‘Ignore’ button will be the third button.

Icon #Defines	Value
DF_ICON_CRITICAL	16
DF_ICON_QUESTION	32
DF_ICON_EXCLAMATION	64
DF_ICON_INFORMATION	128
Default Button #Defines	Value
DF_DEFBUTTON1 (first button)	0
DF_DEFBUTTON2 (second button)	256
DF_DEFBUTTON3 (third button)	512

Table 6-8. Button Specifiers

Button #Defines	Value	Description
DF_MSGBOX_OKONLY	0	OK Button
DF_MSGBOXOKCANCEL	1	OK & Cancel Buttons
DF_MSGBOX_ABORTRETRYIGNORE	2	Abort, Retry & Ignore Buttons
DF_MSGBOX_YESNOCANCEL	3	Yes, No & Cancel Buttons
DF_MSGBOX_YESNO	4	Yes & No Buttons
DF_MSGBOX_RETRYCANCEL	5	Retry & Cancel Buttons

Lastly, the <parameterList> is a combination of strings and variables producing the desired message string.

Additionally, the button pressed by the operator may be tested for as shown in the following program fragment, using the defined return values for the buttons, shown in the table following the example below.

EXAMPLE:

```
$resp = MSGBOX DF_MSGBOX_YESNO + DF_ICON_QUESTION, "Are you Sleepy ?"
MSGDISPLAY "Resp = ", $resp
IF $resp == YES_BUTTON THEN          ; If YES button pressed
    DISPLAY DF_LIST, "Me Too !"
ENDIF
```

Button Return Value Defines	Value
OK_BUTTON	1
CANCEL_BUTTON	2
ABORT_BUTTON	3
RETRY_BUTTON	4
IGNORE_BUTTON	5
YES_BUTTON	6
NO_BUTTON	7

6.4.42.1.1. ParameterLists and Format Specifiers

Parameter Lists may contain string variables with format specifiers within them, the replacement text (where applicable) must follow. The format specifier must be the first string in the MSGDISPLAY command parameter list, after the idType. Its syntax is as follows:

“{specifier1..specifier10}”

A “format specifier” may be used to format the text (strings) within callback commands. The special sequence is replaced with the requested data. For example, “the time is #tm” becomes “the time is 2:21 AM”. Special string sequence recognition is not case sensitive. The #C, #D, #F, #H and #U specifiers, must be within brackets '{}' and the first string in the list. The #DT and #TM specifiers may be within any string and they will be replaced, these should not be within brackets, they should be embedded within the text string.

The following formats may be specified:

String Sequence	Meaning
#C	ASCII character.
#D	Integer
#DT	Current Date (format as specified under Regional Settings, Control Panel)
#F	Floating point (optionally, may specify the number of decimals, i.e. #F3, default is 6, maximum is 15)
#H	Hexadecimal number
#TM	Current Time (format as specified under Regional Settings, Control Panel)
#U	Unsigned integer

Currently there is a maximum limit of 10 specifiers that may be used per MSGDISPLAY command. This will require breaking a string into multiple strings, as shown below.

FORMAT SPECIFIER EXAMPLE:

```
MSGDISPLAY 1, "{#F3 #F}" "Time is #TM, Float 3 decimals and default " $var1 " " $var1  
MSGDISPLAY 1, "{#D #H}" "Integer and Hex " $var1 " 0x" $var1  
MSGDISPLAY 1, "{#D #U #H}" "Integer, Unsigned, and Hex " -1 " " -1 " 0x" -1
```



6.4.42.2. MSGCLEAR Command

SYNTAX: **MSGCLEAR** <*idType*>

The <*idType*> is the numerical type of the message to be cleared.

The **MSGCLEAR** allows messages of the specified type to be cleared from the message display window. All messages of all types may be cleared by specifying -1 for the *idType*.

EXAMPLE:

```
MSGDISPLAY 1 "Warning – Coolant Low" ;Warning priority  
MSGDISPLAY 1 "Warning – Pressure Low" ;Warning priority  
MSGDISPLAY 2 "Fault – Pressure Zero!" ;Fault priority  
MSGCLEAR 1 ;Clear all priority 1 messages  
MSGCLEAR 2 ;Clear all priority 2 messages  
MSGCLEAR -1 ;Clear all messages!
```

6.4.42.3. MSGDISPLAY Command



SYNTAX: **MSGDISPLAY <idType><parameterList>**

The <idType> is the numerical type of the message.

The <parameterList> is the *formatted* data to display.

All Callback commands will set the ErrCode task parameter, if an error occurs during execution.



Each display message writes one line of text into the CDW display list window (Figure 6-4). If the display window already contains messages, by default, the new message is placed on a line below the previous message. A scroll bar allows the user to browse through previous messages.

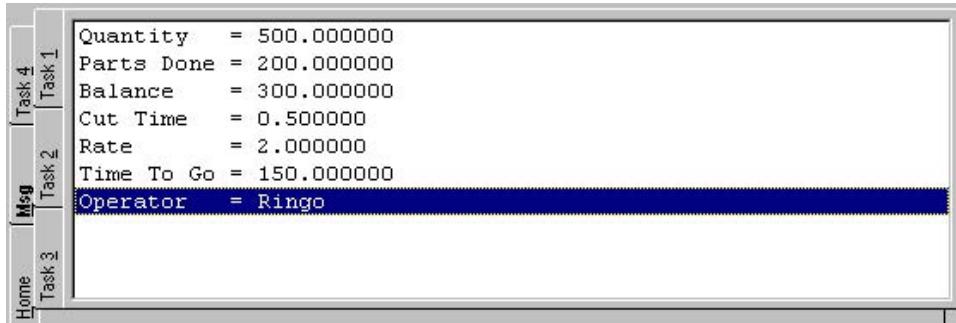


Figure 6-4. The CDW Display List Window

Additionally, each message may be assigned a numerical message type when the message is added to the display list. This allows messages to be assigned a priority, such as fault or warning. This allows all messages of a certain type to be cleared from the display list, without clearing messages of another priority. The value of -1 should not be assigned as a message type. This value is used to clear all messages from the window without regard to the priority assigned to the message.

A “format specifier” may be used to format the text within the **MSGDISPLAY** command.

EXAMPLE:

```
$var1 = 0h5A7B
MSGDISPLAY 1, "Default " $var1
MSGDISPLAY 1, "Warning – Coolant Low" ; Warning priority
MSGDISPLAY 1, "Warning – Pressure Low" ; Warning priority
MSGDISPLAY 2, "Fault – Pressure Zero !" ; Fault priority
MSGCLEAR 1 ; Clear warnings only
```



6.4.42.4. MSGHIDE Command

SYNTAX: MSGHIDE

The **MSGHIDE** command will hide the message display, restoring the current tasks program tracking display. The user may also swap these at any time by selecting the **MSG** or Task tabs of the program tracking display. This command hides messages of all types regardless of their priority.



All Callback commands will set the ErrCode task parameter, if an error occurs during execution.

EXAMPLE:

MSGHIDE ;Hide all messages



6.4.42.5. MSGINPUT Command

SYNTAX: MSGINPUT <Flags><parameterList>[default value]

The <Flags> define the type of data that may be input by the user and the buttons that will be displayed, as defined in Table 6-9 and Table 6-10.

The <parameterList> is the semicolon-delimited data to be displayed. See the text below for more information.

The [default value] is an optional value to be displayed as the default value within the **MSGINPUT** window.

The **MSGINPUT** command will display a message box (Figure 6-5) allowing the operator to be prompted for data.



Figure 6-5. MSGINPUT Command Message Box Display

By default, floating point numbers may be entered, if neither of the flags in Table 6-9 are specified. The value of the OK and/or Cancel button is returned in the *ErrCode* task parameter. This will be set to the value of 1 for the OK_BUTTON or 2 for the CANCEL_BUTTON. If a default value is specified the return value will NOT be changed if the CANCEL button is pressed.

Table 6-9. Input Window Specifiers (* = DEFAULT)

#DEFINE	Hex Value	Description
DF_INPUT_INTEGER	0h800000	Allow integer input only
DF_INPUT_STRING	0h400000	Allow string (text) input

Table 6-10. Button Specifiers (* = DEFAULT)

#DEFINE	Hex Value	Description
DF_MSGBOX_OKONLY	*0	OK Button (default)
DF_MSGBOXOKCANCEL	1	OK & Cancel Buttons

The parameter list is a somewhat complex string containing the semicolon-delimited data to be displayed. The parameter list contains two items, each ending with a semicolon. The first item is the alphanumeric string to be displayed across the title bar of the **MSGINPUT** window. This string may be any combination of strings and variables, and must end with a semicolon (;). The second item within the string is the prompt to be displayed within the **MSGINPUT** window. This string may be any combination of strings and variables, and must end with a semicolon (;).

EXAMPLE:

```
MSGINPUT DF_MSGBOX_OKONLY "Title Bar;Prompt string;" $Default_value
```

This will display “Title Bar” in the window’s title bar and the operator will be prompted with the message “Prompt String”. The value contained in the \$Default_value variable will be displayed in the window, allowing the operator to select “OK” for that value or to enter a new value.

EXAMPLE:

```
$Title = 600
```

```
$ser_num = 3
```

```
$num = 7
```

```
$default = 4
```

```
MSGINPUT DF_MSGBOX_OKONLY "UNIDEX" $Title "Number" $ser_num "Enter the" $num  
"the value;" $default
```

This will display “UNIDEX 600 Number 3” in the title bar of the window, and will prompt the operator with the message “Enter the 7th value”. The value of 4 will be initially displayed as the default value in the window.

EXAMPLE:

```
$strglob0 = MSGINPUT (DF_INPUT_STRING), "Guess a phrase;Enter any phrase;", "Did you  
tighten the Screw?"  
MSGDISPLAY 1, $strglob0
```

**6.4.42.6. MSGLAMP# Command****SYNTAX: MSGLAMP# <color> <parameterList>**

The MSGLAMP# commands allow warning or informational text to be displayed as illuminated text LAMP's within the G code display. There are three Lamps numbered 1-3, from top to bottom of the G code display.



All Callback commands will set the ErrCode task parameter, if an error occurs during execution.

The <color> parameter is defined by a mixture of RGB color values. You may specify an RGB color mixture or use one of the pre-defined colors. There are eight colors pre-defined in the \U600\Programs\AerParam.Pgm file. They are:

GRAY	gray
BLACK	black
RED	red
BLUE	blue
MAGENTA	magenta
GREEN	green
YELLOW	yellow
CYAN	cyan

The <parameterList> is the alphanumeric text to be displayed, limited by the size of the display area within the 'Lamp'. Approximately, 7-9 characters may be displayed within the lamp.

EXAMPLE:

```
MSGLAMP1 RED "Warning!" ; Warning lamp
```

6.4.42.7. MSGMENU Command



SYNTAX: [variant =] **MSGMENU** <Flags> <id> <text>

<Flags> = DF_MENU_ADD, DF_MENU_SHOW, DF_MENU_REMOVE

DF_MENU_ADD - Adds an Item to the list, <id> is value of Item, <text> is what is displayed.

DF_MENU_SHOW - Shows the MSGMENU box, <id> can be DF_MSGBOX_OKCANCEL or DF_MSGBOX_OKONLY.

DF_MENU_REMOVE - Removes an item(s) from list, <id> is item to remove (-1 to remove all).

All Callback commands will set the ErrCode task parameter, if an error occurs during execution.



The MSGMENU command displays Figure 6-6 a list of items allowing the operator to select the appropriate option.

To display a MSGMENU, ADD the appropriate items and then use the DF_MENU_SHOW id with the MSGMENU command.

The value of the option selected is returned in the optional return variable. The ErrCode task parameter will contain the value of the button pressed.

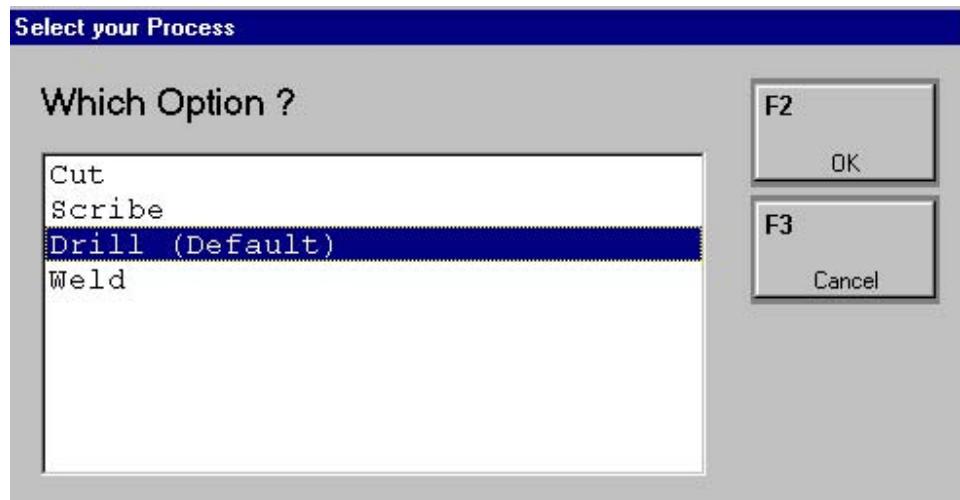


Figure 6-6. MSGMENU Command Display

Example:

```
MSGMENU (DF_MENU_REMOVE), -1 ""
MSGMENU (DF_MENU_ADD), 1 "Cut"
MSGMENU (DF_MENU_ADD), 2, "Scribe"
MSGMENU (DF_MENU_ADD), 3, "Drill (Default)"
MSGMENU (DF_MENU_ADD), 4, "Weld"
$glob5 = MSGMENU (DF_MENU_SHOW), (DF_MSGBOX_OKCANCEL), "Select your
Process;Which Option ?;", 3
MSGDISPLAY 0, "You selected option" $glob5 " Key pressed(" ErrorCode ")"
```

**6.4.42.8. MSGSHOW Command****SYNTAX: MSGSHOW**

The MSGSHOW command will display the message display, hiding the current tasks program tracking display. These may be swapped by the user at any time by selecting the MSG or Task tabs of the program tracking display.



All Callback commands will set the ErrorCode task parameter, if an error occurs during execution.

**6.4.42.9. MSGTASK Command****SYNTAX: MSGTASK <TaskNumber> ""**

EXAMPLE: MSGTASK 0, "" ; Clear current task fault message from task 1

The MSGTASK command will clear the task fault message from the task display area of the U600 MMI display. The <TaskNumber> is zero-based, with 0 representing task number 1. This will not clear the task fault itself. This may be done by setting the TaskFault task parameter to zero for the affected task, i.e.;

TaskFault.1 = 0 ; clear task fault from task 1



All Callback commands will set the ErrorCode task parameter, if an error occurs during execution.

6.4.43. ON command

ON

SYNTAX: **ON** <conditionalExpression> **SET** <fVariant> [<OnValue>] [<OffValue>]
 [<Mode>]
ON CLEAR <conditionalExpression>
ON CLEAR

<conditionalExpression> is the expression to evaluate.
 <fVariant> is the variable/parameter/IO to set the value of .
 <onValue> is the value to set the <fVariant> to when the
 <conditionalExpression> evaluates to TRUE. This defaults to 1 if not provided.
 <offValue> is the value to set the <fVariant> to when the
 <conditionalExpression> evaluates to FALSE. This defaults to 0 if not provided.

Where:

 <Mode> is ONSET_MODE_LEVEL, ONSET_MODE_EDGE, or
 ONSET_MODE_LATCH. This defaults to ONSET_MODE_LEVEL if not specified.

The **ON** command defines a condition to monitor that will set the specified variant (variable or parameter) to the <onValue>, if the defined condition is true. It can also assign an <offValue> to the <fVariant>, when the defined condition is false. On command monitors are active from the moment they are defined, until they are cleared for all programs on the task (the task on which the program was running on, when it defined the monitor). Also, the user can clear a single monitor, or clear all monitors, using the “ON CLEAR” command as shown in the example below.

The ONSET_MODE_LEVEL <Mode> will continuously assign the <OnValue> to the <fVariant> when the <conditionalExpression> is true, and will continuously assign the <OffValue> to the <fVariant> when the <conditionalExpression> is false. For example, in the ONSET_MODE_LEVEL <Mode>, if the <fVariant> is \$GLOB0, then assigning to \$GLOB0 in a CNC program has no effect since its value will always be immediately overwritten by the ON statement. In contrast, the ONSET_MODE_EDGE mode will only assign to the <fVariant> when the value of the <conditionalExpression> changes (and when the ON condition is first defined). The EDGE and LATCH modes differ from the LEVEL mode, in that for the former, the <fVariant> is set only when the <conditionalExpression> changes. In contrast, the LEVEL mode is constantly assigning to the <fVariant>. For example, if the <fVariant> is \$GLOB0 and the EDGE mode is used, then assigning to \$GLOB0 in a CNC program has no effect since its value will be immediately overwritten by the ON statement.

The frequency at which this command evaluates the expression is variable based upon available controller time. The time under a given situation may be examined by reading the value of the AvgPollTimeSec global parameter.

The CNC programmer should be aware that monitors do not clear automatically when a program finishes executing. Monitors are related to the task, not the program. Therefore, if the user defines a monitor in a program without clearing it, the next program run on that task will still have that monitor active. Normally, the CNC programmer should use the **ON CLEAR** statement at the end of the program defining the monitor condition.



There is a limit to the number of established monitors, determined by the task parameter MaxMonitorData.

The old **ON** syntax is valid

```
ON( $BO0==1 ) SET $GLOB0
```

and is equivalent to the following new syntax:

```
ON( $BO0==1 ) SET $GLOB0 1 0 ONSET_MODE_LEVEL
```

EXAMPLES:

```
ON $GLOB0 > 5 SET $BO2 ; Sets a On monitor  
ON CLEAR $GLOB0 > 5 ; Clears a On monitor  
ON CLEAR ; Clears all On monitors  
ON( $BO0==1 ) SET $GLOB0 100 20 ONSET_MODE_LEVEL  
ON( $BO1==1 ) SET $GLOB1 100 20 ONSET_MODE_EDGE  
ON( $BO2==1 ) SET $GLOB2 100 20 ONSET_MODE_LATCH
```

ONSET_MODE_LEVEL

The ONSET_MODE_LEVEL mode of the **ON** command, specifies that when a condition is TRUE the variable is set to the On Value. When it is FALSE, the variable is set to the Off Value. The variable is always set, if changed by the user/program it is forced back to the ON/OFF Value. This is the default mode, if none is specified.

Set <fVariant>=<OnValue> whenever <conditionalExpression> is not 0.

Set <fVariant>=<OffValue> whenever <conditionalExpression> is 0.

ONSET_MODE_EDGE

The ONSET_MODE_EDGE mode of the ON command, sets the variable to the specified values on the rising and falling edges. On a Low-High Transition (condition goes from FALSE to TRUE), the variable is set to the OnValue. On a High-Low Transition (condition goes from TRUE to FALSE), the variable is set to the OffValue. The variable is ONLY set on a transition.

Set <fVariant>=<OnValue> when <conditionalExpression> changes to a 1.

Set <fVariant>=<OffValue> when <conditionalExpression> changes to a 0.

ONSET_MODE_LATCH

The ONSET_MODE_LATCH mode of the ON command, specifies that on a Low-High Transition (condition goes from FALSE to TRUE) the variable is set to the OnValue. The offValue is not used. The variable is ONLY set on a transition.

Set <fVariant>=<OnValue> when <conditionalExpression> becomes 1

6.4.44. Conditional ONGOSUB Command**ONGOSUB****SYNTAX:**

```
ONGOSUB <conditional expression> FARCALL programname [[<label>]] priority
ONGOSUB CLEAR <comparatorExpression>
ONGOSUB CLEAR
```

Where:

conditional expression	
<i>programname</i> is a	<s32Expression>, as limited by filenames.
<i>label</i>	
<i>priority</i> is a	<fExpression> evaluating to: 1, 2 or 3 which may also be represented by PRIORITY_MEDIUM, PRIORITYFAULT, PRIORITY_HIGHEST, respectively.

EXAMPLES:

```
ONGOSUB $GLOB0 > 5 FARCALL "Prog1.Pgm" "ERROR_LABEL" PRIORITY_FAULT
ONGOSUB CLEAR $GLOB0 > 5 ; Clears an ONGOSUB
ONGOSUB CLEAR ; Clears all ONGOSUBs
```

The **ONGOSUB** command redirects program flow if the defined condition occurs. It operates similar to an interrupt. When an ONGOSUB is active (or defined), anytime the specified condition occurs during program operation, execution immediately transfers via a program call (see the FARCALL extended command) to the program location specified in the **ONGOSUB** command. When an ONGOSUB command transfers program control in this way it is said to be “triggered” or “fired.” After the program code in the ONGOSUB routine has completed executing, program control returns to the location from which the ONGOSUB was called from. This capability is especially useful for handling fault or safety conditions that must be monitored continuously. However, unlike the FARCALL command, the user cannot pass arguments to the program called in an **ONGOSUB** command, global or task variables however, could be used for this purpose. The ONGOSUB command is quite powerful, but its intricacies must be understood completely to insure its proper use.

The frequency at which the conditional expression is evaluated is based upon available controller time. The rate at which the conditional expression is evaluated the value of the AvgPollTimeSec global parameter.

ONGOSUBs are active for all programs on the current task from the moment they are defined until they are cleared. A user may clear a single or all **ONGOSUBs**.



The CNC programmer should be aware that **ONGOSUBs** do not clear automatically when a program finishes executing. **ONGOSUBs** are related to the task, not the program. Therefore, if the user defines an **ONGOSUB** in a program without clearing it, the next program run on that task will have that **ONGOSUB** active in it. Normally, a CNC programmer would use an **ONGOSUB CLEAR** at the end of a program defining an **ONGOSUB**. If however a program defines an **ONGOSUB**, then ends, the next program executing will suppress **ONGOSUBs** for the first line of the program only. This allows the programmer to place a **ONGOSUB CLEAR** as the first line in the next program.

There is a limit to the number of **ONGOSUBs** that may be defined, determined by the task parameter **MaxOnGosubData**.



If a label within a program is only used as the target of an **ONGOSUB** within another program, then the CNC program compiler will warn that the label is not used when compiling the program. This is due to the fact that the compiler cannot see the **ONGOSUB** reference in another program. The warning is benign and should be ignored.

It is important to understand that a triggered **ONGOSUB** does not de-trigger the condition causing it. Simply put, unless the programmer clears the **ONGOSUB**, or makes the condition triggering the **ONGOSUB** false or creates an **ONGOSUB** of higher priority, the same **ONGOSUB** immediately triggers again upon exiting the **ONGOSUB** routine.

The calling condition for an **ONGOSUB** is expressed the same as a condition for an IF or WHILE command, so the programmer can use any variable, parameter, binary input or output in the conditional statement.

ONGOSUBs are defined at program run-time, therefore, they are not active until the **ONGOSUB** command executes in the program. Also, the user can remove **ONGOSUBs** at any time by using the **CLEAR** keyword or overwriting an **ONGOSUB**, by defining a new **ONGOSUB** with the same conditional expression, but a different target. It is important to realize that when overwriting or clearing an **ONGOSUB**, the user must use the same exact conditional statement, so the controller recognizes it as the same **ONGOSUB**. For example, if an **ONGOSUB** is defined with the conditional ($\$BOB > 0$), then a **CLEAR** with ($\$BOB \geq 1$) does not clear the **ONGOSUB**.

The user can have multiple **ONGOSUBs** active at once, but never more than one pertaining to the same condition. If the user has one or more **ONGOSUBs** that both trigger at the same time, see priority's.

If the defined condition occurs during motion, the motion will stop before the **ONGOSUB** routine is called. The motion may or may not resume at the conclusion of the **ONGOSUB** routine depending on how the user returns from the **ONGOSUB** program.

Defining an OnGosub

When defining an **ONGOSUB** condition, at least three parameters must be specified: the condition to monitor (in the conditional expression), a program to call if the condition occurs (in the program name), and a priority (in the priority argument). Also, the user can optionally specify a parameter indicating a label within the program to begin execution at.

The Conditional Expression

A conditional expression is a <fExpression>. However, it is recommended to always surround the conditional expression with parenthesis - in some cases this may be necessary in order to avoid obscure and difficult to understand CNC compile errors.

Normally, a CNC programmer would use just the comparators in a conditional expression, (see the first three examples below). However, just as in the C programming language, the CNC programmer can use arithmetic operators also (see the fourth and fifth examples below). Note, that when arithmetic operators are used: commands that use conditional expressions are only concerned with whether the conditional expression evaluates to 0.0 (called FALSE) or non-zero (called TRUE). Also, note that unlike the C programming language, the user cannot use an assignment statement as a conditional expression (the syntax: “\$GLOBAL0=8” is not legal as a conditional expression).

EXAMPLES:

```
( $GLOB0 > 0 )
( SQRT( $GLOB0 + $GLOB1 ) EQ $GLOB2 )
( !( $GLOB1 > 0 ))      ; NOT may also be used for !
( $GLOB1-10 )            ; This evaluates to FALSE only when global variable
                        ; 1 is exactly 10.0
( ABS( $GLOB1 ) -2 )     ; This evaluates to FALSE only when global variable
                        ; 1 is exactly 2.0
```

The Program Name

You must provide the name of the program containing the routine to be executed when the ONGOSUB is true. The program containing the ONGOSUB routine does not need to be present on the UNIDEX 600 Series Controller at compile time, but must be present at runtime (you may use the download only mode of the program automation feature to accomplish such). If a NULL program name “” is supplied, the command will assume that the program to jump to is the current program being compiled. In this case a label should be supplied.

The Optional Label

If the user does not specify a label, execution will begin on the first line of the specified program. If this label is only used as the target of the ONGOSUB from within another CNC program, the compiler will warn that the label is not used, when compiling the program. This is due to the fact that the compiler cannot see the ONGOSUB reference in another program. The warning is benign and should be ignored.

The Priority

The priority provided must be a constant, and one of three values: PRIORITY_MEDIUM, PRIORITYFAULT, and PRIORITY_HIGHEST. Priorities are used to sort out the relationships between different ONGOSUBS and task faults, and are discussed under priority's. In general, PRIORITY_MEDIUM, is the most useful, however the user is strongly recommended to understand priorities before using ONGOSUBs.

Clearing an OnGosub

The user may remove an ONGOSUB definition at any time by preceding the conditional expression with the ‘ONGOSUB CLEAR’ keywords, and omitting the FARCALL, program name, program label and priority. Furthermore, the user can remove all ONGOSUBs defined on the task by also omitting the conditional expression. The user can also redefine an existing ONGOSUB, by defining a new ONGOSUB with the same conditional expression, but a different target. It is important to realize, that when overwriting or clearing an ONGOSUB, the user must use the exact same conditional statement, so the controller recognizes it as the same ONGOSUB. For example, if an ONGOSUB is defined with the conditional ($\$BOB > 0$), then a CLEAR with ($\$BOB \geq 1$) does not clear the previously defined ONGOSUB.

Debugging an OnGosub

When an ONGOSUB statement is successfully executed in a program, the ONGOSUB defined in the statement becomes active. However, if there is an error in the activation of the ONGOSUB statement (e.g. it can't evaluate the conditional expression, or can't find the ONGOSUB CNC program file or label) then it generates the task fault: “Can't evaluate an ONGOSUB”, and sets the TaskWarning and ErrCode task parameters to an error code indicating the problem. You may view the Task Fault and Task Warnings via the AerDebug.Exe utility, with the AerDebug commands: TK n (for n = the appropriate task number), then TSKI.

A common mistake with ONGOSUB's is to not clear the condition that generated it, before leaving the target routine. This results in the ONGOSUB immediately reoccurring after it is exited, therefore, never returning control to the original program.

Another common mistake is not to realize that an OnGosub will not run unless the program is running. This is defined in the Scope of an OnGosub.

Errors in an ONGOSUB

If the controller encounters an error executing an ONGOSUB, (such as a missing label or file) it will generate a task fault; "Can't execute an ONGOSUB". The actual error causing the problem (for example: "Invalid filename") will be indicated as a task warning (MMI users should run the AerDebug.exe utility, then type TSKI to view the task warnings).

Viewing active OnGosub Statements

Once an ONGOSUB fails, an evaluation, the controller no longer attempts to evaluate it. The user can view the ONGOSUB's defined (and their current status), using the ZON command within the AerDebug.exe utility (use the TK command first, to switch to the appropriate task). The state of the ONGOSUB will be "Active" if it evaluated OK, or "Broken" if it did not. If the description of the ONGOSUB in the ZON command is too long to view on the display (shows '...' at end of line), then use the OUTON command before the ZON command, and examine the file that was written for the full description. Keep in mind, that this ONGOSUB description line is in compiled text format, and its syntax may vary slightly from that which was entered in the CNC program, specifically with respect to the use of temporary variables to resolve complex expressions.

Scope of an OnGosub

When an ONGOSUB is active, it is active from EVERY and ANY program running on the task, including programs called from the original program. Furthermore, the ONGOSUB will not be deactivated when the program ends. Therefore, if the user defines an ONGOSUB in a program without clearing it, the next program run on that task will have that ONGOSUB active in it. Normally, a CNC programmer would use an ONGOSUB CLEAR at the end of a program establishing an ONGOSUB, to avoid this. The user can clear a single ONGOSUB, or clear all ONGOSUBs simultaneously.

However, an ONGOSUB will not be active when there is no program running on the task. If a task has no purpose other than to define the ONGOSUBs, then the programmer must place an "infinite loop" program on that task in order to keep the ONGOSUB active.

ONGOSUB conditions are not tested prior to the first line executed in a program. This allows you to clear the ONGOSUBs on the first line.

Multiple OnGosubs

The user can have multiple ONGOSUBs defined, but never more than one pertaining to the same condition. Even so, there is a limit to the total number of ONGOSUBs that may be defined in a task, determined by the MaxOnGosubData task parameter.

If there are multiple ONGOSUBs defined, one ONGOSUB may trigger while another is processing (see Priority's for details). The ONGOSUBs will behave exactly the same as regular program calls, in that, control returns back to the first ONGOSUB after the second one has completed. ONGOSUBs can be stacked no greater than the MaxCallStack task parameter.

Priorities

Each ONGOSUB is assigned a priority of either PRIORITY_MEDIUM, PRIORITY_FAULT or PRIORITY_HIGHEST. It should be emphasized that the FAULT priority level can be used with any condition, not just faults. However, special considerations (see Relationships to Faults) make this level a natural choice for use with faults.

The UNIDEX 600 Series Controller maintains a value known as “current priority”, which can be at any one of four priority levels, which are, in ascending order: PRIORITY_NORMAL, PRIORITY_MEDIUM, PRIORITY_FAULT and PRIORITY_HIGHEST. The current priority is normally at the lowest level: PRIORITY_NORMAL. However, once an ONGOSUB occurs, the current priority is reset to the ONGOSUB’s priority. Once that ONGOSUB exits, then the priority may or may not return to its level prior to that ONGOSUB occurring (see Returning From an OnGosub).

We must emphasize that restarting program execution, or associating a new program to the task does not clear the current priority level. Only an ONGOSUB, M47, FARGOTO/FARJUMP, task reset, or a firmware download changes the current priority level.

However, if an ONGOSUB condition evaluates true, it will execute only if its priority level is greater than the “current priority.” If the “current priority” is equal or greater than the ONGOSUB’s priority, the ONGOSUB is deferred until such time as the current priority falls below the ONGOSUBS priority. Therefore, an ONGOSUB of equal or lower priority can never interrupt an executing ONGOSUB command, but a higher priority ONGOSUB can.

EXAMPLE PROGRAM:

```
; Prg1.Pgm

$GLOBAL1 = $GLOBAL1+1      ; 1st line
$GLOBAL1 = $GLOBAL1-1      ; 2nd line
$GLOBAL2 = $GLOBAL1
$GLOBAL1 = 0
M02                      ; 3rd line

; Prg2.Pgm

$GLOBAL1 = $GLOBAL1+10     ; 1st line
M02                      ; 2nd line
```

Note, that the Prg2.Pgm interrupt is of higher priority.

```
ONGOSUB ($GLOBAL1 > 0) FARCALL "Prg1.Pgm" PRIORITY_MEDIUM
ONGOSUB ($GLOBAL1 == 2) FARCALL "Prg2.Pgm" PRIORITY_FAULT
```

`$GLOBAL1 = 2 ; After this line is executed, PRG2 is called, which executes
; the M2 after its 1st line executed. After PRG2 finishes,
; PRG1 runs. Final value of GLOBAL2 is 12.`

`$GLOBAL1 = 1 ; After this line is executed, PRG1 is called, after 1st line of
; PRG1 executed PRG2 is called, after its done, we go back to
; PRG1, final value of $GLOBAL2 is 11.`

If a number of ONGOSUBS are triggered (their conditions are true) simultaneously, the highest priority ONGOSUB will be executed. If one or more ONGOSUBS exist at the same priority, it executes the last one defined.

If a program called from an ONGOSUB command calls another subroutine or program, then the called program/subroutine has the same level of priority as the caller. For example, if in the above example Prg2.Pgm called another program, then like Prg2, that program would not be interrupted by the Prg1 ONGOSUB command.

Returning from an OnGosub

It is important to understand that a triggered ONGOSUB does not de-trigger the condition causing it. Simply put, unless the programmer clears the ONGOSUB, or makes the condition triggering the ONGOSUB false or creates an ONGOSUB of higher priority, the same ONGOSUB immediately triggers again upon exiting the ONGOSUB routine, resulting in an infinite loop.

There are three ways to exit a program, and each has its own special relationship to an ONGOSUB command:

the FARGOTO/FARJUMP command

the RETURN command

a termination M Code (M2, M47)

The FARGOTO command is the simplest to understand. When the user FARGOTOS anywhere, including the current program, it clears the current priority (sets it to PRIORITY_NORMAL). Therefore after the FARGOTO, ONGOSUBs of any priority may be triggered.

The RETURN command returns to the command executed when the ONGOSUB was triggered. If motion was running at the time the ONGOSUB was triggered, there are special considerations (see Relationship To Motion).

An M47, like a FARGOTO, clears the current priority level, allowing any ONGOSUB triggered to interrupt. An M2 exits the program, but does not reset the current level. For this reason it is NOT recommended to use an M2 to exit an ONGOSUB.

Relationship to Motion

If the ONGOSUB occurs during motion, the motion stops (is decelerated smoothly) before calling the ONGOSUB routine. After axes motion has been interrupted, you may move any axis, whether it was involved in the interrupted move or not. The motion will only resume (accelerate up to speed smoothly) after leaving the ONGOSUB routine, if the ONGOSUB is exited with a RETURN statement. Exactly how the motion is resumed depends upon the ReturnType specified, refer to the RETURN command for more information.

Relationship to Task Faults

Normally a TaskFault will stop a program that's running. However, if the "current priority" level (see Priority's) is PRIORITY_FAULT or PRIORITY_HIGHEST, a task fault will not stop the running program. Therefore, if the program is executing an ONGOSUB, the program will only be stopped by a task fault only if the ONGOSUB priority is PRIORITY_MEDIUM.

Often an ONGOSUB will be defined for the purpose of taking special action when a task fault occurs, for example: "**ONGOSUB** (TaskFault > 0) **FARCALL** "tmp.pgm". PRIORITY_HIGHEST". These ONGOSUBs must have their priority level defined as FAULT or HIGHEST, or else the ONGOSUB will stop immediately as it is triggered.

Miscellaneous

ONGOSUBS are not tested prior to the first line executed. This allows you clear the ONGOSUBS on the first line.

Unlike the FARCALL command, the user cannot pass arguments to the program called in an ONGOSUB, global or task variables however, could be used for this purpose.

ONGOSUB's conditional expressions are tested prior to the execution of each line and the interrupted line is understood to be the line that executes immediately after the ONGOSUB test. So, if a line defines a condition (ONGOSUB watches for \$BO7 > 0, and the executed line is \$BO7 = 1), then the "line after interrupted" line is actually the line, TWO lines after the \$BO7 = 1 line. Refer to the following example.

```
ONGOSUB ($BO7>0).....  
N10 $BO7=1 ; fires ONGOSUB manually  
N20 G1.. ; not this line, but  
N30 G1 .. ; This is the line returned to
```

EXAMPLE PROGRAM: (RunPgm.pgm)

```
ONGOSUB ($BI2 EQ 1) FARCALL SHIELD_SAFETY2 PRIORITY_FAULT
:DOSTUFF
; suppose shield comes up, this sets BI2=1, we jump to SHIELD_SAFETY2
; do more stuff
M2

;*****SET POINT *****
: SHIELD_SAFETY2
$BO2 = 1           ; closes shield, setting BI2 to 0
FARGOTO "RunPgm.Pgm" "SETPOINT"

;*****SET POINT *****
:SETPOINT
DISPLAY 0 "Going_to_Set_Point" 2
M5                 ; turn off spindle
G1 X0 Y0
GOTO DOSTUFF
```

6.4.45. Oscillate Move Command**OSC****SYNTAX:** OSCILLATE <axisLetter> <distance> <feedrate>Where: *distance* and *feedrate* are <*fExpressions*>**EXAMPLE:** OSCILLATE X 50. 300.

distance — is the distance of the moves (sign is initial direction).
feedrate — is the velocity in user units/minute.

The *asynchronous motion* command causes the specified axis to oscillate (cycle) the specified distance at the specified velocity. The sign (+ or -) of the distance determines the initial direction of the move and is in user units. The feedrate is in user units per minute. To halt the axis, specify a zero feedrate or distance in subsequent uses of this command, or execute an **ENDM** on that axis.

Each portion of the oscillation moves is a separate **INDEX** move. In other words, an “OSC X position velocity” is equivalent to:

```
While FOREVER
    INDEX X p v
    INDEX X -p v
EndWhile
```

Each portion of the oscillation move (INDEX in the example above) command will have its own acceleration and deceleration, as determined by the acceleration/deceleration axis parameters (**ACCEL**, **DECCEL**, **ACCELMODE/DECCELMODE**, **ACCELERATE**, **DECCELERATE**). Thus the shape of the oscillation profile is affected by these parameters.

The **OSCILLATE** command uses the same acceleration/deceleration axis parameters as a **G0** command. It does not use the acceleration/deceleration task parameters, like a **G1** command.

The move is asynchronous and program execution resumes immediately after the move starts. The controller does not wait for the move to end before continuing on to the next command.

6.4.46. POPMODES Command

SYNTAX: POPMODES

EXAMPLE: **POPMODES** ; Restore modal G code states

Restores the setting of the Model task parameter to the value before the PUSHMODES command was executed.

Executing this command prior to a PUSHMODES command will generate a "Empty Stack" fault.

6.4.47. PUSHMODES Command

SYNTAX: PUSHMODES

EXAMPLE: **PUSHMODES** ; Save modal G code states

Stores the setting of the Mode1 task parameter. This is typically used to store the modal G code states prior to calling a subroutine, so they can be restored upon exiting. However, it can be used at any time, along with POPMODES, to store and restore modal G code states.

EXAMPLE:

```
G70 G90  
PUSHMODES  
CLS routine  
POPMODES  
X10           ; at this point we are in G70 and G90 mode  
...  
M2  
  
DFS routine  
    G71 G91  
ENDDFS
```

6.4.48. Initialize Touch Probe**PROBE****SYNTAX:** **PROBE** (*channelNum*) (*activeLevel*) (*array[x]*)

Where:

- channelNum* is the virtual input bit number (-1 for high speed position latch)
activeLevel is the active level (1/0) (0 for high speed position latch)
array[x] is an array variable to store the positions

The UNIDEX 600 Series controller provides support for digital touch probe measuring. It allows the user to determine the location of a part in space.

The **PROBE** command initializes the touch probe and the G51 command activates probe monitoring. When the probe input is detected, the controller returns the current position of each axis in the specified variables. This allows the probe to begin moving toward the users part and have the probe stop when it contacts the part. The program may then use the position information returned to determine the physical location of the part in space.

When the probe input is detected, the controller will generate a Probe fault. You may disable, halt, or abort the motion, based on the setting of each axis fault masks: **DISABLEMASK**, **HALTMASK**, **AUXMASK**, **ABORTMASK**, **INTMASK**, and **BRAKEMASK**.



Be sure to set the Probe Fault bit in the FAULTMASK axis parameter to enable the detection of the touch probe, then set the bit in the appropriate mask parameter (DISABLEMASK, HALTMASK, AUXMASK, ABORTMASK, INTMASK and BRAKEMASK) for the action to occur on this fault.

Once activating a probe cycle, the controller actively monitors the designated input channel until the probe input is detected. When a probe touch occurs, the cycle is complete. To initiate a probe measuring cycle again, execute another G51 command.

The parameters for the probe command include the virtual I/O number, active level and array variable to store the axes positions. The first parameter, channel number, refers to the virtual input channel through which the probe input will be monitored. To specify the high speed position latch input on the UNIDEX 600 controller, enter -1 as the channel number. The high speed position latch input is a physical hardware trigger to a gate array, so, the trigger input must physically connect to the high speed position latch input on each card (UNIDEX 600, 4EN-PC #1, 4EN-PC #2, 4EN-PC #3) having axes whose position must be latched. The second parameter, active level, permits the user to specify the polarity of the probe in use. The high speed position latch hardware input is always active low, specifying an active high level will have no effect. The final parameter specifies the first location of an array into which the position information is stored. The size of the array should correspond to the number of axes specified by the G51 command multiplied by the number of probe hits to be taken before the execution of the next **PROBE** command. Each probe hit increments a pointer into the array by the number of axes being collected. The execution of the **PROBE** command resets the pointer to the beginning of the array.

The G51 command may be repeated after executing the **PROBE** command to determine various points on the users part, see example program below.

EXAMPLE PROGRAM:

```
DVAR $Posit[16]      ; Define $Posit program variable array to store positions
;
; Be sure the array is declared large enough to hold all of the points sampled !
;
PROBE 10 0 $Posit[0]    ; Initialize touch probe input on virtual input bit 10.
; The probe being used is active low (0). Position information
; will be placed into the $Posit array, starting at $Posit[0]

G51 X Y               ; Enable touch probe for X and Y
G1 X5.0 Y3.0 F30     ; Start motion toward probe
...
; $Posit[0] = X POS, $Posit[1] = Y POS

G51 X Y Z             ; Enable touch probe for X, Y and Z
G1 X5.0 Y3.0 F30.    ; Start motion toward probe
...
; $Posit[2] = X POS, $Posit[3] = Y POS, $Posit[4] = Z POS

PROBE 10 0 $Posit[0]    ; Over-write data in the $Posit array
G51 Z                 ; Enable touch probe for Z only
G1 X5.0 Y3.0 F30.    ; Start motion toward probe
...
; $Posit[0] = Z POS
```

6.4.49. PROGRAMDOWNLOADFILE Command



SYNTAX: PROGRAMDOWNLOADFILE <program name>

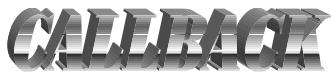
EXAMPLE: PROGRAMDOWNLOADFILE "C:\U600\Programs\Cycle.pgm"

Where: <program name> is the name of the program to download, which may contain format specifiers.

This command will read the program from the disk drive, compile the program if it needs compiled, and download it to the controller. If the CNC program that is downloaded contains a compiler error, a task fault will be generated.

All Callback commands will set the ErrCode task parameter, if an error occurs during execution.





6.4.50. PROGRAMEXECUTE Command

SYNTAX: PROGRAMEXECUTE <task index> <program name> [label] [mode]

EXAMPLE: PROGRAMEXECUTE 0, "C:\U600\Programs\Cycle.pgm", "StepCycle", 1

Where: <task index> is the task to change programs on.

<program name> is the name of the new program to execute.

[label] is an optional label to begin execution at.

[mode] is the optional execution mode of the program;

TASKEXEC_DEFAULT	-1 ; Execute default Mode (Run/Step)
TASKEXEC_RUN_INTO	0 ; Run into subroutines
TASKEXEC_STEP_INTO	1 ; Step into subroutines
TASKEXEC_STEP_OVER	2 ; Step over subroutines
TASKEXEC_RUN_OVER	3 ; Run over subroutines

This command will execute a program that is present on the controller. If a program is currently executing on the task, the program will be stopped. This command will execute a subroutine if the label parameter is a valid subroutine name. The mode of execution can also be specified.



Executing the PROGRAMEXECUTE command with the task index parameter equal to the current task will result in an error.



All Callback commands will set the ErrCode task parameter, if an error occurs during execution.



6.4.51. PROGRAMEXECUTEFILE Command

SYNTAX: PROGRAMEXECUTEFILE <task index> <program name>

EXAMPLE: PROGRAMEXECUTEFILE 0 "C:\U600\Programs\Cycle.pgm"

Where: <task index> is the task to change programs on.

<program name> is the name of the new program to execute, which may contain format specifiers.

This command will read the program from the disk drive and execute the program on the task specified. If a program is currently executing on the task, the current program will be stopped.

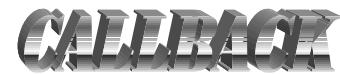
Executing the PROGRAMEXECUTEFILE command with the task index parameter equal to the current task will result in an error.



All Callback commands will set the ErrCode task parameter, if an error occurs during execution.



6.4.52. PROGRAMTASKRESET Command



SYNTAX: **PROGRAMTASKRESET** <task index> <abort> <reset> <deassociate> [<programToAssociate>]

EXAMPLE: **PROGRAMTASKRESET** 0, 1, 1, 1

Where: <task index> is the task to change programs on.

<abort> is a flag to determine if the task should be aborted.

<reset> is a flag to determine if the task should be reset.

<deassociate> is a flag to determine if the task should be deassociated.

<programToAssociate> is an optional parameter that may specify a CNC program to associate with the task.

This command will abort, reset, and/or deassociate the current program on the specified task, based on the flag parameters.

The last (optional) parameter may be used to associate a new CNC program. This parameter is a string and format specifiers may be used. The CNC program specified by the last parameter will be associated to the task after the task is reset.

Executing the PROGRAMTASKRESET command with the task index parameter equal to the current task will result in an error.



All Callback commands will set the ErrCode task parameter, if an error occurs during execution.



CALLBACK

6.4.53. PROGRAMUNLOAD Command

SYNTAX: PROGRAMUNLOAD <programHandle>

Where: <programHandle> is the name of the CNC program (no path specifier), which may contain format specifiers.

The ErrCode Task parameter will contain an Error Code if the command fails or you may test the return code as shown below in the example. This command may not be used to unload the CNC program it is executing from, you must do this from another task.



You may need to execute PROGRAMTASKRESET before unloading a CNC program.



All Callback commands will set the ErrCode task parameter, if an error occurs during execution.

EXAMPLE PROGRAM:

```
PROGRAMUNLOAD "TEST.PGM" ; unloads test.pgm
if ErrorCode != 0 then
    ; ...
end if

; or (user variable)

$ErrorCode = PROGRAMUNLOAD "TEST.PGM"
if $ErrorCode != 0 then
    ; ...
end if
```

6.4.54. PSO Card Based Commands

The PSO Card based commands execute on the optional PSO-PC card, typically configuring the card to generate a laser firing pulse. There are four types of PSOx commands:

PSOD – Distance, to specify the vectorial Distance between laser firing pulses (see Section 6.4.55 on page 6-97).

PSOF – Firing, enable/Disable the laser firing pulse and specify the axes to track (see Section 6.4.56 on page 6-100).

PSOP – Pulse, define the laser firing pulse (see Section 6.4.57. on page 6-102).

PSOS – Scaling, define/enable/disable scaling of axes (see Section 6.4.58. on page 6-105).

PSOT – Set/Clear analog/digital outputs or specify analog outputs to track axes position or velocity (see Section 6.4.59 on page 6-106).

All PSOx commands will wait until all axes in a previous motion command are “in-position”. This implies that all PSOx commands will wait, even if an axis is out of position due to instability of the axis or the INPOSLIMIT axis parameter set too low.



6.4.54.1. Configuring the PSO-PC Card to Fire a Laser

There are four basic steps to (typically) configuring the PSO-PC card to generate a laser firing pulse. There are however, many other ways.

1. Define the Laser firing pulse with one of the PSOP commands.
2. Specify the distance between laser pulses with the PSOD 0 command.
3. Scale the machine resolutions of the axes with the PSOS command, if required.
4. Define and enable the axes to track with the PSOF 3 command.

6.4.55. Position Synchronized Output Firing Distance Entry PSOD

SYNTAX: PSOD mode <*iExpression*>

The **PSOD** command specifies the number of machine steps to travel before output synchronization occurs on the laser output (LOUT1). Distances may be entered individually or sequentially through the use of variables. This command is only used in conjunction with the “**PSOF 3**” and **PSOP** commands.

6.4.55.1. Mode Argument for PSOD Command

SYNTAX: PSOD mode <*iExpression*>

The mode argument defines the ways to use the **PSOD** command. Currently, only the PSOD 0 command is supported.

6.4.55.2. Pulse Output at an Incremental Distance PSOD 0

SYNTAX: PSOD 0 <*iExpression*>

PSOD 0 indicates that the pulse output occurs at a fixed incremental distance, provided by *iExpression*, and must be less than 2^{23} machine steps. Refer to Table 6-11. An example illustrating this is shown in Figure 6-7. The distance is specified in machine steps.

EXAMPLE:

PSOP 0, 105	;Single pulse 10.5 ms wide
PSOD 0, 5	;Generate a pulse every 5 machine steps

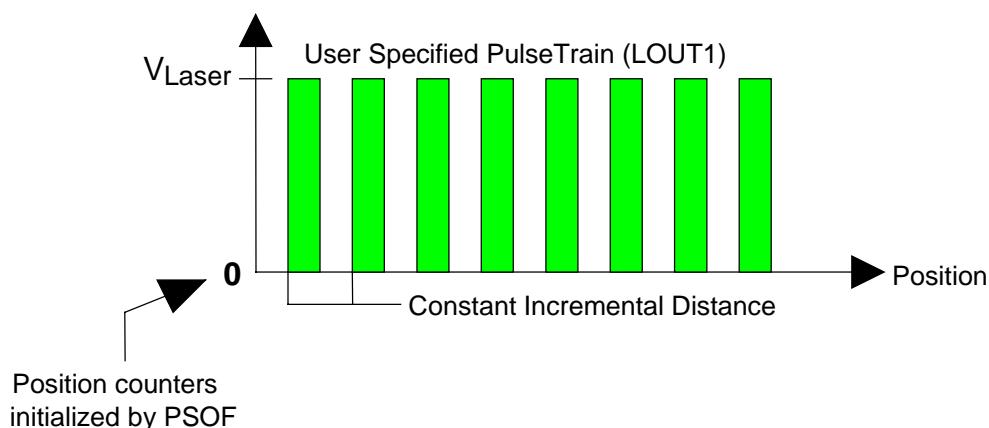
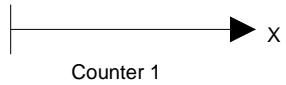
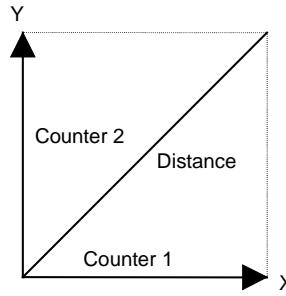
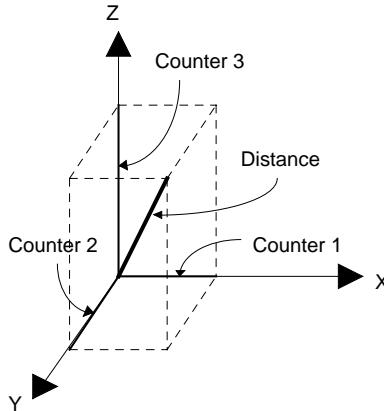


Figure 6-7. Trigger Pulse Fired at Constant Increments

Table 6-11. Distance Calculations for Multiple Axes Using the PSOD Command

Number of Axes	Distance Calculation	Diagram
1	Distance = Counter1	
2	Distance = $\sqrt{\text{Counter1}^2 + \text{Counter2}^2}$	
3	Distance = $\sqrt{\text{Counter1}^2 + \text{Counter2}^2 + \text{Counter3}^2}$	

6.4.55.3. Fire Equidistantly**PSOD 7****SYNTAX:** PSOD 7, <#pulses> <distance>

The PSOD 7 command will generate the specified number of firing pulses across the <distance> specified. The specified <distance> does not have to be a multiple of the <#pulses> specified. The fractional step count error will be accumulated during the tracking, causing no cumulative error. There will however be a machine count variation in the firing pulses, if the <#pulses> cannot be evenly divided into the <distance> specified. This allows the firing pulses to be evenly distributed (approximately) across the specified distance, without constantly changing the firing pulse distance to compensate for the desired non-integer firing distance.

At the completion of the specified <distance> the command will begin generating firing pulses again until disabled by the PSOF, 0 command. This command is most useful when generating firing pulses around the circumference of a rotary part.

EXAMPLE:

```
PSOP 0, 105      ; Single Pulse, 10.5 ms wide
PSOD,7,432,4000 ; Generate 432 firing pulses
PSOF, 3, 1, 2    ; Enable firing from Ch's 1 and 2
...              ; Generate motion
PSOF, 0          ; Disable firing
```

6.4.55.4. Offset Firing Pulse**PSOD 8****SYNTAX:** PSOD 8, <MachineSteps>

The PSOD 8 command will suspend tracking, or ignore the specified number of <MachineSteps>, causing the next firing pulse to be offset that distance.

EXAMPLE:

```
PSOD 8, 2000 ; Offset firing sequence by 2000 steps
```

6.4.56 Enable/Disable Position Synchronized Output Firing PSOF

The **PSOF** command activates or deactivates the tracking features and pulse train output on LOUT1.

SYNTAX: PSOF mode, iExpression

6.4.56.1. Mode Arguments for PSOF

The *mode* argument defines one of four possible ways to use the **PSOF** command. The arguments for the mode command can range in value from 0 to 3 and the following sections describe their meanings.

6.4.56.2. Disable Laser Output Pulse

PSOF 0

SYNTAX: PSOF 0

“PSOF 0” disables the output firing pulse train and tracking features. This is the default mode on power up of the PSO-PC.

6.4.56.3. Laser Output Fires Continuously

PSOF1

SYNTAX: PSQF 1

“PSOF 1” activates the output firing pulse train (as established by the **PSOP** command). The pulse train continues until it is disabled by the “PSOF 0” command. No position tracking occurs in this mode.



“PSOP 0” and “PSOP 4” may not be used with the “PSOF 1” command.

EXAMPLE:

PSOP 1, 50, 105, 75	:Single Pulse: :5ms lead, 10.5ms wide, :7.5ms tail
PSOF 1	:Fires the Pulse continuously

6.4.56.4. Fire Laser a Specified Number of Times PSOF 2

SYNTAX: PSOF 2 <*iExpression*>

“PSOF 2” activates the output firing pulse train (as established by the **PSOP** command) for *number*, number of times. If *iExpression*=0, then the output firing pulse train will not be activated until the previous output firing pulse train is complete. No position tracking occurs in this mode.

“PSOP 0” and “PSOP 4” may not be used with the “PSOF 2” command.



EXAMPLE:

PSOP 1, 50, 105, 75	:Single Pulse: ;5ms lead, 10.5ms wide ;7.5ms tail
PSOF 2, 525	;Fires that Pulse 525 times

6.4.56.5 Laser Output Synchronized with Position PSOF 3

SYNTAX: PSOF 3, <*ch_number*>[[,<*ch_number*>]][[,<*ch_number*>]]

“PSOF 3” activates the output firing pulse train (as established by the **PSOP** command).

The *ch_number* specifies up to three PSO-PC encoder channels to track. The user may specify up to three of the four channels. The channels are normally assigned to the first four axes in your system, with those axes named X, Y, Z and U by default.

The position counter will lock on to the motion of the axes attached to the specified channels. Up to three axes may be simultaneously tracked. Output firing occurs at distances established by the PSOD 0 command.

EXAMPLE:

PSOP 1, 50, 105, 75	:Single Pulse: ;5ms lead, 10.5ms wide, ;7.5ms tail
PSOD 0, 5	;Pulse every 5 Machine Steps
PSOF 3, 1, 2	;Track PSO-PC channels 1 & 2

6.4.57. Position Synchronized Output Pulse Configuration PSOP

The **PSOP** command configures the pulse output train.

SYNTAX: **PSOP mode, { width**

| lead, width, trail

| lead, width, trail, ramp, gap }

6.4.57.1. Mode Arguments for PSOP

The *mode* argument defines one of four possible ways to use the **PSOP** command. The arguments for the mode command can range in value from 0 through 2 and 4. The following sections describe their meaning.

6.4.57.2. Simple Single Pulse PSOP 0

SYNTAX: **PSOP 0, width**

“PSOP 0” defines the width *w* of the firing pulse to be generated, in tenths of milliseconds. This pulse is generated each time the defined firing condition occurs. Refer to Figure 6-8 and the **PSOD** command.

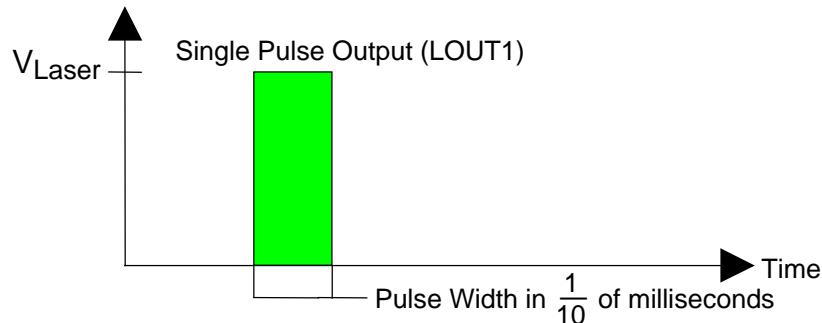


Figure 6-8. Single Pulse Generated on Firing Condition

EXAMPLE:

```
PSOP 0, 105          ;Single Pulse: 10.5 ms wide
PSOD 0, 5            ;Generate Pulse every 5 Machine Steps
PSOF 3, 1, 2         ;Track PSO-PC channels 1 & 2
```

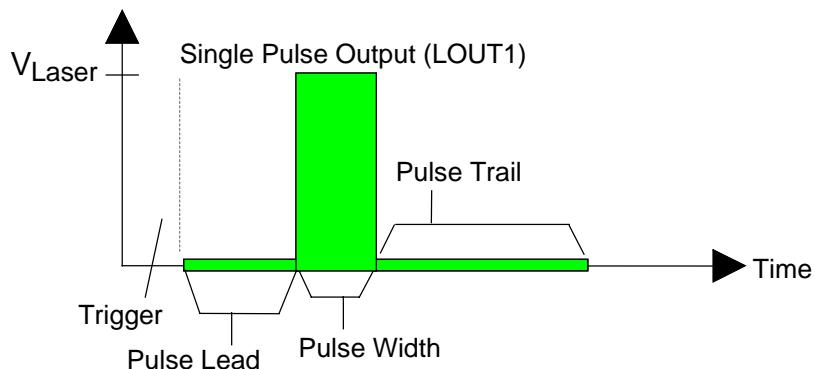
6.4.57.3. Single Pulse with Lead, Width and Trail PSOP 1

SYNTAX: PSOP 1, *lead*, *width*, *trail*

“PSOP 1” defines a single pulse to be generated, on the trigger condition, with definable pulse lead (*lead*), pulse width (*width*) and pulse trail (*trail*) characteristics. The pulse lead, width and trail arguments are specified in tenths of milliseconds and must be integer values. The pulse lead time is a delay after the firing trigger occurs before generating the pulse. The pulse trail time is a delay after the pulse is generated before tracking of the axes begins again.

EXAMPLE:

<pre>PSOP 1, 50, 105, 75 PSOD 0, 53 PSOF 3, 1, 2</pre>	<i>;Single Pulse:</i> <i>;5ms lead, 10.5ms wide</i> <i>;7.5ms trail</i> <i>;Fire Pulse every 53 Machine Steps</i> <i>;Track PSO-PC channels 1 & 2</i>
--------------------------------------------------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------



Lead, Width, & Trail are in $\frac{1}{10}$ of milliseconds

Figure 6-9. Single Pulse Output with Lead, Width, and Trail

6.4.57.4. Level based Laser Control

To toggle the laser firing output, on and off (level control), use the following syntax:

EXAMPLE:

<pre>PSOP, 1, 0, 10, 0 PSOF, 1 ; ; Do something ; PSOF, 0</pre>	<i>; Define lead, trail as 0</i> <i>; Set firing output on</i> <i>;</i> <i>;</i> <i>;</i> <i>; Set firing output off</i>
-----------------------------------------------------------------	-----------------------------------------------------------------------------------------------------------------------------------------

6.4.57.5. Simple One-shot Pulse **PSOP 4****SYNTAX:** **PSOP 4, width**

The PSOP 4 command defines the width of a single pulse to be generated, on the trigger condition, in microseconds (with a minimum value of 1 microsecond). Refer to the figure below.

EXAMPLE:

```
PSOP 4, 25          ; Single One-Shot Pulse 25 µs wide
PSOD 0, 7          ; Fire Pulse every 7 Machine Steps
PSOF 3, 1, 2        ; Track PSO-PC channels 1 & 2
```

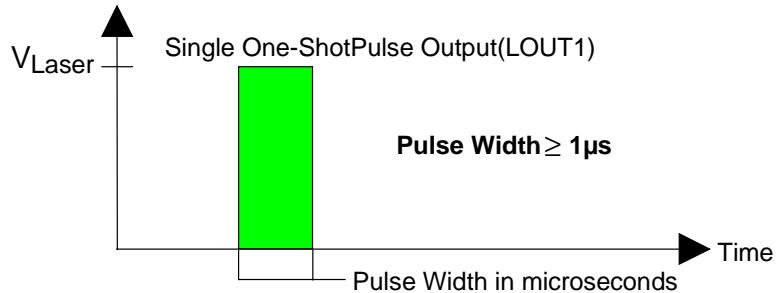


Figure 6-10. Single One-shot Pulse Output

6.4.58. Position Synchronized Output Scaling**PSOS**

The **PSOS** command allows the scaling of axes that have different machine resolutions allowing the PSO-PC card to track/fire on a true vectorial distance, velocity, etc.

SYNTAX: **PSOS mode**

Where mode is 0 through 2.

6.4.58.1. Disabling Scaling**PSOS 0**

SYNTAX: **PSOS 0**

“PSOS 0” disables scaling for all axes. This is the default mode on power up of the PSO-PC. This command does not undefine the scaling defined with the **PSOS 2** command. The **PSOS 1** command may be executed to enable scaling after the **PSOS 0** command.

6.4.58.2. Enable Scaling**PSOS 1**

SYNTAX: **PSOS 1**

“PSOS 1” enables scaling for all axes, previously defined with the **PSOS 2** command.

6.4.58.3. Define PSO Axes Scaling**PSOS 2**

SYNTAX: **PSOS 2, channel, scaling**

“PSOS 2” defines scaling for an axis, allowing axes which have different machine resolutions to be tracked by the PSO-PC as true vectorial distances, velocities, etc. The *channel* parameter specifies the input channel of the PSO-PC that you wish to scale. The *scaling* parameter, specifies the desired scaling of the specified axis, to match it to the resolution of the other axes. This scaling parameter may be a fraction that is less than or greater than zero. Scaling may be disabled with the PSOS 0 command and later re-enabled with the PSOS 1 command without redefining the axes scaling.

EXAMPLE:

```
PSOS, 2, 1, 1.5      ; SET CHANNEL 1 SCALING TO 1.5
PSOS, 2, 2, 0.5      ; SET CHANNEL 2 SCALING TO 0.5
PSOS, 1              ; ENABLE SCALING
PSOT, 4, 0, 0, 10, 100 ; ANALOG OUTPUT CHANNEL 0 PROPORTIONAL TO
                       ; VELOCITY. 0 TO 10 VOLTS OUTPUT OVER
                       ; 0 TO 100 CNTS/MSEC INPUT VELOCITY
PSOF, 3, 1, 2        ; TRACK ENCODER INPUT CHANNELS 1 AND 2
.
.
PSOS, 0              ; DISABLE SCALING (ANALOG OUTPUT STILL ACTIVE)
.
.
PSOS, 1              ; RE-ENABLE SCALING
```

6.4.59 Digital/Analog Output Command**PSOT**

The **PSOT** command is used to set/clear digital output lines on the PSO-PC card (OUT0 through OUT15) or set the voltage (-10.0 V to 10.0 V) of the analog outputs (AOUT1 and AOUT2).

SYNTAX:

```
PSOT mode{ bit#, state ... | states | dac#, voltage ... | dac#, v0, vmax, velocity  
| dac#, v0, vmax, position }
```

6.4.59.1. MODE Argument for PSOT

The *mode* argument defines one of four possible configurations (0, 2, 4, or 6) for setting digital and analog outputs. The following sections describe those modes.

6.4.59.2. Set Individual Output State**PSOT 0****SYNTAX:** **PSOT 0, bit#, state**

The PSOT 0 command sets individual digital output lines (argument *bit#*, specifying OUT0 through OUT15) to either a logic high (*state*=1) or logic low signal (*state*=0).

The “*bit#*” argument specifies an individual bit number that corresponds to one of the 16 digital outputs (0=OUT0, 1=OUT1, ..., 15=OUT15) on the PSO-PC card.

The “*state*” argument specifies the logic state (0=low, 1=high) to set. Multiple groups of bit numbers and states may be specified.



Programming an output bit to a logical “1” state results in the corresponding pin being pulled to ground. A programmed “0” state results in a high impedance state on the corresponding output pin. During reset, all outputs are in the high impedance state. All input lines are pulled to +5V by a 10KΩ resistor.

EXAMPLE:

PSOT 0, 4, 1	;Turns output #4 ON
PSOT 0, 4, 0	;Turns output #4 OFF
PSOT 0, 3,0, 13,1, 4,1, 10,0	;Spaces added for clarity ;Turns output #3 OFF, ;turns output #13 ON, ;turns output #4 ON, ;turns output #10 OFF, ;simultaneously.

6.4.59.3. Set Analog Outputs to Discrete Values PSOT 2

SYNTAX: PSOT 2, *dac#*, *voltage*

The “PSOT 2” command sets the output voltage of DAC0 (*dac#=0*) which is AOUT1 or DAC1 which is AOUT2 (*dac#=1*) to a constant value. Refer to Figure 6-11. DAC output voltages can range from -10.0 V to 10.0 V and have a resolution of 0.3 mV.

The “*dac#*” argument specifies which of the two digital-to-analog converters (0 or 1) to set. The value 0 corresponds to AOUT1 and the value 1 corresponds to AOUT 2. This argument is only used by commands “PSOT 2”, “PSOT 4” and “PSOT 6”.

The “*voltage*” argument defines the desired output voltage (-10.0 to 10.0 volts) of the specified DAC channel (0 or 1). Both DAC channels and their respective voltages may be specified in the same PSOT command (e.g., PSOT 2, 0, 5, 1, -5).

The analog outputs (AOUT1 and AOUT2) are via 16 bit D/A’s.



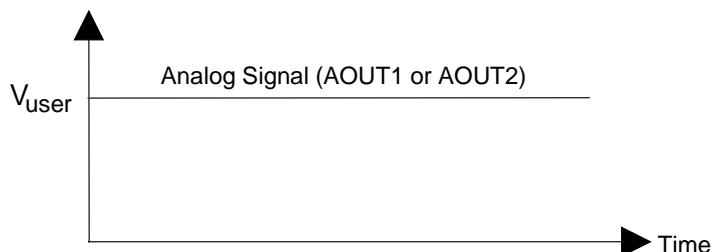
Note: Commas are required as delimiters between parameters, particularly when negative numbers are used, to prevent mathematical operations from occurring!



WARNING

EXAMPLE:

PSOT 2, 0, 5.25	;AOUT1 is 5.25 VDC
PSOT 2, 1, -3.70	;AOUT2 is -3.70 VDC



(Voltage between ± 10 VDC can be specified to the D/A)

Figure 6-11. User-Specified Analog Voltage

6.4.59.4. Velocity Ramping

PSOT 4

SYNTAX: PSOT 4, *dac#*, *vzero*, *vmax*, *velocity*

The PSOT 4 command sets the output voltage of DAC0 (*dac#=0*) which is AOUT1 or DAC1 which is AOUT2 (*dac#=1*) to a value that is proportional to the velocity (velocity ramping). Refer to the figure below. DAC output voltages can range from a programmable minimum value (at zero velocity) specified by argument *vzero* to a maximum voltage (at the target velocity, *velocity*) specified by argument *vmax*.

The “*dac#*” argument specifies which of the two digital-to-analog converters (0 or 1) the user wants to set. The value 0 corresponds to AOUT1 and the value 1 corresponds to AOUT 2.

The “*vzero*” argument specifies the zero-state analog output voltage for proportional output modes “PSOT 4” (*vzero* is the analog output voltage at zero velocity) and “PSOT 6” (*vzero* is the analog output voltage at the initial position). This argument can range from -10.0 volts to 10.0 volts.

The “*vmax*” argument specifies the maximum analog output voltage at target velocity, *velocity*, in mode “PSOT 4” or at the target position, *position*, in mode “PSOT 6”. This argument can range from -10.0 volts to 10.0 volts.



The target velocity is specified in machine steps per millisecond.

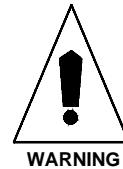


Keep in mind that the PSO-PC card operates in units of Machine Step/ms. A Machine Step is an integer. Likewise, the velocity is the same. Therefore, a 16 Machine Step/ms maximum velocity with a minimum voltage of 0 VDC and a maximum of 5 VDC causes the voltage to step at 0.3125 VDC increments ($[5 - 0]/16 = 0.3125$), even though a 16 bit D/A can step at 0.0003 VDC increments. So, moving at 1 Machine Step/ms is +0.3125 VDC; moving at 2 Machine Steps/ms is +0.6250 VDC, etc. For that reason, expect some variation at slow velocities and at velocities that are not integers when converted to Machine Steps/ms. This includes Velocity Errors when traveling at a constant speed. So, for this example the amount of variation will be the incremental voltage, which is 0.3125 VDC.



Velocity Tracking is activated by the “PSOF 3” command. Keep in mind, for the PSOF command to execute, it must have a pulse defined with the PSOP, or PSOD commands. The user must define a pulse even though it will not be generated.

Comma's are required as delimiters between parameters, particularly when negative numbers are used to prevent mathematical operations from occurring !



WARNING

EXAMPLE:

```

PSOP 0, 105          ; Single Pulse 10.5 ms ide
PSOD 0, 5            ; That Pulse every 5 Machine Steps
PSOT 4, 0, 0, 5, 16   ; Velocity Tracking
                      ;   using D/A #0 = AOUT1
                      ;   min voltage = 0 VDC
                      ;   max voltage = +5 VDC
                      ;   max velocity = 16 Mach Steps/ms
PSOF 3, 1, 2          ; Track PSO-PC channels 1 & 2

```

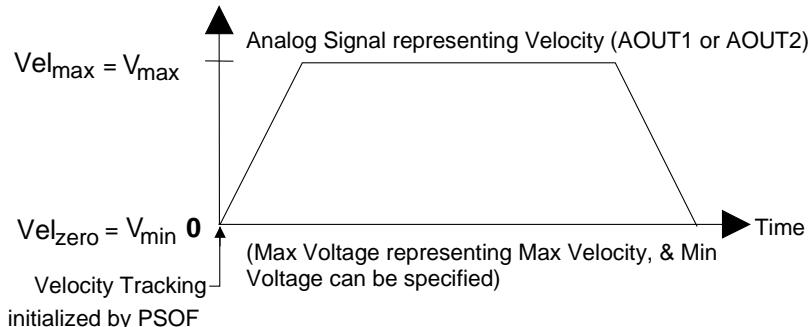


Figure 6-12. Velocity Ramping

6.4.59.5. Position Ramping**PSOT 6**

SYNTAX: **PSOT 6, dac#, vzero, vmax, position**

The PSOT 6 command sets the output voltage of DAC0 (*dac#=0*) which is AOUT1 or DAC1 which is AOUT2 (*dac#=1*) to a value that is proportional to the position (position ramping). Refer to Figure 6-13. DAC output voltages can range from a programmable minimum value (at initial position) specified by argument *vzero* to a maximum voltage (at target position, *position*) specified by argument *vmax*.

The “*dac#*” argument specifies which of the two digital-to-analog converters (0 or 1) to set. The value 0 corresponds to AOUT1 and the value 1 corresponds to AOUT 2.

The “*vzero*” argument specifies the zero-state analog output voltage for proportional output modes “PSOT 4” (*vzero* is the analog output voltage at zero velocity) and “PSOT 6” (*variable* is the analog output voltage at the initial position). This argument can range from -10.0 volts to 10.0 volts.

The “*vmax*” argument specifies the maximum analog output voltage at target velocity, *velocity*, in mode “PSOT 4” or at the target position, *position*, in mode “PSOT 6”. This argument can range from -10.0 volts to 10.0 volts.



Keep in mind that the PSO-PC card operates in units of Machine Steps for position. A Machine Step is an integer. Therefore, a 250 Machine Step target position with a minimum voltage of -1 VDC and a maximum of 8 VDC causes the voltage to step at 0.0360 VDC increments ($[8 - (-1)]/250 = 0.0360$), even though a 16 bit D/A can step at 0.0003 VDC increments. So, when at 1 Machine Step beyond the current position the voltage is -0.9640 VDC; when at 2 Machine Steps the voltage is -0.9280 VDC, etc. For this reason, expect some variation in the analog signal while stepping at 0.0360 VDC increments. For our example, the amount of variation will be the incremental voltage, which is 0.0360 VDC.



Position Ramping is activated by the “PSOF 3” command. Keep in mind, for the PSOF command to work it must have a pulse defined with the PSOP or PSOD commands. The user must define a pulse even though it will not be generated.



WARNING

Commas are required as delimiters between parameters, particularly when negative numbers are used, to prevent mathematical operations from occurring!

EXAMPLE:

```
PSOP 0, 105          ; Single Pulse 10.5 ms wide
PSOD 0, 5            ; That Pulse every 5 Machine Steps
PSOT 6, 0, -1, 8, 250 ; Position Ramping -
                      ;   using D/A #0 = AOUT1
                      ;   min voltage = -1 VDC
                      ;   max voltage = +8 VDC
                      ;   Target Position = 250 Machine Steps
PSOF 3, 1, 2          ; Track PSO-PC channels 1 & 2
```

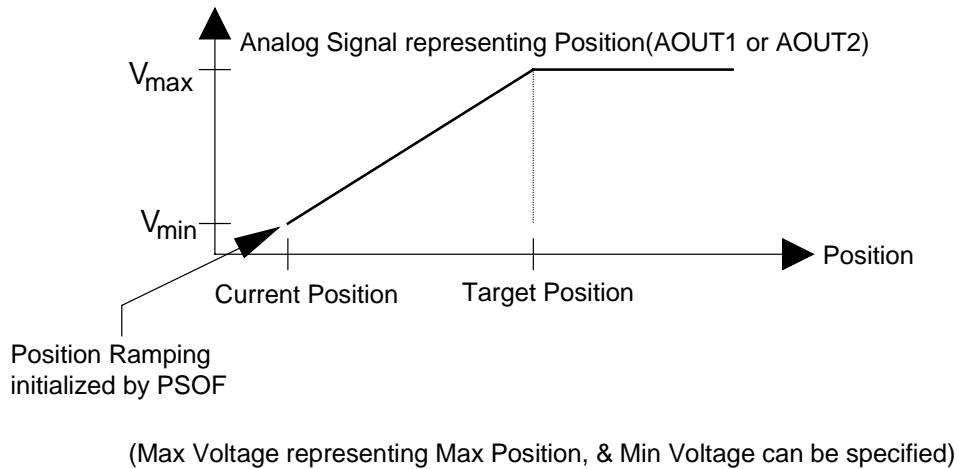


Figure 6-13. Position Ramping

6.4.59.6. PSOT 4 *velocity* Argument

The “*velocity*” argument defines the target velocity (in machine steps per millisecond) at which the analog output defined by *dac#* will be at its maximum (as defined by *vmax*). Velocity can range from $-2 \times 10E23$ to $2 \times (10E23 - 1)$ machine steps per millisecond. This argument is used only in mode “PSOT 4”.

6.4.59.7. PSOT 6 *position* Argument

The “*position*” argument defines the target position (in machine steps) at which the analog output defined in *dac#* will be at its maximum (as defined by *vmax*). *Position* can range from $-2 \times 10E23$ to $2 \times (10E23 - 1)$ machine steps. This argument is used only in mode “PSOT 6”.

6.4.60. Release Command

SYNTAX: RELEASE <axisMask>

EXAMPLE: RELEASE X Y

This command is the opposite of the **CAPTURE** command. It releases control of the axis from the current task. Refer to the **CAPTURE** command for more details.

6.4.61. Repeat Loop

REPEAT / RPT

SYNTAX: REPEAT </Expression>

<NCNBlock>

...

<NCNBlock>

ENDREPEAT

It is often desirable to have a group of CNC program blocks execute unconditionally a fixed number of times. A repeat loop is a simple way of performing such a task.

The **RPT/REPEAT** command designates that the group of program blocks executes multiple times. A parameter used in conjunction with this command specifies the number of times to execute this block.

Also, the user may nest **RPT/REPEAT** loops within each other (see the following example), limited only by available memory.

EXAMPLE PROGRAMS:

```
DVAR $MYVAR      ; Define a variable named MYVAR
RPT 10          ; The following program block will be ;repeated 10 times
    G1 G9 X0.1 F100.
ENDRPT          ; End of the repeat loop

$MYVAR = 5      ; Repeat counts may also be specified in variables
RPT $MYVAR      ; Enclosed blocks will be executed 5 times
    RPT 2        ; This inner loop will be executed twice each time the
                  ; outer loop is executed
        G1 G9 X-0.05 F100.
    ENDRPT       ; End of the inner repeat loop
ENDRPT          ; End of the outer repeat loop
```

6.4.62. Canned Function Overview

Canned functions provide more flexibility on how and when to call a subroutine. There are many uses of Canned Functions. Each of these uses is demonstrated in examples 1 through 4 below, respectively. However, regardless of how you would like to use canned functions, it is suggested that you review Calling a subroutine from another task first. Canned functions are defined like regular subroutines, with the DFS command. However, canned functions must also be “set” or “registered” via the SETCANNEDFUNCTION command. Calling, or Executing, canned functions. The Canned Function (subroutine) must be present on the UNIDEX 600 controller. It may be Auto Downloaded via the Program Automation Page. Canned functions will not execute if a task fault is present. You must use the ONGOSUB command, to customize actions during fault conditions.

6.4.62.1. SETCANNEDFUNCTION Command

SYNTAX: **SETCANNEDFUNCTION <id> <programHandle>, <programLabel>, <flags>**

This command defines a canned function, which may be called (or executed), several ways. Canned Functions may be disabled (see section 6.4.62.2.). See the Canned Function Overview for more information (see section 6.4.62.).

Where:

id - is the number of the canned function. Valid canned function id's may range from 1 to the upper limit defined by the NumCannedFunctions task parameter.

programHandle - is the name of the program containing the subroutine that the canned function calls. The current CNC program may be specified by indicating a null file handle, i.e.; “”, however, doing so will imply the current CNC program at the time the Canned Function is executed (see CannedFunctionID task parameter or EXECCANNEDFUNCTION)

programLabel - is the label of the subroutine that the canned function calls.

flags - are used to specify how the canned function operates.

0 - runs the Canned Cycle as an autonomous operation. Does not interrupt motion. (Default)

1 - follow current user mode. That is Auto Mode/Step Mode (Step Over/Step Into). This is used for debugging. The CNC program lines within the canned function (subroutine) will be displayed as they are executed.

2 - allows canned cycle to interrupt motion and return from the canned function via the canned functions specified return type.

4 - executes canned cycle at the end of a motion block.

6.4.62.2. Disabling Canned Functions

Canned Functions are disabled by specifying a NULL <program handle> and a NULL <program label>. The flags specified when disabling the canned function are insignificant.

```
SETCANNEDFUNCTION 1 "", "", 1
```

6.4.62.2.1. Calling a Subroutine from another Task

Download the following CNC program (CanSubs.Pgm), via the Program Automation page of the UNIDEX 600 MMI, selecting the Download Only option.

```
M02

DFS CANNED1
    MSGDISPLAY 0, "In Canned Cycle 1..."
    IF $P.DEFINED THEN
        MSGDISPLAY 5, "Waiting..."
        G4 F$P
        MSGCLEAR 5, ""
    ENDIF
    $BO1 = 1
    MSGDISPLAY 0, "Ending Canned Cycle 1..."
ENDDFS

DFS CANNED2
    MSGDISPLAY 0, "In Canned Cycle 2..."
    $BO2 = 1
    MSGDISPLAY 0, "Ending Canned Cycle 2..."
ENDDFS
```

Execute the following CNC program to register the subroutine, possibly via the Program Automation page of the UNIDEX 600 MMI, selecting the Auto Run Silent option.

```
SETCANNEDFUNCTION 1 "CANSUBS.PGM", "CANNED1", 0
SETCANNEDFUNCTION 2 "CANSUBS.PGM", "CANNED2", 0
```

Canned functions can now be executed with the following commands:

```
EXECCANNEDFUNCTION 1, 0      ; Execute Canned Function 1 (no parameters)
EXECCANNEDFUNCTION 1, 0, P5  ; Execute Canned Function 1 (with a parameter)
```

Or from any task as follows:

CannedFunctionID.1=1	;Execute Canned Function 1
CannedFunctionID.1=2	;Execute Canned Function 2

6.4.62.2.2. Calling Subroutines from the Manual I/O keys

Define and register your canned functions as described in Calling a subroutine form another task.

See the \U600\Ini\ManIO#.Ini file [Page2.Key11] for further documentation.

6.4.62.2.3. An easier to use ONGOSUB, or monitor Command

Here we define a CNC program to warn the user when a grinding wheel has worn too small.

Download the Following Program to define the subroutine (ToolStuff.Pgm). This may be done via the Program Automation page of the UNIDEX 600 MMI, selecting Download Only.

```
M02      ; This line avoids a compiler warning.

DFS TOOLCHANGE
; GLOBAL0 holds the wheel diameter
  MSGDISPLAY 0, "Warning: tool diameter too small:" $GLOBAL0, "Change tool."
  MSGDISPLAY 0, "Install new tool, enter tool diameter"
  $GLOBAL0 = MSGINPUT "Wheel Diameter;Enter new wheel diameter (inches);"
ENDDFS
```

Execute the following program to register the CNC subroutine. This maybe done via the Program Automation page of the UNIDEX 600 MMI, selecting Auto Run Silent.

```
SETCANNEDFUNCTION 10 "ToolStuff.Pgm", "TOOLCHANGE", 0
ON( $GLOBAL0 < .5 ) SET CannedFunctionID 10 0 ONSET_MODE_LATCH
M02
```

When \$GLOBAL0 changes to less than .5, the ON monitor automatically sets the CannedFunctionID task parameter to 10, calling the TOOLCHANGE subroutine.

In this example, it is important that the TOOLCHANGE subroutine set \$GLOBAL0 to a value higher than .5, to prevent the routine from executing continuously.



6.4.62.2.4. Implementing Canned Cycles

Download the following Program to define the subroutine (Drill.Pgm) that will be called as a canned function. This may be done with the Program Automation page of the UNIDEX 600 MMI, Selecting the Download Only option.

```
M02          ; This line avoids a compiler warning.  
DFS DRILL  
      M2000      ; Activates output to lower the drill  
ENDDFS
```

Execute the following CNC command to register the canned function (this can be done automatically, by putting the command line in a CNC program, and adding it to the Program Automation Page)

```
SETCANNEDFUNCTION      10      "DRILL.PGM",      "DRILL",  
AER_CANNEDFUNCTION_ENDOFBLOCK
```

The DRILL subroutine is then called automatically, at the end of every G0 / G1 / G2 / G3 motion command.

6.4.63. Return from Subroutine/Program**RETURN****SYNTAX: RETURN**

The RETURN statement is used to exit subroutines or programs. After a RETURN is executed, program execution will continue on the line immediately after the CLS, or FARCALL command that was executed. If you execute a RETURN in a “top-level” program (one which was not called from another program) the program will stop execution (as if an M02 were executed). If a task reaches the end of a CNC program without executing a return or other program control execution code, such as an M02, it will implicitly execute a return type of RETURNTYPE_NULL.

6.4.63.1. RETURN from an ONGOSUB Command**SYNTAX: RETURN [[<fExpression>]]**

RETURNTYPE_NULL,	= 0	(Default)
RETURNTYPE_START,	= 1	
RETURNTYPE_INTERRUPT,	= 2	
RETURNTYPE_END,	= 3	
RETURNTYPE_OFFSET,		= 4

The RETURN statement is used to exit subroutines (or programs), or in this topic, to return from an ONGOSUB command. Since ONGOSUB’s occur asynchronous to program execution (they can occur in the middle of a CNC line, during a command) the task must know how to handle the line it is executing when it is interrupted. It can either re-execute the line or skip it. The user must specify which is desired for their application. A further complication arises when the interrupted line was executing synchronous motion, such as a G1 command. The controller may be instructed to perform motion adjustments in order to complete the move properly. Typically, return type RETURNTYPE_INTERRUPT or RETURNTYPE_OFFSET would be used to maintain programmed position when a move is interrupted. After axes motion has been interrupted, you may move the axes via any valid Immediate Mode command, within the ONGOSUB subroutine.

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. Any synchronous 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 any interrupted synchronous CNC command 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

The interrupted line is aborted. 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.

6.4.64. SetParm Command

SYNTAX: SETPARM <AxisList> <AxisParameter> <Value>

This command changes the current value of any axis parameter. All axis parameters may be modified, similarly as they may be modified sing the AerDebug utility program.

The <AxisList> parameter is a list of axes to which this command applies. This list may contain any number of axes, but all axes mentioned must be associated with the current task.

The <AxisParameter> parameter to this command is the name of the axis parameter whose value is to be modified.

The <Value> parameter is the value to which this parameter is to be set. This value is an integer and may be specified using a numeric literal, a variable or a simple expression. This value will be applied to all axes specified.

EXAMPLE:

```
SETPARM X IMAX 10000      ;; Set IMAX parameter for the X axis to 10000
SETPARM X Y Z DRIVE VAR1 ;; Set DRIVE parameter for the X, Y and Z axes equal to
                           VAR1
```

6.4.65. Slew Command

SLEW

SYNTAX: **SLEW** *port* <x_axis> <speed> [<y_axis> <speed>]

Where:

<port> - is either the serial port (U630\631) or the joystick port (UNIDEX 600\650).
 <x_axis> - is the axis to be commanded by the x axis movement of the slew device.
 <speed> - is the maximum speed in user units/minute.
 <y_axis> - is the optional second axis to be moved by the y axis motion of the slew device.

The **SLEW** command allows the user to position the axes manually through the use of a RS-232 serial trackball or mouse (UNIDEX 630\631) or an analog joystick (UNIDEX 600\650). On the UNIDEX 630\631, the port number represents the serial port on the controller board that connects to the trackball and/or mouse (0-3).

The mouse and trackball must be connected to the desired serial port on the (UNIDEX 630\631) controller board when its firmware loads (on power up).



On the UNIDEX 600, the port refers to the connection of the joystick. Port 0 refers to the joystick port on the UNIDEX 600 board, 1 to 3 refers to the joystick port on the optional encoder expansion boards 1 to 3. In all cases, the joystick connects to the joystick connector on the DR500, BB500 or BB501 that interfaces the user to the UNIDEX 600 or encoder expansion board.



Speed is specified in user units/minute.

The *speed* is the maximum velocity commanded by the input device in user units. The *x_axis* parameter is the axis designator for the axis commanded by the movement of the X axis of the slew device. The *y_axis* parameter, optionally specifies the axis designator commanded by the movement of the Y axis of the slew device. The SLEW command does not complete (will continue executing) until the slew mode is disabled. To disable the slew mode, press the fire button on the joystick, or press the left button on the mouse or trackball.

EXAMPLE PROGRAM:

SLEW 0 X 1000 Y 500 ; Activates the joystick through the UNIDEX 600 card

6.4.66. Start Motion (STRM) Command

STRM

SYNTAX: **STRM** <*axisLetter*> <*direction*> <*speed*>

axisLetter is the axis

direction is the free run direction (1 is positive, -1 negative)

speed is the speed to free run at in user units per minute (or rpm for rotational axes)

This asynchronous motion command begins an axis free running, in the specified direction. The axis will begin motion with no target, stopping only when an ENDM command is executed.

The move is an asynchronous motion command so program execution resumes immediately after the move starts without waiting for the move to end.

The **STRM** command uses the same acceleration and deceleration axis parameters as the G0 command. It does not use the accel/decel task parameters, like the G1 command.

Use the ENDM command to end motion on an axis that executed a **STRM** command.

6.4.67. String Functions

SYNTAX : <stringExpression> is <stringFunction> (arguments)

EXAMPLES:

\$GLOB0 = STRLEN(\$STRGLOB0)	; find the length of a string
\$GLOB0 = STRCMP(\$STRGLOB0, "end", 0)	; compare the length of two strings
\$GLOB0 = STRFIND(\$STRGLOB0, "=", 0)	; look for a string within another string
\$GLOB0 = STRCHAR(\$STRGLOB0, "1234567890")	; look for a set of char's within another string
\$GLOB0 = STRTODBL(\$STRGLOB0, 4)	; convert a string to a number
\$GLOB0 = STRTOASCII(\$STRGLOB0)	; find the ASCII value of a character
\$STRGLOB0 = STRMID(\$STRGLOB0, 1, 5)	; remove a string from a larger string
\$STRGLOB0 = STRUPR(\$STRGLOB0)	; convert a string to upper-case
\$STRGLOB0 = STRLWR(\$STRGLOB0)	; convert a string to lower-case
\$STRGLOB0 = DBLTOSTR(\$GLOB0, 4)	; convert a number to a string

6.4.67.1. STRLEN

SYNTAX : <fVariable> = **STRLEN**(<stringExpression>)

EXAMPLE: \$GLOB0 = **STRLEN**(\$STRGLOB0) ; find the length of a string

Returns the length (number of characters) in the string as a numeric value. The function takes one argument (and that argument must be a string).

For example:

STRLEN("cat") is 3

6.4.67.2. STRCMP

SYNTAX : <fVariable> = **STRCMP**(<stringExpression>, <stringExpression>, <bCaseSensitive>)

EXAMPLE: \$GLOB0 = **STRCMP**(\$STRGLOB0, "end", 0) ;compare the length of two strings

Compares two strings and returns 0 if the strings are the same and 1 if the strings differ. This function takes three arguments. The first two arguments are the two strings to compare. The third argument indicates case sensitivity. If the third argument is 0, the comparison is case insensitive (when comparison is case insensitive, "DOG" is the same as "dog"). If the third argument is 1, the comparison is case sensitive.

For example:

STRCMP("cat", "Cat", 1) is 1

STRCMP("cat", "Cat", 0) is 0

6.4.67.3. STRFIND

SYNTAX: <fVariable> = **STRFIND**(<stringExpression>, <stringExpression>, <bCaseSensitive>)

EXAMPLE: \$GLOB0 = **STRFIND**(\$STRGLOB0, "=", 0) ;looks for a string within another string

Looks for occurrence of the second string inside the first string. If it does not find the second string in the first string, it returns -1. If the second string is found in the first string it returns the index of the character in the first string, where the first occurrence of the second string is found. Indices returned are zero based. The function requires three arguments. The first argument is the string to look in and the second argument is the string to look for. If the second string is null (equal to "") then STRFIND always returns 0. The third argument indicates case sensitivity. A zero in the third argument means the comparison is case insensitive. If the third argument is 1, the comparison is case sensitive.

For example:

STRFIND("alazydogjumped", "DOG",0) will return the value 5.

6.4.67.4. STRCHAR

SYNTAX : <fVariable> = **STRCHAR**(<stringExpression>, <stringExpression>)

EXAMPLE: \$GLOB0 = **STRCHAR**(\$STRGLOB0, "1234567890") ; look
for a set of char's within another string

Looks inside the first string for occurrence of any character in the second string. If the function does not find any character of the second string in the first string, it returns -1. If characters from the second string are found in the first string, it returns the index of the character in the first string, where the first occurrence of a character in the second string is found. Indices returned are zero based and the comparison is always case sensitive. This function takes two arguments. The first argument is the string to look in, and the second argument is the string to look for.

For example:

STRCHAR("POWER=5.67", "0123456789.-+") will return the value 6.

6.4.67.5. STRTODBL

SYNTAX : <fVariable> = **STRTODBL**(<stringExpression>,)

EXAMPLE: \$GLOB0 = **STRTODBL**(\$STRGLOB0) ;convert a string to a number

Converts a string to a double value. If the string is not a valid double, it returns 0. Trailing invalid characters are ignored.

For example:

STRTODBL("7.889inches") is 7.889

STRTODBL("length=7.889") is 0.

6.4.67.6. STRTOASCII

SYNTAX : <fVariable> = **STRFIND**(<stringExpression>, <index>)

EXAMPLE: \$GLOB0 = **STRFIND**(\$STRGLOB0, 0) ; find the ASCII value of a character

Converts a given character of a string to its ASCII value. This function takes two arguments. The first argument is the string, and the second argument is the index of the character to translate (index is zero based). If the index specifies a character outside of the string, 0 is returned.

For example:

STRTOASCII("cat", 1) is 97

STRTOASCII("cat", 44) is 0.

6.4.67.7. STRUPR

SYNTAX : <fVariable> = **STRFIND**(<stringExpression>)

EXAMPLE: \$STRGLOB0 = **STRFIND**(\$STRGLOB0) ; convert a string to upper-case

Converts the passed string to upper case. Non alphabetic characters are ignored. The function takes one argument (the string to be converted).

For example:

STRUPR("5.6 Inches") is "5.6 INCHES".

6.4.67.8. STRLWR

SYNTAX : <fVariable> = **STRLWR**(<stringExpression>)

EXAMPLE: \$STRGLOB0 = **STRLWR**(\$STRGLOB0) ; convert a string to lower-case

Converts the passed string to lower case. This expression takes one argument (the string to be converted). Non alphabetic characters are ignored.

For example:

STRLWR("5.6 InchesPerSecond") is "5.6 inchespersecond".

6.4.67.9. DBLTOSTR

SYNTAX : <fVariable> = DBLTOSTR(<stringExpression>, <NumDigits>)

EXAMPLE: \$STRGLOB0 = DBLTOSTR(\$GLOB0, 4) ;convert a number to a string

Converts a numeric value into an ASCII string. This function takes two arguments. The first argument is the value to convert, and the second argument is the number of digits past the decimal point to convert. If the second argument is too large or less than zero, it will fill out the size of the string with trailing zeros, if necessary.

For example:

`DBLTOSTR(12.3456789, 3)` is “12.345”

`DBLTOSTR(12.3456789, 0)` is “12”

6.4.67.10. STRMID

SYNTAX : <fVariable> = **STRMID**(<stringExpression>, <StartChar>, <EndChar>)

EXAMPLE: \$STRGLOBO = STRMID(\$STRGLOBO, 1, 5) ; remove a string from a larger string

Selects a sub-string of characters from a string. This expression takes three arguments. The first argument is the string to select from. The second and third arguments are the starting and ending indices (the smaller of the two will be the starting index). Indices are zero based, and the string will include the characters specified by the two indices. If one index is -1, it will select characters all the way to the end of the string.

For example:

STRMID("POW=7.6 watts, 4, 6) is "7.6"

STRMID("POW=7.6 watts, 4, -1) is "7.6 watts"

STRMID("POW=7.6 watts, 4, 4) is "7"

6.4.68. SYNC Command

SYNC

SYNTAX: **SYNC <axisLetter><table><mode>**

Where:

axisLetter is the slave axis.

table is the table number (table numbers can range from 0 to 99).

mode is the SYNC mode (see below).



See the Camming Overview (see section 6.4.35.) and the Camming Performance Tip (section 6.4.35.3) for more information.

This statement synchronizes a master axis to a slave. The synchronization will not occur until all motion (if any is in progress) is complete on the slave. The user is cautioned that electronic camming behavior can be complex and the following description along with the description of the related parameters must be fully understood to produce the desired results.

After synchronizing a slave to a master axis, the slave's motion follows the master axis' motion. After unsynchronizing a slave, its motion no longer follows the master axis. The user must provide a synchronization mode as a parameter in the SYNC statement. The SYNC mode may be 0, 1, 2, or 3. SYNC mode 0 removes synchronization of a slave axis from a cam table. SYNC modes 1, 2, and 3 synchronize the axis to the cam table. To select the correct mode for synchronizing, the programmer must be cognizant of a number of complex details of the electronic camming process described below.

Mode 1 enables relative synchronization and mode 2 enables absolute synchronization. The incremental mode does not check the position of the slave relative to the master when the cam table is engaged. All output motion occurs relative to the current position. The absolute mode forces the slave to move to the position indicated by the current master position by generating an "infeed" command. Using the following as an example; suppose that at the moment of synchronization, the master is currently at position "m" and the slave at position "x". Furthermore, suppose that the table specifies that at master position "m" the slave should be at position "s". Mode 2 directs the slave to move from position "x" to position "s" as the table synchronizes. This infeed move in mode 2 initiates simultaneously with the initialization of the electronic camming move. Therefore, immediately after synchronizing the axes, the total motion is the sum of the motion directed by electronic camming and the motion directed by SYNC mode 2. At a later point in time, after the infeed move has completed, the total motion will be determined by the electronic camming. The infeed move is performed at the speed specified by the SYNCSPED axis parameter and is subject to normal acceleration and deceleration as dictated by the ACCEL and DECEL axis parameters. SYNC mode 1 does not cause any automatic movement of the slave. Instead, the table is interpreted, so that at master position "m", the slave is assumed to be at the correct position "s". All of the following points in the table will be offset accordingly (x-s is effectively added to all slave position table coordinates).

The program can synchronize to a new cam table while the slave is already synchronized to a current table (without removing synchronization in between with SYNC mode 0's). However, SYNC modes 1 and 2 provide no protection for any jump in slave commanded positions that may be caused by the potentially large difference in values between the two tables. The tables switch instantaneously without any automatic acceleration or deceleration.

Do not remove synchronization from an axis in motion, since the slave will stop abruptly. Insure that the slave axis is not moving, such as via the ENDM command.

SYNC mode 3 is velocity mode. In SYNC mode 3 the slave values are interpreted as axis velocities, not axis positions. The table controls slave velocity based on a master axis position. The axis velocity is specified in Units/Millisecond.

All cam tables are circular in operation. When the master axis exceeds the last point in the table, synchronization will resume with the first cam table point. Therefore it is important to note that if the master axis position is ever to exceed the bounds of the cam table when synchronization is active then the first and last slave positions in the table should be identical. This will prevent abrupt changes in the slave output position as the table transitions from the last to first point (or first to last depending on the direction of the master).

Care should be taken when engaging cam table motion as it's incorrect application may result in abrupt changes in slave velocity. Make sure that the master axis and slave axes are stationary when engaging or disengaging cam table motion, or use MAXCAMACCEL (SYNC mode 3 only).

If you repeatedly sync and desync, you may have to set MASTERPOS before syncing to avoid 32-bit overruns and result jerks of the slave.

**EXAMPLE:**

```
SYNC Y 3 2      ; Synchronizes the table  
SYNC Y 3 0      ; Desynchronizes the table
```

The following axis parameters are relevant to camming (master/slave motion):

MAXCAMACCEL - Mode 3 offers acceleration/deceleration protection used when synchronizing axes on the fly (without desynchronizing in between with SYNC mode 0's). If this parameter is non-zero, the slave axis will not exceed this acceleration while electronically camming. Modes 1 and 2 do not use this parameter. To deactivate this feature, set the parameter value to "0".

CAMOFFSET - This parameter is added to the master position before doing the table lookup. For example, if the table covers master positions from 0 to 360 degrees the actual master position is 2 degrees and CAMOFFSET is 3 degrees, then 5 degrees will be used as the master position to index into the table. Note that this parameter is set in encoder counts and not user units.

MASTERLEN - Specified for rotary master axes. Setting the MASTERLEN equal to the number of counts per revolution of the master axis will modulo the MASTERPOS. As an example, if the MASTERLEN parameter is set to 1000, then the masterpositon would have the following sequence of positions: ...997,998,999,0,1.... Note that the slave entries at the beginning of the table and the end of the table should be continuous to avoid abrupt changes in the slave output position.

EXAMPLE:

SYNC Y 3 2 ; Synchronizes the table
SYNC Y 3 0 ; Desynchronizes the table

6.4.69. Track Command

SYNTAX: TRACK~<axisLetter>~<mstr_start>~<mstr_dist>~<slave_dist>

Where :

The mstr and slave parameters are all fExpressions.

<axisLetter> specifies the slave axis that is to begin tracking its master's position. The axis must not be moving when this command is executed (the master axis can be moving however). The master axis provided in a prior CFGMASTER command.

<mstr_start> specifies the starting master position 'Dm0' to begin tracking.

<mstr_dist> specifies the distance ' ΔD_m ' over which the master will travel as the slave accelerates from zero speed to the desired tracking ratio.

<slave_dist> specifies the distance ' ΔD_s ' over which the slave will travel as it accelerates from zero speed to the desired tracking ratio.



See the Camming Performance Tip (section 6.4.35.3) for more information.

The master starting position, and the two distances must be entered in user units.

This command establishes a master-slave relationship, where the slave axis is accelerated from zero speed up to a speed which is a user defined fixed ratio of the master axis speed. The time at which the acceleration begins, the final speed of the slave, and the fixed speed ratio between master and slave are all determined by the parameters provided in the command, as described below.

Tracking motion of the slave axis begins after the master axis begins moving, (via any other motion), and arrives at the position specified in the <mstr_start> parameter. Tracking is active until an ENDM is executed on the slave axis (See the example below).

Before using this command, the user must have used the CFGMASTER command to establish the identity of the master axis. The master axis must be traveling at a constant speed throughout the acceleration, however, if at a later time the master axis changes speed, the slave will adjust so as to maintain the specified ratio. The time at which the acceleration begins, the final speed of the slave, and the fixed ratio are all determined by the parameters specified in the TRACK command, as described below.

Prior to using this command, the master must be moving at a non-zero constant velocity (V_m), and the slave must stationary. Upon execution of this command, the slave begins accelerating from zero speed when the master position has exceeded D_{m0} . If the master position has already exceeded D_{m0} then acceleration will begin immediately. The acceleration will produce a linearly increasing velocity. At some later time (T_1), the slave will reach its desired speed (V_s), and it will stop accelerating. During this time interval ΔT (from T_0 to T_1) the master will travel a distance $D_{m1}-D_{m0}$ (or ΔD_p), and the slave will travel a distance of ΔD_s .

The diagram below (Figure 6-14) portrays positive values for ΔD_m and ΔD_s . However, either parameter may be negative. The sign of these parameters indicates the sign of the slave velocity relative to the master's velocity. A negative sign indicates a negative velocity for the respective axis.

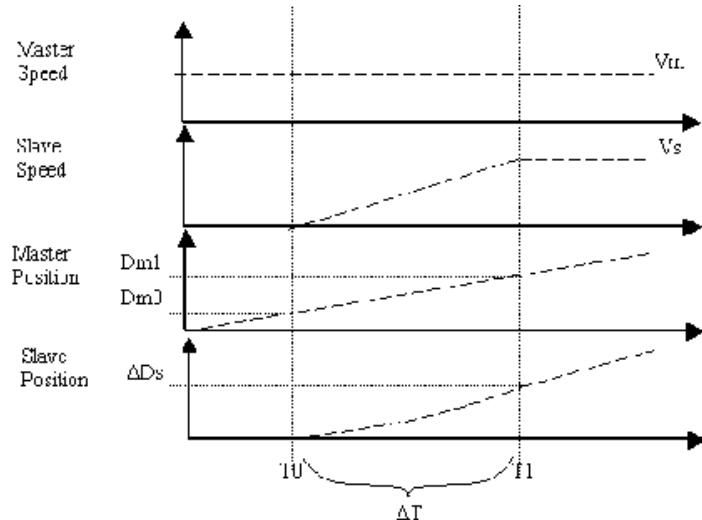


Figure 6-14. Track Command Δ Diagram

The user might want to specify the TRACK command with different values for ΔD_m and ΔD_s . For example, the user might want to specify (V_s/V_m) (the gear ratio) directly. The user needs to only start with D_m and any two of the other four initial parameters: V_s , ΔT , ΔD_m and ΔD_s . The equations below (derived using the simple law: $V = \int D dt$): can be of assistance in deriving the proper parameters to use with the TRACK command:

$$\Delta D_s = \Delta T * V_m$$

$$\Delta D_m = 2 * \Delta D_s * (V_m / V_s) = 2 * \Delta T * V_m * (V_m / V_s)$$

Note, that in the above equations either V_m , or the speed ratio (V_m/V_s) must be known beforehand. In the common simple case where $V_m = V_s$ (master and slave travel at same speed), the above simplifies to :

$$\Delta D_s = \Delta T * V_m$$

$$\Delta D_m = 2 * \Delta D_s$$

While the track command is active, the user may also “add” additional motion on top of the slave with other asynchronous motion commands (such as STRM, and INDEX) independently of the track relationship. The TRACK command is terminated by an ENDM command, after which the slave will decelerate to zero speed.

EXAMPLE PROGRAM:

HOME X Y	; Start X and Y at zero
CFGMASTER X 0 Y	; X is slave, Y is master
TRACK X 1.0 2 4	; Slave starts when master is 1, slave at full speed ; when master=1+2=3, and slave=4
G90 F1000	
G1 Y5	; Starts master moving, slave will track.
ENDM X	; end tracking
G1 Y5	; Starts master moving, slave does not track.

6.4.70. VOLCOMP Command



SYNTAX: **VOLCOMP** <TempScale>, <PresScale>, <HumidScale>, <TempOffset>, <PresOffset>, <HumidOffset>, <Temp>, <Pressure>, <Humidity>

EXAMPLE: \$var = **VOLCOMP** 12.004, 72.536, 1.0, 15.222, 535.74, 0, -1, 0, 50
; calculate the correction factor

This command will calculate the correction factor for the wavelength of light based upon the actual temperature, pressure and humidity. It is typically used with Aerotech's LZR1100 environmental compensator. This correction factor may then be used via the G151 command to update the scaling of the axes.

The <TempScale>, <PresScale>, <HumidScale>, <TempOffset>, <PresOffset> and <HumidOffset> parameters are used in the calculation as follows:

Pressure Value = Sensor Value * PresScale + PresOffset

The <Temp>, <Pressure>, and <Humidity> may be specified as an analog input to read the sensor from, or a value may be specified. Temperature is in units of degree Celsius. Pressure is in absolute millimeters of mercury. Humidity is specified as a percentage, 50 represents 50%.

Analog inputs are specified as values 0 through -15, where -2 would represent analog input 2. See the first half of the example below.

Values may be directly specified, as a value greater than zero, through the maximum range. If a value is specified directly, the scale and offset parameters (6 total) must be specified as 1. They will not be used in the calculation. See the second half of the example below.

EXAMPLE PROGRAM:

DVAR \$CorFactr

```
; Specify :
; Temp. Scale=12.004, Pressure Scale=72.536, Humidity Scale=1.0
; Temp. Offset= 15.222, Pressure Offset= 535.74, Humidity Offset= 0
; Read temperature from analog input 1, pressure from input 0 and humidity equals 50(%)
$CorFactr = VOLCOMP 12.004, 72.536, 1.0, 15.222, 535.74, 0, -1, 0, 50
G151 F(1 / $CorFactr) ; update scaling
G1 X3.5 F100. ; move

; Specify temperature as 20C, pressure as 29.8 mm Hg and humidity as 60(%)
$CorFactr = VOLCOMP 1, 1, 1, 1, 1, 1, 20, 29.8, 60 ; calculate the correction factor
G151 F(1 / $CorFactr) ; update scaling
G1 X3.5 F100. ; move
```

6.4.71. Wait Command**WAIT**

SYNTAX: **WAIT** <conditionalExpression> < iExpression > <TimeToWait>
 [<Flag>]

The **WAIT** command will wait until the specified condition is true, or the command times out. The program will pause execution on that line, until either the condition is true, or it times out. If the command times out, the controller generates a TaskFault. The “<iExpression>” is the time-out parameter in milliseconds. If the time specified as -1, then the command will never time out, it will wait indefinitely. The conditional expression is evaluated at the rate indicated by the AvgPollTimeSec task parameter. However, if a non-zero value is specified for the <Flag> parameter, then instead of generating a TaskFault when timing out, it instead sets the ErrCode task parameter to the hexadecimal value E0082006.

EXAMPLES:

```
WAIT ( $GLOBAL1 == 1) 10000      ; A TaskFault is generated on a time out
WAIT ( $GLOBAL1 == 1) $GLOB0 1    ; The ErrCode task parameter is set on a time
out
WAIT ( $GLOBAL1 == 1) -1         ; Will wait forever ( never time out)
```

6.4.72. Conditional Looping**WHILE / WHL**

SYNTAX: **WHILE** <conditionalExpression> [[**DO**]]

<CNCBlock>

...

<CNCBlock>

ENDWHILE

The ellipses ‘...’ above indicate a series of zero or more CNC program lines.

It is often desirable to have a group of program blocks execute repeatedly until a specific condition becomes true. The **WHILE-ENDWHILE** construct, provided by the UNIDEX 600 Series controller’s programming language permits easy implementation of such a function. The “**DO**” keyword is optional.

When the **WHILE** command is executed, the controller evaluates the specified conditional expression. If the conditional expression evaluates true (not equal to 0.0), the following CNC program blocks up to the **ENDWHILE** command are executed. The conditional expression is then re-evaluated. This repeats until the condition is false (zero). At this time, program flow proceeds to the command immediately following the **ENDWHILE** keyword. If the condition evaluates false on the first evaluation, the CNC program lines within the **WHILE** construct will never be executed.

EXAMPLE PROGRAM:

```
DVAR $OUTER $INNER      ; Define variables OUTER and INNER
$OUTER = 0               ; Initialize outer loop counter to zero
WHILE $OUTER LT 10 DO   ; Execute this loop until the variable OUTER ;is greater
; than or equal to 10
    $OUTER = $OUTER + 1 ; Increment the loop counter
ENDWHILE                 ; End of while construct
WHILE $OUTER GE 1 DO    ; Execute this loop until the variable OUTER ;is less
; than or equal to 0
    $INNER = 0          ; Initialize inner loop counter
    WHILE $INNER LE 5 DO ; While loops can be nested inside of one ;another
        $INNER = $INNER + 1

        IF $INNER EQ 5 THEN ; Even conditional commands can be in while loops
            $OUTER = $OUTER -1

        ENDIF

    ENDWHILE

ENDWHILE
```

▽ ▽ ▽

CHAPTER 7: CUSTOM COMMANDS

In This Section:	Page
• Introduction.....	7-1
• Custom M-codes (Using Defines).....	7-2
• Custom G-codes (Using Calls).....	7-3
• Custom Commands (Using Callback Commands)	7-5

7.1. Introduction

This chapter describes ways the programmer can customize or extend the Aerotech UNIDEX 600 Series controller CNC language. Also, described are two techniques for creating custom G or M-codes; customization using calls and customization using defines. This chapter also provides an explanation of the **CALLBACK** commands that allow the C programmer to attach a G or M-code to an executable program.

The programmer can override the meaning of an existing G or M-code, or define the meaning of a new G or M-code. This also implies that care must be exercised when assigning a G or M code number, so that an existing code is not redefined ! If the programmer overrides an existing code, then they can optionally switch on or off the override at will, either executing their routine or the Aerotech default action.

The customization routines and definitions could be placed in a separate file and included into the main program, so that the customization is transparent to the main program. For example, a program with G2's that performs circles, is user transformable into a program that performs programmer defined actions on a G2. The user does this by placing the single line including the appropriate file at the beginning of the program or in another file included by the main program.

Customization using defines is slightly quicker in execution, but customization by calls allows passing of arguments to the custom G or M-code via arguments. Custom M-codes would be used to perform I/O functions, where speed is important and no arguments are normally needed. For this reason customization by define would normally be used in custom M-codes. Custom G-codes would be used to perform more complex motion related activities that might require parameters. For this reason customization by call would normally be used in custom G-codes. However, the emphasis is, the user may use either method with either type of custom code.

7.2. Custom M-codes (Using Defines)

By using the “#define” compiler directive command the programmer can replace any G-code or M-code with their own CNC program. The define inserts the program block specified directly into the program.

There is no limit to the length of a program a user can set up in a define. The “\” character, used as a continuation character, tells the #define statement to insert a line break and continue looking for text on the next line. Note that the last text line of the define includes the “\” character, because a blank line must follow.

The examples below explain customization using defines and inline comments with those examples. However, we recommend that the programmer fully understand the “#define” command before proceeding.

In the example below, the user defined that M1100 sets output bit 1, but waits for input bit 2 before proceeding to the next CNC line in the program.

EXAMPLE:

```
#define M1100 $BO1 = 1\ ;FIRST LINE OF DEFINE
$GLOBAL0 = M2200\ ;Reads virtual input bit #1
IF ($BI2 EQ 1)\ ;Sets output bit #1 if input bit #2 is set
    $BO1 = 1\
ENDIF\ ;BLANK LINE MUST FOLLOW
;THIS IS THE REQUIRED BLANK LINE
```

7.2.1. Custom M-Code Tips

When creating custom defined G-codes, the exact format of the G-code does make a difference. For example, a **G1** is not the same as a **G01** when creating a custom G-code. In fact, this mechanism has an interesting use. The user can define **G1** to be their custom code, and then **G01** instead executes the default action.

There are a few powerful statements that are of interest to the user when defining M-codes; they are **ONGOSUB**, **ON**, and **WAIT**.

WAIT allows the user to hold on a CNC line until a certain condition is true or time out (this could be used to make an improved version of the example in Section 7.2.).

The **ON** and **ONGOSUB** establish conditions that are monitored constantly throughout program execution. One condition could be when an I/O bit is on or off. This is useful for watching safety type conditions. The **ON** command simply sets a bit when the condition is true. The **ONGOSUB** and **RETURN** command allow for more complex activities when the condition is satisfied, including complete control over motion recovery if the condition occurs during motion.

7.3. Custom G-codes (Using Calls)

By using the “#define” compiler directive command and the **CALL/FARCALL** extended command, the programmer can replace any G-code or M-code with their own CNC program. The axis block that might normally accompany that G-code is then interpreted as arguments to the programmer’s CNC program. Furthermore, the user can switch on and off the G-code override by using the “#define” (with no second argument) and “#include” compiler directives.

The examples below explain customization using defines and inline comments with those examples. However, we recommend that the programmer fully understand the “#define”, “#include”, and “CALL” or “FARCALL” commands before proceeding. Another requirement is the understanding of the syntax of CNC words and call arguments. The example below uses all of these.

In example 1 the user replaced the “G1” command with a call to the subroutine “myG1”, that simply adds any passed X, Y or Z parameter values and places them in global variable zero.

Notice the use of the “.DEFINED” keywords, to ensure we only process those parameters passed into the program.



EXAMPLE 1

```

NO1 G1 X2 Y2          ;Does a normal G1
#define G1 CALL myG1    ;Replaces G1 with call to myG1
NO2 G1 X2.1 Y2 z3.4   ;This calls myG1 which sets $GLOB0 to 7.5
NO3 GX2.1Y2z3.4       ;Same as NO2, but with no separating spaces
#define G1              ;Reverts back to normal G1
NO4 G1 X2 Y2          ;Does a normal G1
NO5 M02

DFS myG1
$GLOB0 = 0
IF ($X.DEFINED) $GLOB0 = $GLOB0 + $X ;Before adding X, make sure we were
;passed an X value
IF ($Y.DEFINED) $GLOB0 = $GLOB0 + $Y ;Before adding Y, make sure we were
;passed a Y value
IF ($Z.DEFINED) $GLOB0 = $GLOB0 + $Z ;Before adding Z, make sure we were
;passed a Z value
RETURN
ENDDFS

```

Note, the “\x20” at the end of the first define statement, (between N01 and N02). The “\x” tells the compiler that the next two digits specify the hexadecimal code of an ASCII character to insert. In this case, “20” is the hexadecimal code for a space. Without this trailing space, line N03 would generate a compiler error, because the subroutine name would be seen as “myG1X2.1Y2z3.4”. This technique must be used, because the compiler removes trailing spaces from replacement strings (at the end of the #define).

This technique is not limited to just spaces, any ASCII code can be inserted this way. Furthermore, it is not limited to just the end of the define, for example:

```
#define G1 CALL /x60/x79/x47/x31 ; This is the same as "#define G1 CALL myG1"
```

The example below shows where the user made the **G1** test binary input 1 before moving. Otherwise, it does not move. Note that the subroutine must switch off the **G1** override before running the standard **G1**. Also, the syntax for transferring the passed argument values into the standard **G1** include the use of a “p” argument to receive the feedrate (only axis and argument CNC words can be passed in). The example suffers from the limitation that it only works in **G91** mode, or if both X and Y values passed in on every move. For example, if we are in **G90** mode and at point {3,3}, then an execution of “G1 X6” moves to {6,0}, not the expected {6,3}. It is left as an exercise to the reader to use the “DEFINED” keyword to correct this defect.

EXAMPLE

```
#define G1 CALL myG1 ;Replaces G1 with call to myG1
G1 X10 Y10
G1 X10 Y10 F100 ;WARNING!! THIS IS INCORRECT SYNTAX, THE F
;CANNOT BE PASSED
G1 X10 Y10 p100 ;This is OK, use "p" to pass in feedrate
M02

DFS myG1
    IF ($BI2 EQ 0) ;If it's OK to move
        #define G1 ;So I can call the normal G1
        $GLOBO = 0
        IF ($p.DEFINED) ;Feedrate was passed in, use it
            G1 X$X Y$Y F$p
        ELSE ;No feedrate passed in, use the default.
            G1 X$X Y$Y
        ENDIF
        #define G1 CALL myG1 ;To reestablish myself
    ENDIF
    RETURN
ENDDFS
```

7.3.1. Custom G-Code Tips

It cannot be emphasized enough that the user must check the “DEFINED” argument variant before using the argument value, otherwise the data may be undefined.

7.4. Custom Commands (Using Callback Commands)

Callback commands allow the user to execute programs. The callback, combined with either of the methods described in sections 7.2. and 7.3., allow the user to define **M** or **G**-codes that execute user-supplied executables.

▽ ▽ ▽

APPENDIX A: GLOSSARY OF TERMS

In This Section:

- IntroductionA-1

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 *task* 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: WARRANTY AND FIELD SERVICE**In This Section:**

- Laser Products.....B-1
- Return Procedure.....B-1
- Returned Product Warranty DeterminationB-1
- Returned Product Non-warranty Determination.....B-2
- Rush Service.....B-2
- On-site Warranty RepairB-2
- On-site Non-warranty RepairB-2

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.

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.

Laser Products

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.

Return Procedure

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
Warranty Determination***

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



A

ABS, 3-9
 Absolute Dimension Programming Mode, 5-95
 Absolute Position Programming, 5-95
 Accel Rate Parameter, 5-88
 Accel Time Parameter, 5-83, 5-87
 Accel/Decel, 5-1, 5-24, 5-81, 5-84
 Accel/Decel Rate Based, 5-88
 Accel/Decel Time Based, 5-87
 Acceleration, 5-39
 Acceleration Mode, Linear, 5-86
 Acceleration Mode, Sinusoidal, 5-84
 Acceleration Rate, Set, 5-86, 5-120
 Acceleration Rates, Setting, 5-89
 Acceleration Time, Set, 5-83
 Acceleration, Instantaneous, 5-39
 Acceleration, Sinusoidal, 5-85
 Acceleration/Deceleration, 5-1, 5-7, 5-81, 5-84
 AccelMode Parameter, 5-84
 ACOS, 3-9
 Activate Cutter Compensation Left, 5-65
 Activate ICRC Left, 5-64
 Activate Left Cutter Compensation, 5-65
 Activate Normalcy Mode Left, 5-48
 Activate Normalcy Mode Right, 5-49
 AerCamTableSetMode, 6-55
 AerMoveInfeedSlave, 6-55
 Aliases, 3-22
 Allow Safe Zone, 5-23, 5-24, 5-54, 5-55
 Analog inputs, 3-29
 Analog Output Control Using PSOT Command, 6-8
 Analog/Digital Output Command (PSOT), 6-106
 ANSI C language standard, 4-2
 APT Variables, 3-7
 Argument Lists, 3-6
 Arguments, 3-6
 Arithmetic operators, 3-1
 ASCII characters, 2-2
 ASCII codes, 1-6
 ASIN, 3-9
 Assignment commands, 3-17
 Asynchronous motion, 5-3
 Asynchronous moves, 5-36
 Axes, Circular, 5-107
 Axes, Linear, 5-107
 Axis parameters, 3-25
 Axis Points, 3-6
 Axis processor based commands, 6-2

B

Backlash compensation, 5-57, 5-59
 BAND, 3-13
 BI, 3-28
 Binaries, 3-28

BIND, 6-12
 Blending contoured moves, 5-36
 Block Delete Character, 2-3
 BNOT, 3-13
 BO, 3-28
 BOR, 3-13
 BXOR, 3-13

C

Call Library Subroutine, 6-47, 6-64, 6-117
 Call Subroutine, 6-47, 6-64, 6-117
 CALLDLL, 6-14
 Cancel Fixture Offset, 5-74
 Case sensitivity, 1-6
 CCW Motion, 5-34, 5-42, 5-54, 5-59
 Characters, 2-2
 Circular Axes, 5-107
 Circular direction codes, 5-110
 Circular Interpolation, 5-25, 5-34, 5-41, 5-42, 5-54
 Circular Interpolation, Inverse, 5-111
 Circular move, 5-6
 Circular moves, 5-47
 CLLS, 6-117
 Clockwise, Spindle, 5-129
 CLS, 6-117
 CNC Block Constants, 3-5
 CNC block syntax, 5-13
 CNC design philosophy, 1-3
 CNC expressions, 3-5
 CNC letter
 E, 3-3
 I, 3-3
 J, 3-3
 K, 3-3
 M, 3-3
 S, 3-3
 T, 3-3
 CNC Letter, 3-3
 CNC Manual Data Input Screen, 5-132
 CNC masks, 3-4
 CNC words, 3-4
 Command Sets, 2-1
 Comment Operator, 2-3
 Comparators, 3-8
 Compiler directive commands, 4-1
 Completed Touch Probe Cycle, 5-73
 Computation precedence, 3-10
 Condition Branch on Errors, 6-79
 Conditional Looping, 6-132
 Conditional Statement, 6-47
 Constant Acceleration vs. 1-Cosine, 5-85
 Constants, 3-14
 Contoured moves, 5-5, 5-6
 Control, Data Collection, 6-4, 6-15, 6-19, 6-25
 COS, 3-9
 Counterclockwise, Spindle, 5-129

- Custom commands, 7-1
 Custom G-codes, 7-2
 Custom M-codes, 7-2
 Cutter Compensation Axes, Set, 5-67
 Cutter Compensation Radius, Set, 5-66
 Cutter Radius Compensation, Intersectional, 5-1, 5-58
 CW Motion, 5-25, 5-41, 5-54
 Cycle, 6-88
 Cycles, Measuring Probe, 5-73, 6-90
- D**
- DATA, 6-19, 6-25
 Data Collection Control, 6-4, 6-15, 6-19, 6-25
 Deactivate Cutter Compensation, 5-63
 Decel Rate Parameter, 5-87, 5-88
 Decel Time Parameter, 5-87
 Deceleration /Acceleration, 5-7
 Deceleration by Force, 5-40, 5-43, 5-51
 Deceleration Rate, Set, 5-87, 5-120
 Define program array, 6-28
 Define statement, 4-2
 Define Subroutine, 6-13
 Define User Variable, 6-27
 Delay, 5-35
 Design philosophy, 1-3
 Detected Probe Input, 5-73
 Digital Output Control Using PSOT Command, 6-8
 Digital Touch Probe Measuring, 5-73
 Digital/Analog Output Control Command (PSOT), 6-106
 Dimensions, Safe Zones, 5-55
 Disable backlash compensation, 5-57
 Disable Feedrate Override, 5-132
 Disable Normalcy Mode, 5-48
 Disable Polar or Cylindrical coordinate transformation, 5-68
 Disable Safe Zones, 5-55
 Disabled Mode of Operation, 5-48
 Discontinue Cutter Compensation, 5-63
 DISPLAY, 6-27
 Distance Programming, 5-96
 Distance Programming Mode, 5-95
 Distances, Setting, 5-89, 5-90
 Dominant Feed Parameter, 5-106
 Dominant Feedrate Overview, 5-105
 DVAR
 reserved word, 6-27
 DVAR command, 6-27
 DVAR extended command, 3-21
 Dwell, 5-35
- E**
- E word, 5-5
 ELSE
 reserved word, 6-47
- Enable cylindrical coordinate transformation, 5-71
 Enable Safe Zone, 5-23, 5-24, 5-54, 5-55
 Enable spindle shutdown, 5-108
 Enable/Disable Position Synchronized Output Firing (PSOF Command), 6-100
 ENDM command, 6-30
 ENDREPEAT
 reserved word, 6-112
 ENDRPT
 reserved word, 6-112
 English Units, 5-89, 5-90
 Entry Block, User Defined, 6-64
 ErrCode task parameter, 6-3
 EXECUTE, 6-30
 Execute OS/2 Program, 6-30
 Execution, Stop, 5-128
 EXP, 3-9
 Expression Component, 3-2
 Expression Element, 3-1
 Expression Type, 3-2
 Expressions, 3-1
- F**
- F word, 5-5
 FARCALL, 6-33
 Fast Feedrate, 5-23
 F-code parameter blocks, 5-22
 Feed Per Min. Feedrate Programming, 5-100
 Feed Per Spindle Rev. Feedrate Programming, 5-101
 Feedrate, 5-14
 Feedrate Mode Programming, 5-99, 5-100, 5-101
 Feedrate Override Lock, 5-132
 Feedrate Override Unlock, 5-132
 Feedrate Override, Disable, 5-132
 Feedrate, Fast, 5-23
 Feedrate, Linear Dominant, 5-107
 Feedrate, Rapid, 5-23
 Feedrates, Setting, 5-89, 5-90
 Field Service Policy, B-1
 File Close Command, 6-36
 File Open Command, 6-36
 File Write Command, 6-40
 Filenames, 4-6
 Firing Distance
 calculations using multiple axes, 6-98
 Firing Distance Command, 6-97
 Firing Distance Entry, 6-7
 Fixture Offset, 5-74
 Fixture Offset Example, 5-80
 Fixture Offset#1, 5-74
 Fixture Offset#2, Setting, 5-76
 Fixture Offset, Canceled, 5-74
 Fixture offsets, 5-74
 Fixture Offsets, 5-76
 Floating point constants, 3-7, 3-12
 Floating point expressions, 3-7

Floating point functions, 3-9	G61, 5-83
Floating point operators, 3-8	G62, 5-84
Floating point variables, 3-8	G63, 5-84
Force Deceleration, 5-40, 5-43, 5-51	G64, 5-86
Force deceleration to zero, 5-109, 5-123	G65, 5-86
FRAC, 3-9	G66, 5-87
FREE, 6-43	G67, 5-87
Functionality, 1-3	G68, 5-88
	G70, 5-89
	G71, 5-90
	G8, 5-39
G	G82, 5-90
G0, 5-23	G9, 5-40
G01, 5-24	G90, 5-95
G02, 5-25	G91, 5-96
G03, 5-34	G92, 5-97
G04, 5-35	G93, 5-99
G100, 5-108	G94, 5-100
G101, 5-108	G98, 5-106
G108, 5-109	G99, 5-107
G109, 5-109	G-code blocks, 3-6
G110, 5-110	G-codes, 5-2
G111, 5-111	GLOB, 3-1, 3-2
G12, 5-41	Global parameters, 3-23
G13, 5-42	Global variables, 3-18
G130, 5-114	Go To User Defined Entry Block, 6-44, 6-64
G131, 5-114	
G16, 5-43	
G165, 5-120	
G166, 5-120	
G17, 5-44	H
G18, 5-44	Halt Program, 5-128
G19, 5-44	Halt Spindle Movement, 5-130
G20, 5-48	Hexadecimal numbers, 3-12
G21, 5-48	HOME, 6-46
G22, 5-49	Home, Position, 5-97
G26, 5-51	
G27, 5-51	
G28, 5-51	I
G29, 5-51	ICRC, 5-1, 5-58
G34, 5-54	ICRC, Activate Right, 5-64
G35, 5-54	ICRC, Cutter Compensation Activate Left, 5-65
G36, 5-54	ICRC, Deactivate, 5-63
G37, 5-55	IF
G38, 5-57	reserved word, 6-47
G39, 5-57	If-Then-Else-EndIf Statement, 6-47
G40, 5-63	Inch Dimension Programming Mode, 5-89
G41, 5-64	Include statement, 4-6
G42, 5-65	Incremental Position Programming, 5-96
G43, 5-66	Index expression, 3-21
G44, 5-67	Initializing Variables, 6-27
G45 command, 5-68	Input, Probe, 5-73
G47 command, 5-71	Instantaneous Acceleration, 5-39
G51, 5-73, 6-90	INT, 3-9
G53, 5-74	Integer constants, 3-12
G54, 5-74	Integer expressions, 3-12
G55, 5-76	Integer operators, 3-13
G60, 5-83	Interpolation, Circular, 5-25, 5-34, 5-41, 5-42, 5-54
	Interpolation, Circular Inverse, 5-111

Interpolation, Linear, 5-24
 Intersection Cutter Radius Compensation, 5-1, 5-58
 Inverse Circular Interpolation, 5-111
 Inverse Time Feedrate Programming, 5-99

J

JUMP, 6-44
 Jump to User Defined Entry Block, 6-44

K

Keyword "F", 5-35, 5-107
 Keywords "I/J/K", 5-110, 5-111

L

Labels, 3-15
 Laser Pulse Output Definition, 6-8
 Left Cutter Compensation, Activate, 5-65
 Left, Path Compensation, 5-66
 Line terminator, 4-4
 Line Terminators, 2-2
 Linear Acceleration Mode, 5-86
 Linear Axes, 5-107
 Linear Feedrate Dominant, 5-107
 Linear Interpolation, 5-24
 Linear move, 5-6
 Lines, 2-3
 Locate Part in Space, 5-73
 Lock Spindle Feedrate Override, 5-132
 Lock, Feedrate Override, 5-132
 Loop, Conditional, 6-132
 Loop, Repeat, 6-112

M

M0, 5-128
 M01, 5-128
 M02, 5-128
 M03, 5-129
 M04, 5-129
 M05, 5-130
 M19, 5-130
 M30, 5-130
 M47, 5-131
 M48, 5-132
 M49, 5-132
 M50, 5-132
 M51, 5-132
 Machine Parameter Set-up Screen, 5-4
 Machine parameters, 3-26
 Manual Feedrate Override, 5-23
 Map command, 1-7
 M-codes, 5-2
 Measuring Cycles, Probe, 5-73, 6-90
 Metric Dimension Programming Mode, 5-90

Metric Units, 5-89, 5-90
 MFO adjustments, 5-39
 Mirror image example, 5-92
 Modal, 5-7
 Modal velocity profiling, 5-109
 Mode of Operation Disabled, 5-48
 Mode, Activate Normalcy Right, 5-49
 Mode, Linear Acceleration, 5-86
 Mode, Normalcy, 5-1, 5-45
 Mode, Normalcy Disabled, 5-48
 Mode, Normalcy Left, 5-48
 Modify Variables, 6-28
 Motion modifier words, 5-21
 Motion type words, 5-21
 Motion types, 5-2
 Motion, CCW, 5-34, 5-42, 5-54
 Motion, CW, 5-25, 5-41, 5-54

N

Normalcy Left, 5-49
 Normalcy Mode, 5-1, 5-45
 Normalcy Mode Left, Activate, 5-48
 Normalcy Mode Right, Activate, 5-49
 Normalcy Right, 5-49

O

Offset words, 5-21
 Offset#1, Fixture, 5-74
 ON command, 6-77
 ONGOSUB, 6-79
 Operators, 3-14
 Optional Stop, 5-128
 OS/2 Program Execution, 6-30
 OSC
 reserved word, 6-88
 OSCILLATE
 reserved word, 6-88
 OSCillate command, 6-88
 Output Definition of Laser Pulse, 6-8
 Output Firing, Enabling and Disabling, 6-100
 Outputs
 control using PSOT command, 6-8
 PSOT Command, 6-106
 setting output voltages of DACs using PSOT, 6-107
 setting using PSOT command, 6-106
 Override, Feedrate Lock, 5-132
 Overview, Accel/Decel, 5-1, 5-81, 5-84
 Overview, Dominant Feedrate, 5-105
 Overview, ICRC, 5-1, 5-58
 Overview, Normalcy Mode, 5-1, 5-45

P

Parameter, Accel Rate, 5-88
 Parameter, Accel Time, 5-83, 5-87

- Parameter, AccelMode, 5-84
 Parameter, Decel Rate, 5-87, 5-88
 Parameter, Decel Time, 5-87
 Parameter, Dominant Feed, 5-106
 Parameter, Ramp Type, 5-87, 5-88
 Parameter, Time Based, 5-83
 Parameters, 3-22
 Parameters, Plane Select, 5-110, 5-111
 Parameters, Rate Based, 5-87
 Parenthesis, 6-5
 Parsing, 1-3
 Parts rotation, 5-93
 Path Compensation Left, 5-66
 Path Compensation Right, 5-65
 PAXIS, 3-25
 Permit Safe Zone, 5-23, 5-24, 5-54, 5-55
 PGLOB, 3-23
 Place Items In Window, 6-27
 Plane Select Parameters, 5-110, 5-111
 Plane Selection, 5-51
 PMACH, 3-26
 Polar/Cylindrical transformations, 5-68
 Position Programming, Incremental, 5-96
 Position Synchronized Output Firing Distance
 Command, 6-97
 Position Synchronized Output Pulse Configuration
 (PSOP), 6-102
 Positioning, Point-to-point, 5-23
 Precedence, 3-10
 Prerequisites, 1-2
 Probe Input, 5-73
 Probe Measuring Cycles, 5-73, 6-90
 Probe Touch, 5-73
 Program array variables, 3-21
 Program Execution, Restart, 5-130, 5-131
 Program Flow, 6-28
 Program Stop, 5-128
 Program variables, 3-20
 PROGRAMMING, 5-1
 Programming Mode, Absolute Dimension, 5-95
 Programming Mode, Distance, 5-95
 Programming Mode, Metric, 5-90
 Programming Units, 5-89
 Programming, Distance, 5-96
 Programming, Feed Per Min., 5-100
 Programming, Feed Per Spindle Rev. Feedrate, 5-101
 Programming, Feedrate Mode, 5-99
 PSOD Command, 6-97
 PSOF Command, 6-100
 PSOP Command, 6-102
 PSOT Command, 6-106
 PTASK, 3-24
 Pulse Configuration Command (PSOP), 6-102
- R**
- Ramp Type Parameter, 5-87, 5-88
- Range, Array Index, 6-28
 Rapid Feedrate, 5-23
 Rapid Motion, 5-39
 Rate Based Parameters, 5-87
 Read Cycle, Touch Probe, 5-73
 REF, 6-46
 Registers, 3-29
 Related documentation, 1-2
REPEAT
 reserved word, 6-112
 Repeat Loop, 6-112
 Replacement string, 4-4
 Replacement strings, 4-2, 4-5
 Restart Program Execution, 5-130, 5-131
 Restore preset position registers, 5-90
 Restrict Safe Zones, 5-55
 Restricted Areas USAF, 5-55
 RI, 3-29
 Right, Path Compensation, 5-65
 RO, 3-29
 Rotary Feedrate Dominant, 5-106
 RPT, 6-112
 reserved word, 6-112

S

- Safe Zone Boundaries, 5-55
 Safe Zone Fault, 5-55
 Safe Zone, Disable, 5-55
 Safe Zone, Enable, 5-23, 5-24, 5-54, 5-55
 Select, Plane, 5-51
 Semicolon, 2-3
 Servo update rate, 5-114
 Set Acceleration Rate, 5-86, 5-120
 Set Acceleration Time, 5-83
 Set Cutter Compensation Axes, 5-67
 Set Cutter Compensation Radius, 5-66
 Set Deceleration Rate, 5-87, 5-120
 Set Fixture Offset #2, 5-76
 Set Profile Time, 5-84
 Setting Acceleration Rates, 5-89
 Setting Fixture Offsets, 5-74
 Simultaneous movement of multiple axes, 5-5
 SIN, 3-9
 Sinusoidal Acceleration, 5-85
 Sinusoidal Acceleration Mode, 5-84
 Software Home, 5-97
 Space, Locate Part, 5-73
 Special Symbol, (), 6-5
 Specify Cutter Compensation Axes, 5-67
 Specify Cutter Compensation Radius, 5-66
 Specify Fixture Offset, 5-74
 Specify Subroutine, 6-13
 Speeds, Setting, 5-89
 Spindle Feedrate Override, Lock, 5-132
 Spindle Feedrate Override, Unlock, 5-132
 Spindle Movement, Stop, 5-130

Spindle Off, 5-130
 Spindle Off/Re-orient, 5-130
 Spindle On Clockwise, 5-129
 Spindle On Counterclockwise, 5-129
 Spindle shutdown mode, 5-108
 SQRT, 3-9
 Stand-alone words, 5-21
 Stop Cutter Compensation, 5-63
 Stop Spindle Movement, 5-130
 Stop, Optional, 5-128
 Stop, Program, 5-128
 String32 expressions, 3-14
 String32 operators, 3-15
 String32 constants, 3-15
 String32 variables, 3-15
 STRTASK, 3-20
 Subroutine, Call, 6-12, 6-33, 6-47, 6-64, 6-117
 Subroutine, Define, 6-13
 Subroutine, Library Call, 6-12, 6-33, 6-47, 6-64, 6-117
 Symbols, Comment, 2-3
 Synchronization, 6-97
 Synchronous motion, 5-2
 Syntax parsing, 1-3

T

TAN, 3-9
 Target Positions, 5-5
 Target word, 4-3
 Task parameter
 ErrCode, 6-3
 Task parameters, 3-24
 Task variables, 3-20
 THEN
 reserved word, 6-47
 Time Based Parameter, 5-83
 Time, Dwell, 5-35
 Time-outs, 6-3

Tool Orientation, 5-45
 Touch Probe Measuring, Digital, 5-73
 Touch Probe Read Cycle, 5-73
 Tracking Axes, Specifying, 6-7
 Truncation, 5-5

U

Units, English, 5-89, 5-90
 Units, Metric, 5-89, 5-90
 Units, Programming Mode, 5-89, 5-90
 Unlock Spindle Feedrate Override, 5-132
 Unlock, Feedrate Override, 5-132
 User Defined Entry Block, Jump, 6-44, 6-64
 User Variable, 6-27

V

Value, Axis Parameter, 6-119
 Variable, User, 6-27
 Variables, 1-4, 3-18
 Variables V0-V255, 6-97
 Variants, 3-14, 3-16
 Vector feedrate, 5-6
 Velocity, 5-6, 5-39, 5-40
 Velocity Profile with G9, 5-41
 Velocity Profile without G9, 5-40
 Velocity profiling, 5-36
 Verbal comparator, 3-8
 Virtual I/O, 3-28

W

Wait for Cycle Start, 5-130
 Warranty Policy, B-1
 While-Do-Endwhile, 6-132
 Whitespace, 1-6, 2-2
 Window, Item Placing, 6-27

▽ ▽ ▽



READER'S COMMENTS

UNIDEX U600 Series CNC Programming, Win NT/95 Manual P/N EDU 158, June 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) 963-7009

