CLC
Multi-Axis Coordinated Motion Control

Reference Manual
DOK-VISMOT-VM*-05VRS**-FBK1-AE-P
FOREWORD

Special Notations:

Special notations are used in this manual to assist the reader in identifying unique conditions or information that is important. Three categories of notations are listed below in ascending order of importance.

Note: A NOTE is a tip, suggestion or emphasized procedure for operating the equipment.

Caution: A CAUTION appears when a condition exists which could cause operating faults or damage to the equipment.

Warning: WARNING statements identify conditions which could cause bodily harm and/or severe damage to the equipment if the operator is not careful operating the equipment. A WARNING typically describes the potential hazard, its possible effect, and measures that must be taken to avoid the hazard.

Please NOTE that due to variations found in the operating conditions of certain applications and their working environments, the notations in this manual cannot identify all potential problems or hazards. Caution and discretion must always be used in operating machinery, especially when using electrical power. Equipment should only be installed and operated by trained personnel.

* Repair and Training services are available from REXROTH INDRAMAT.

The Rexroth Corporation
Indramat Division
5150 Prairie Stone Parkway
Hoffman Estates, IL, Illinois 60192
Ph. (847) 645-3600 | FAX (847) 645-6201
# RECORD OF REVISIONS

<table>
<thead>
<tr>
<th>Revision Level</th>
<th>Date</th>
<th>Description of Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>10/91</td>
<td>Initial Release</td>
</tr>
<tr>
<td>B</td>
<td>6/94</td>
<td>Revised to include updates</td>
</tr>
<tr>
<td>C</td>
<td>10/94</td>
<td>Revised to include updates, chapters reorganized (not released to customers)</td>
</tr>
<tr>
<td>D</td>
<td>12/94</td>
<td>Revised for VisualMotion Version 1.2</td>
</tr>
<tr>
<td>E</td>
<td>9/95</td>
<td>Revised for VisualMotion Version 2.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Updated Icon and Text Programming chapters</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Updated I/O and Parameter chapters</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Reorganized into two manuals:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 Start-Up Guide - IAE 68010</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 Reference Manual - IAE 74792</td>
</tr>
<tr>
<td>F</td>
<td>10/96</td>
<td>Revised for VisualMotion Version 5.0</td>
</tr>
</tbody>
</table>
CONTENTS

FOREWORD

RECORD OF REVISIONS

CHAPTER 1. INTRODUCTION

1.1. PURPOSE OF MANUAL ................................................................. 1-1
  1.1.1. Manual Overview .............................................................. 1-1
1.2. OVERVIEW OF CLC ................................................................. 1-3
  1.2.1. CLC System Architecture .................................................. 1-3
  1.2.2. Indramat’s VisualMotion™ Programming Interface .................... 1-4
  1.2.3. CLC Teach Pendant ............................................................ 1-4
  1.2.4. CLC Operating System ....................................................... 1-4
1.3. CLC MOTION CAPABILITIES .................................................... 1-5
  1.3.1. Non-Coordinated Motion ...................................................... 1-5
  1.3.2. Coordinated Motion .......................................................... 1-6
  1.3.3. Electronic Line Shaft (ELS) ................................................ 1-7

CHAPTER 2. CLC INPUT/OUTPUT SYSTEMS

2.1. I/O OVERVIEW ........................................................................... 2-1
2.2. REMOTE I/O SYSTEMS ............................................................. 2-2
2.3. I/O MAPPER ............................................................................. 2-3
  2.3.1. I/O Mapper Ladder Logic Format ......................................... 2-4
  2.3.2. I/O Mapper Boolean Format ............................................... 2-4
  2.3.3. I/O Mapper Operators ......................................................... 2-5
  2.3.4. I/O Mapper Considerations ............................................... 2-5
2.4. I/O BIT FORCING .................................................................... 2-6
2.5. READING AND WRITING PHYSICAL I/O ................................... 2-6
2.6. CLC REGISTERS ...................................................................... 2-7
  2.6.1. Register 1: System Control ................................................. 2-8
  2.6.2. Registers 2-5: Task Control ............................................... 2-10
  2.6.3. CLC Cycle Control Considerations ..................................... 2-13
  2.6.4. Register 6: System Diagnostic Code .................................. 2-14
  2.6.5. Registers 7-10: Task Jog Control ....................................... 2-15
  2.6.6. Registers 11-18, 209-240: Axis Control ............................... 2-18
  2.6.7. Register 21: System Status ................................................ 2-20
  2.6.8. Registers 22-25: Task Status ............................................. 2-22
  2.6.9. Registers 27 and 28: Eagle Module Inputs/Outputs (CLC-V Only) 2-23
  2.6.10. Register 29: ELS Master Control Register ......................... 2-24
  2.6.11. Register 30: ELS Master Status Register ............................ 2-26
2.6.13. Registers 40-89: DEA I/O .................................................................................. 2-29
2.6.14. Registers 90 and 91: Latch and Unlatch .............................................................. 2-29
2.6.15. Registers 95-97: Teach Pendant Status Registers .............................................. 2-30
2.6.16. Registers 98, 99: Teach Pendant Control - Task A-B, C-D .............................. 2-31
2.6.17. CLC Reserved Register Tables ............................................................................. 2-32

CHAPTER 3. PARAMETERS

3.1. OVERVIEW .............................................................................................................. 3-1
3.2. PARAMETER TRANSFER COMMANDS ............................................................... 3-3
3.3. LIST OF PARAMETERS ......................................................................................... 3-4
3.4. SYSTEM PARAMETERS ......................................................................................... 3-13
  3.4.1. System Setup .................................................................................................... 3-13
  3.4.2. Jogging and Display ....................................................................................... 3-22
  3.4.3. Program Management ..................................................................................... 3-24
  3.4.4. System Status ................................................................................................ 3-27
  3.4.5. Electronic Line Shaft Parameters 1 of 2 ......................................................... 3-32
  3.4.6. VME Bus ......................................................................................................... 3-36
  3.4.7. Teach Pendant ............................................................................................... 3-43
  3.4.8. Electronic Line Shaft Parameters 2 of 2 ......................................................... 3-46
  3.4.9. System Parameter Lists .................................................................................. 3-51
3.5. TASK PARAMETERS .............................................................................................. 3-62
  3.5.1. Task Setup ....................................................................................................... 3-62
  3.5.2. Coordinated Motion ....................................................................................... 3-64
  3.5.3. Robotics .......................................................................................................... 3-68
  3.5.4. Coordinated Motion Status ............................................................................ 3-69
  3.5.5. Task Status ..................................................................................................... 3-71
  3.5.6. Task Parameter Lists ...................................................................................... 3-76
3.6. AXIS PARAMETERS ............................................................................................. 3-77
  3.6.1. Axis Setup ....................................................................................................... 3-77
  3.6.2. Axis Status ..................................................................................................... 3-93
  3.6.3. Electronic Line Shaft ...................................................................................... 3-97
  3.6.4. Axis Feedback Capture (Registration) ............................................................. 3-102
  3.6.5. Optional SERCOS Data .................................................................................. 3-103
  3.6.6. Axis Parameter Lists ...................................................................................... 3-104
3.7. DRIVE PARAMETERS .......................................................................................... 3-105
  3.7.1. Drive Status .................................................................................................... 3-105
  3.7.2. Required DDS Setup ..................................................................................... 3-106

CHAPTER 4. VISUALMOTION MENU COMMANDS

4.1. INTRODUCTION ................................................................................................. 4-1
4.2. THE FILE MENU ............................................................................................... 4-1
4.2.1. Program Management ................................................................. 4-2
4.2.2. Archive ...................................................................................... 4-4
4.2.3. Transfer Cams ............................................................................ 4-5
4.2.4. Transfer Events ........................................................................... 4-6
4.2.5. Transfer I/O Mapper ................................................................. 4-6
4.2.6. Transfer Parameters ................................................................. 4-7
4.2.7. Transfer Points ........................................................................... 4-8
4.2.8. Transfer Variables ................................................................. 4-8
4.2.9. Transfer Zones ........................................................................... 4-9
4.2.10. Print ......................................................................................... 4-9

4.3. The Edit Menu ............................................................................. 4-10
4.3.1. Clear Current Task .................................................................... 4-10
4.3.2. Find, Find Next ......................................................................... 4-10
4.3.3. Add Subroutine .......................................................................... 4-11
4.3.4. Add Event Function .................................................................. 4-11
4.3.5. Labels - User Labels .............................................................. 4-11
4.3.6. Labels - Register Labels .......................................................... 4-13
4.3.7. Labels - Bit Labels .................................................................... 4-14
4.3.8. Import User Label File ............................................................ 4-14
4.3.9. Export User Label File .............................................................. 4-14

4.4. The View Menu ........................................................................... 4-15
4.4.1. Subroutines .............................................................................. 4-15
4.4.2. Event Functions ....................................................................... 4-15
4.4.3. Zoom Out .................................................................................. 4-15

4.5. The Setup Menu ........................................................................... 4-16
4.5.1. Card Selection .......................................................................... 4-16
4.5.2. Configuration ............................................................................ 4-17
4.5.3. Drives ....................................................................................... 4-17
4.5.4. Coordinated Motion .............................................................. 4-18
4.5.5. I/O Setup .................................................................................. 4-19
4.5.6. Overview .................................................................................. 4-20
4.5.7. Pendant Security ....................................................................... 4-21
4.5.8. CLC Serial Ports ..................................................................... 4-22
4.5.9. VME Configure ........................................................................ 4-22

4.6. The Tools Menu ........................................................................... 4-23
4.6.1. Breakpoint Control ................................................................. 4-23
4.6.2. CAM Builder ............................................................................ 4-24
4.6.3. Jogging ..................................................................................... 4-28
4.6.4. Oscilloscope ............................................................................ 4-31
4.6.5. Show Program Flow .............................................................. 4-32

4.7. The Data Menu ........................................................................... 4-32
4.7.1. Events ....................................................................................... 4-32
4.7.2. I/O Mapper .............................................................................. 4-33
4.7.3. PID Control Loops ................................................................. 4-39
4.7.4. PLS ........................................................................................................... 4-43
4.7.5. Points ....................................................................................................... 4-49
4.7.6. Registers ................................................................................................... 4-51
4.7.7. Sequencer .................................................................................................. 4-54
4.7.8. Variables .................................................................................................. 4-57
4.7.9. Zones ......................................................................................................... 4-58
4.8. The Status Menu .......................................................................................... 4-59
  4.8.1. Diagnostic Log ......................................................................................... 4-59
  4.8.2. Drives ...................................................................................................... 4-59
  4.8.3. Drives on Ring ......................................................................................... 4-60
  4.8.4. System ..................................................................................................... 4-60
  4.8.5. Tasks ....................................................................................................... 4-61
4.9. The Options Menu ....................................................................................... 4-62
4.10. Help ............................................................................................................. 4-63

CHAPTER 5. PROGRAMMING CONCEPTS

5.1. Overview ....................................................................................................... 5-1
5.2. Program Tasks .............................................................................................. 5-1
  5.2.1. Command Execution ............................................................................. 5-1
5.3. Events ........................................................................................................... 5-2
  5.3.1. Time-based Events ................................................................................. 5-2
  5.3.2. Distance-based Events ......................................................................... 5-3
  5.3.3. Repeating Axis Position (Rotary) Events ............................................. 5-5
  5.3.4. Interrupt Input Events ........................................................................... 5-6
  5.3.5. VME Events ......................................................................................... 5-7
  5.3.6. Feedback Capture (Probe) ................................................................... 5-8
  5.3.7. Event Tables .......................................................................................... 5-9
5.4. Subroutines .................................................................................................. 5-10
  5.4.1. Function Arguments and Local Variables ........................................... 5-10
5.5. Sequencer ..................................................................................................... 5-11
  5.5.1. Single Stepping a Sequencer Step ......................................................... 5-12
  5.5.2. Single Stepping a Step Function ............................................................ 5-12
5.6. Data and Expressions ................................................................................... 5-13
  5.6.1. Integers and Floats ................................................................................ 5-13
  5.6.2. Constants and Variables .................................................................... 5-13
  5.6.3. Expressions ......................................................................................... 5-15
  5.6.4. Mathematical and Logical Operators .................................................. 5-16
5.7. Tables ............................................................................................................ 5-17
  5.7.1. Event Table ............................................................................................ 5-19
  5.7.2. CLC Path Planner ............................................................................... 5-20
  5.7.3. Absolute Point Table ........................................................................... 5-22
  5.7.4. Relative Point Table ............................................................................ 5-23
  5.7.5. Zone Protection Table .......................................................................... 5-24
CHAPTER 6. ICON PROGRAMMING

6.1. INTRODUCTION ........................................................................................................... 6-1
  6.1.1. Working with VisualMotion’s Pointer and Icon Palettes ....................................... 6-1
  6.1.2. Connecting Icons ................................................................................................. 6-2
  6.1.3. Icon Labels and Comments .................................................................................. 6-3
  6.1.4. User Defined Labels ............................................................................................ 6-3

6.2. CLC ICONS .................................................................................................................. 6-4
  6.2.1. Accel ...................................................................................................................... 6-5
  6.2.2. Axis ....................................................................................................................... 6-6
  6.2.3. AxisEvt .................................................................................................................. 6-11
  6.2.4. Branch .................................................................................................................. 6-12
  6.2.5. Calc ....................................................................................................................... 6-14
  6.2.6. Cam ....................................................................................................................... 6-16
  6.2.7. CamAdj .................................................................................................................. 6-17
  6.2.8. CamBuild .............................................................................................................. 6-18
  6.2.9. Circle .................................................................................................................... 6-20
  6.2.10. Decel ................................................................................................................... 6-22
  6.2.11. ELS ...................................................................................................................... 6-23
  6.2.12. ELSAdj ................................................................................................................. 6-26
  6.2.13. ELSMode ............................................................................................................ 6-27
  6.2.14. Event ................................................................................................................... 6-28
  6.2.15. Finish ................................................................................................................... 6-29
  6.2.16. Go ......................................................................................................................... 6-29
  6.2.17. Home ................................................................................................................... 6-30
  6.2.18. I/O ......................................................................................................................... 6-31
  6.2.19. Join ....................................................................................................................... 6-32
  6.2.20. Joint ...................................................................................................................... 6-33
  6.2.21. Line ....................................................................................................................... 6-34
  6.2.22. Move .................................................................................................................... 6-35
  6.2.23. Msg ....................................................................................................................... 6-36
  6.2.24. Param ................................................................................................................... 6-37
  6.2.25. Path ....................................................................................................................... 6-39
  6.2.26. Path/Hold ............................................................................................................ 6-40
  6.2.27. PID ....................................................................................................................... 6-41
  6.2.28. PLS ....................................................................................................................... 6-43
  6.2.29. Position ................................................................................................................ 6-44
  6.2.30. PrmBit ................................................................................................................... 6-45
  6.2.31. Prmint ................................................................................................................... 6-46
  6.2.32. Probe ................................................................................................................... 6-47
  6.2.33. ProbeEvt ............................................................................................................... 6-48
  6.2.34. Ratio .................................................................................................................... 6-49
  6.2.35. Reg ....................................................................................................................... 6-50
  6.2.36. RobotOrg ............................................................................................................. 6-51
6.2.37. RobotTool .......................................................... 6-51
6.2.38. Scissor .............................................................. 6-52
6.2.39. Seq ................................................................. 6-52
6.2.40. Size ................................................................. 6-54
6.2.41. Start ................................................................. 6-55
6.2.42. Stop ................................................................. 6-56
6.2.43. Sub ................................................................. 6-57
6.2.44. Veloc ............................................................... 6-58
6.2.45. VME ................................................................. 6-59
6.2.46. VMEEvt ........................................................... 6-61
6.2.47. Wait................................................................. 6-62

CHAPTER 7. TEXT LANGUAGE PROGRAMMING

7.1. OVERVIEW .......................................................... 7-1
7.2. DIRECTIVES ........................................................ 7-1
  7.2.1. EQU (Equate) .................................................. 7-1
  7.2.2. DEFINE .......................................................... 7-2
  7.2.3. EVENT/START & EVENT/END .............................. 7-3
  7.2.4. TASK/START & TASK/END ................................. 7-3
7.3. CLC INSTRUCTIONS ............................................... 7-4
  7.3.1. Instruction Format ........................................... 7-4
  7.3.2. AXIS/EVENT .................................................. 7-7
  7.3.3. AXIS/HOME .................................................. 7-8
  7.3.4. AXIS/INITIALIZE ........................................... 7-9
  7.3.5. AXIS/MOVE (Single Axis, Non-Coordinated) ............ 7-11
  7.3.6. AXIS/RATIO .................................................. 7-13
  7.3.7. AXIS/SPINDLE (Continuous Velocity Mode) ............ 7-15
  7.3.8. AXIS/START .................................................. 7-17
  7.3.9. AXIS/STOP ................................................... 7-19
  7.3.10. AXIS/WAIT (Axis Wait For In-Position) ................. 7-21
  7.3.11. CALL .......................................................... 7-23
  7.3.12. CAM/ACTIVATE ............................................. 7-23
  7.3.13. CAM/ADJUST ............................................... 7-25
  7.3.14. CAM/BUILD .................................................. 7-26
  7.3.15. CAM/STATUS Command .................................... 7-28
  7.3.16. CAPTURE/ENABLE ......................................... 7-29
  7.3.17. CAPTURE/SETUP ........................................... 7-31
  7.3.18. DATA/SIZE (Configure Memory Allocations) ........... 7-33
  7.3.19. DEFINE (Define Name For Variable) ..................... 7-36
  7.3.20. DELAY (Suspend Task Execution) ....................... 7-38
  7.3.21. ELS/ADJUST (Adjust ELS Axis) .......................... 7-39
  7.3.22. ELS/DECEL (Set Master Decel) .......................... 7-41
  7.3.23. ELS/INIT (Initialize ELS Axes) ........................ 7-42
7.3.24. ELS/MODE (Set ELS Axis Mode) .......................................................... 7-45
7.3.25. ELS/STOP ................................................................. 7-46
7.3.26. EVENT/DONE (Signal Event Completed) ........................................ 7-47
7.3.27. EVENT/END (Mark End of Event) ................................................. 7-49
7.3.28. EVENT/START (Start of Event function) ........................................ 7-50
7.3.29. EVENT/TRIGGER (Trigger a Task Event) ....................................... 7-51
7.3.30. EVENT/WAIT (Pause Task for Event Done Signal) ............................ 7-53
7.3.31. FUNCTION/ARG ............................................................ 7-55
7.3.32. FUNCTION/END .......................................................... 7-56
7.3.33. FUNCTION/START ......................................................... 7-57
7.3.34. GOSUB (Go To Subroutine) ...................................................... 7-58
7.3.35. GOTO (Go To Mark) .......................................................... 7-59
7.3.36. IF (If-Else-Endif Conditional Branch) .......................................... 7-60
7.3.37. KINEMATIC (Use a Kinematic Definition for a Task) ....................... 7-64
7.3.38. LOCAL/VARIABLE ........................................................ 7-65
7.3.39. MESSAGE/DIA (Task Diagnostic Message Definition) ....................... 7-66
7.3.40. MESSAGE/STATUS (Task Status Message Definition) ....................... 7-67
7.3.41. MOVE/CIRCLE (Coordinated Move with Circular Interpolation) .......... 7-68
7.3.42. MOVE/Joint (Coordinated Move Joint Point to Point) ....................... 7-70
7.3.43. MOVE/LINE (Coordinated Move with Straight Line Interpolation) ....... 7-71
7.3.44. PARAMETER/BIT (Initialize Parameter Bit) .................................... 7-73
7.3.45. PARAMETER/GET (Load Parameter to a Variable) ........................... 7-75
7.3.46. PARAMETER/INIT (Initialize a Parameter) .................................... 7-77
7.3.47. PARAMETER/SET (Set a Parameter) ........................................... 7-78
7.3.48. PATH/ABORT (Aborts Coordinated Motion) .................................... 7-80
7.3.49. PATH/POSITION (Get Current Path Absolute Position) ..................... 7-81
7.3.50. PATH/RESUME (Resume Coordinated Motion) .................................. 7-82
7.3.51. PATH/STOP (Halt Coordinated Motion) ......................................... 7-83
7.3.52. PATH/WAIT (Pause Program for Motion) ....................................... 7-84
7.3.53. PID/CONFIGURE ........................................................................ 7-85
7.3.54. PLC/CLEAR (Clear I/O Register Bit) ............................................. 7-86
7.3.55. PLC/READ (Read I/O Register(s)) .............................................. 7-87
7.3.56. PLC/SET (Set I/O Register Bit) .................................................. 7-88
7.3.57. PLC/TEST (Test I/O Register Bit) ................................................ 7-89
7.3.58. PLC/WAIT (Pause Program for I/O) ............................................. 7-90
7.3.59. PLC/WRIT (Write to I/O Register(s)) ......................................... 7-91
7.3.60. PLS/INIT (Compiler directive) .................................................. 7-92
7.3.61. RETURN (Return From Subroutine) .............................................. 7-94
7.3.62. ROBOT/ORIGIN ......................................................................... 7-95
7.3.63. ROBOT/TOOL ........................................................................... 7-96
7.3.64. SEQUENCER .............................................................................. 7-97
7.3.65. SEQ/LIST ................................................................................ 7-98
7.3.66. SEQ/STEP ................................................................................ 7-99
7.3.67. TASK/AXES (Task Axes Definition) ............................................. 7-100
7.3.68. TASK/END (Mark the End of a Task) ................................................. 7-102
7.3.69. TASK/START (Define the Start of a Task(s)) ................................. 7-103
7.3.70. VAR/INIT ...................................................................................... 7-104
7.3.71. VME/BROADCAST (Broadcast Event to VME) ................................. 7-105
7.3.72. VME/EVENT (Send Event to VME Unit) ........................................ 7-106
7.3.73. VME/READ (VME Bus Data Read) ................................................. 7-107
7.3.74. VME/WRITE (VME Bus Data Write) ................................................. 7-109

CHAPTER 8. PROGRAM DEBUGGING/MONITORING AND DIAGNOSTICS

8.1. FINDING PROGRAM PROBLEMS ....................................................... 8-1
8.1.1. Test Code .................................................................................. 8-1
8.2. CLC COMPILER BASE CODE ......................................................... 8-2
8.2.1. Base Code instruction mnemonics and valid arguments ..................... 8-2
8.3. ICON LANGUAGE WARNINGS AND ERROR MESSAGES .................... 8-19
8.4. TEXT LANGUAGE ERROR MESSAGES .............................................. 8-22
8.4.1. First Pass Errors ........................................................................ 8-22
8.4.2. Second Pass Compiler Errors ..................................................... 8-23
8.5. CLC STATUS AND DIAGNOSTICS ..................................................... 8-31
8.5.1. Status Messages ........................................................................ 8-31
8.5.2. Warning Messages .................................................................... 8-31
8.5.3. Shutdown Messages .................................................................. 8-31
8.5.4. Diagnostic Messages .................................................................. 8-32
8.5.5. Fatal System Errors .................................................................... 8-32
8.6. SYSTEM DIAGNOSTICS - CODES AND MESSAGES ......................... 8-32
8.6.1. Diagnostic Codes ........................................................................ 8-32
8.6.2. Status Messages ........................................................................ 8-33
8.6.3. Warning Messages ..................................................................... 8-34
8.6.4. Shutdown Messages ................................................................... 8-34

APPENDIX A. CLC DDE SERVER

A1.1. DYNAMIC DATA EXCHANGE ....................................................... A-1
A1.1.1. The Dynamic Data Exchange Server ........................................... A-2
A1.1.2. Dynamic Data Exchange Interface ............................................. A-2
A1.2. THE COMMUNICATION SERVERS MAIN WINDOW .......................... A-4
A1.2.2. Settings Menu - Serial Communications ...................................... A-7
A1.2.3. Settings Menu - VME Communications ....................................... A-8
A1.2.4. Settings Menu - PC Bus Communications .................................... A-9
A1.2.5. DDE Menu .............................................................................. A-10
A1.3. AT MODEM CONFIGURATION DIALOG ....................................... A-12
A1.4. SERVER TOPIC NAME ................................................................ A-13
APPENDIX B . DIRECT ASCII COMMUNICATION

B 1.1. OVERVIEW ................................................................. B-1
B 1.2. CLC COMMUNICATION PROTOCOL ............................................. B-1
  B 1.2.1. Reading Data from the CLC .............................................. B-2
  B 1.2.2. Writing Data to the CLC ............................................... B-2
  B 1.2.3. Communication Errors ................................................. B-3
  B 1.2.4. Checksum ................................................................ B-3
  B 1.2.5. End of Message .......................................................... B-4
  B 1.2.6. Backspaces and White spaces .......................................... B-4
  B 1.2.7. Numeric Data Formats .................................................. B-4
  B 1.2.8. Format of Data Sent to the CLC ..................................... B-4
B 1.3. COMMAND CLASSES/ SUBCLASSES ........................................ B-5
  B 1.3.1. Parameters ................................................................. B-5
  B 1.3.2. Variables .................................................................. B-5
  B 1.3.3. PID ........................................................................ B-6
  B 1.3.4. Point Tables ............................................................... B-6
  B 1.3.5. Event Tables ............................................................... B-7
  B 1.3.6. Program Communication ............................................. B-7
  B 1.3.7. Functions ................................................................. B-8
  B 1.3.8. I/O Registers ............................................................. B-8
  B 1.3.9. Sequencer Data .......................................................... B-8
  B 1.3.10. Zones .................................................................. B-9
B 1.4. DRIVE AND CLC PARAMETERS AND SUBCLASSES ......................... B-9
  B 1.4.1. Parameter Data Subclass .............................................. B-9
  B 1.4.2. Name Text Subclass .................................................... B-10
  B 1.4.3. Units Text Subclass ..................................................... B-10
  B 1.4.4. Upper Limit, L: Lower Limit Subclasses ....................... B-10
  B 1.4.5. Attribute Subclass ....................................................... B-10
  B 1.4.6. Parameter Lists Subclasses ....................................... B-10
  B 1.4.7. SERCOS Parameter Sets ............................................. B-10
B 1.5. PARAMETER LISTS ............................................................ B-11
  B 1.5.1. Listing a Parameter .................................................... B-11
  B 1.5.2. Parameter List Block Transfer ..................................... B-13
B 1.6. USER PROGRAM VARIABLES ................................................. B-16
  B 1.6.1. 'P': Data ................................................................ B-16
  B 1.6.2. 'T': Label Text .......................................................... B-16
B 1.7. PID ........................................................................ B-17
  B 1.7.1. Getting the assigned variables ..................................... B-17
B 1.8. POINT TABLES ................................................................ B-18
  B 1.8.1. Point Table Data ......................................................... B-18
  B 1.8.2. Point Table Data, Row Format .................................... B-20
  B 1.8.3. List of All User-defined Labels (V) ............................... B-21
B 1.9. EVENT TABLES .................................................................. B-21
APPENDIX C. ELS CONFIGURATION

C 1.1. ELECTRONIC LINE SHAFTING (ELS) OVERVIEW ................................................. C-1
C 1.2. ELS MASTERS ........................................................................................................ C-1
  C 1.2.1. Real Masters .................................................................................................... C-1
  C 1.2.2. Virtual Masters ............................................................................................... C-2
  C 1.2.3. Second Master ................................................................................................. C-2
C 1.3. ELS SLAVES ........................................................................................................... C-2
  C 1.3.1. Phase Synchronization ..................................................................................... C-3
  C 1.3.2. Velocity Synchronization ................................................................................ C-5
  C 1.3.3. CAM Synchronization ..................................................................................... C-5
  C 1.3.4. Dynamic synchronization .............................................................................. C-6
  C 1.3.5. Slave Secondary Mode (Non-Dynamic Synchronization) ................................. C-7
C 1.3.6. Summary of ELS Parameters and Registers ................................................. C-8
C 1.4. CAM APPLICATIONS ...................................................................................... C-9
C 1.4.1. Overview .................................................................................................... C-9
C 1.4.2. Cam Commands ......................................................................................... C-9
C 1.5. BUILDING CAM TABLES ............................................................................. C-10
C 1.5.1. PCAM Cam Build Type ............................................................................ C-11
C 1.5.2. VCAM Cam Build Type ............................................................................ C-12
C 1.5.3. ACAM Cam Build Type ............................................................................ C-14
C 1.5.4. SCAM Cam Build Type ............................................................................ C-15
C 1.6. CAM EQUATION ............................................................................................. C-16
C 1.6.1. Consider H = 0 and L ≠ 0, Sph = 0. ....................................................... C-16
C 1.6.2. Consider H ≠ 0, L = 0, M = N = 1 and Mph = Sph = 0. ....................... C-17
C 1.6.3. Consider H = L = 1, M = N = 1, and Mph = Sp = 0? ........................... C-19
C 1.6.4. Consider L = 0, H = 1, M = 2, N = 1 and Mph = Sp = 0 ..................... C-20
C 1.6.5. Consider L = 0, H = M = N = 1 and Mph = 90, Sph = 90 degrees. .... C-21
C 1.7. CLC CAM ALIGNMENT ................................................................................ C-22
C 1.7.1. Aligned Status Bit (Axis Status Register bit 8) ........................................ C-23
C 1.8. DRIVE CAMS ................................................................................................ C-24
C 1.8.1. Drive Cam Equation ............................................................................... C-24
C 1.8.2. Axis Configuration .................................................................................... C-24
C 1.8.3. Drive Cam Table Parameters ................................................................ C-25
C 1.8.4. Drive Cam Alignment .............................................................................. C-26
C 1.8.5. Phase Adjusted Status (Axis Status Register bit 4) ............................... C-27
C 1.8.6. Drive Cam Configuration Notes ............................................................... C-28
C 1.9. DIFFERENCES BETWEEN CLC AND DRIVE CAMS ................................ C-29
C 1.10. APPLICATION (ROTARY CUTOFF) .......................................................... C-30

APPENDIX D. CLC DRIVE PARAMETER EDITOR

D 1.1. OVERVIEW .................................................................................................... D-1
D 1.2. FILE MENU .................................................................................................. D-2
D 1.2.1. Transfer Parameters ............................................................................. D-2
D 1.2.2. Print ......................................................................................................... D-2
D 1.3. PARAMETER MENU ................................................................................... D-3
D 1.3.1. Analog Outputs ...................................................................................... D-3
D 1.3.2. Drive Direction ....................................................................................... D-4
D 1.3.3. Drive Name ............................................................................................. D-5
D 1.3.4. Drive Monitoring ................................................................................... D-5
D 1.3.5. Drive Tuning .......................................................................................... D-6
D 1.3.6. Drive Limits ............................................................................................ D-7
D 1.3.7. Drive Reference ...................................................................................... D-8
D 1.3.8. Mechanical ............................................................................................. D-10
D 1.3.9. Overview ............................................................................................... D-11
D 1.4. OSCILLOSCOPE ......................................................................................... D-13
D 1.4.1. File Menu .................................................................................. D-13
D 1.4.2. Source Menu ............................................................................. D-14
D 1.4.3. Timing ...................................................................................... D-14
D 1.4.4. Signal Selection ......................................................................... D-15
D 1.4.5. Abort, Upload and Enable Trigger ........................................... D-16
CHAPTER 1. INTRODUCTION

1.1. Purpose of Manual

This document is a reference for the users of the CLC motion control card. For information pertaining to other system components, refer to the following documentation:

- DIAx03 Drive with Electronic Transmission Function
  (Pub No. to be determined, Part No. to be determined)
- DDS 2.1 W/ Analog Interface Application Manual
  (Pub No. 209-0069-4315-03, Part No. 257182)
- DDS 2.1/3.1 W/ Sercos Interface Application Manual
  (Pub No. 209-0069-4321-04, Part No. 262156)
- DKS Intelligent Digital Servo Drive Project Planning Manual
  (Pub No. 209-0069-4355-00, Part No. 259310)
- DKS and MDD Intelligent Digital AC Servo Drives Application Manual
  (Pub No. 209-0069-4351-01, Part No. 261809)

NOTE: This manual is to be used in conjunction with the CLC Start Up Guide, IAE 68010 Rev. A, 12/95 (Part No. 601563). Contact Indramat Customer Service if you do not have a copy of this Guide.
## 1.2. Manual Overview

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CHAPTER 1. INTRODUCTION</strong></td>
<td>Describes the CLC’s general theory of operation and its motion capabilities.</td>
</tr>
<tr>
<td>Chapter 2 - CLC Input/Output Systems</td>
<td>Describes the functionality of the built-in, memory-based I/O registers and the CLC’s I/O mapper.</td>
</tr>
<tr>
<td>Chapter 3 - Parameters</td>
<td>Lists the CLC parameters, divided into four functionally related groups: System, Axis, Drive and Task.</td>
</tr>
<tr>
<td>Chapter 4 - VisualMotion Menu Commands</td>
<td>Describes VisualMotion’s menu commands.</td>
</tr>
<tr>
<td>Chapter 5 - Programming Concepts</td>
<td>Describes the basic programming elements that are common to both the CLC Icon and Text languages and their implementation.</td>
</tr>
<tr>
<td>Chapter 6 - Icon Programming</td>
<td>Describes how to use icons and other VisualMotion commands to create motion programs.</td>
</tr>
<tr>
<td>Chapter 7 - Text Language Programming</td>
<td>Describes the CLC’s Text Language programming capability, which permits writing CLC programs using Windows’ Notepad or a similar ASCII text-only editor.</td>
</tr>
<tr>
<td>Chapter 8 - Programming Debugging / Monitoring and Diagnostics</td>
<td>Discusses debugging strategies and describes the CLC compiler base code. It also provides a listing of Icon and Text language compiler error messages, as well as various CLC status and diagnostic messages.</td>
</tr>
<tr>
<td>Appendix A - CLC DDE Server</td>
<td>Describes the DDE Server, an application through which VisualMotion communicates with the CLC and other applications.</td>
</tr>
<tr>
<td>Appendix B - Direct ASCII Communications</td>
<td>Describes the CLC serial communication protocol, which can be used to communicate with a wide variety of devices and programs.</td>
</tr>
<tr>
<td>Appendix C - ELS Configuration</td>
<td>Describes the CLC / Drive resident cams and how to build, run and modify cam applications.</td>
</tr>
<tr>
<td>Appendix D - CLC Drive Parameter Editor</td>
<td>Describes the CLC Drive Parameter Editor which includes drive tuning, limits and mechanical setup, as well as oscilloscope and diagnostic applications.</td>
</tr>
<tr>
<td>Appendix E - CLC Transfer Utility</td>
<td>Describes CLC_XFER.exe utility which is a Windows based Dynamic Data Exchange Server application used to transfer data to/from Indramats CLC motion control cards.</td>
</tr>
</tbody>
</table>
1.3. Overview of CLC

1.3.1. CLC System Architecture

The CLC card is a part of a larger motion control system which also includes digital servo drives and SERCOS, a fiber-optic communication system. The CLC can provide multi-axis coordinated and non-coordinated motion control with tightly integrated I/O logic control functions. The flexibility of the CLC allows it to be used for a wide variety of applications, from general motion control to sophisticated multi-axis electronic line shafting (ELS) to robotics.

CLC controls use SERCOS (SErial Realtime COmmunications System) fiber-optic interface to interconnect with Indramat drives. The SERCOS interface is an internationally accepted standard for real-time high-speed digital communication. It requires only a single daisy-chained fiber-optic cable interconnecting the drives with the control. Synchronous data protocol guarantees response time. It provides continuous monitoring and diagnostics for all devices and includes comprehensive standardized definitions for control loop functions.

The CLC’s open bus architecture provides easy I/O interfacing to virtually any manufacturer’s PLC’s and I/O cards. Both digital and analog I/O are available. Interrupt-type inputs can be used to provide minimum response time recognition of external events.

Three versions of the CLC card are currently available:

**CLC-V**
VME bus architecture.

**CLC-P**
IBM PC-AT bus architecture.

**CLC-D**
Plugs into Indramat’s digital drives, providing an exceptional cost-effective motion control solution.
1.3.2. Indramat’s VisualMotion™ Programming Interface

The CLC motion control card combines an integrated multi-tasking environment with a unique graphical Windows based programming interface - VisualMotionTM (VM). VM provides simplified point-and-click programming, operation and management. With this software, system builders have a flexible and comprehensive environment, including easy DDE integration with applications such as Wonderware InTouch or Visual Basic programs.

1.3.3. CLC Teach Pendant

An optional Teach Pendant can also be used to control CLC operation and adjust CLC parameters. The teach pendant is a hand-held instrument with 16 x 42 character display and a 40-key sealed membrane keypad. The pendant provides a convenient operation and position programming interface for Indramat CLC Motion Control. Refer to the CLC Start Up Guide for more information.

1.3.4. CLC Operating System

A CLC card can simultaneously control up to four independent user tasks (A, B, C & D). Each task can control a coordinated group of two or three axes and any number of independent motion axes. Depending on the application, a single CLC may control up to 40 axes.

CLC multitasking is cooperative and can consecutively execute one instruction from each task. Special event functions preempt normal multitasking for real-time response and are prioritized according to the hierarchy of the event’s associated task (Task A - highest priority, Task D - lowest priority). Two additional executive tasks manage the user interfaces through serial communication ports. One controls communication to the Host PC, the other controls the Teach Pendant when used.

Each task also has a queue for events, permitting asynchronous initiation of multiple independent events within each task. CLC events are a privileged form of subroutine, suspending all CLC program tasks until pending events are done. Each task may have up to four events active at one time. The CLC provides several event types including time-based, distance-based, and position-based events, and others.

Task A events have the highest priority, while events associated with Task D have the lowest. Therefore, an event currently being executed within Task C will be suspended by an event from Task B. Once the active events are completed the suspended program tasks automatically re-activate and resume execution.
1.4. CLC Motion Capabilities

The CLC supports three kinds of motion (non-coordinated, coordinated, and electronic line shaft) and several modes of each type.

1.4.1. Non-Coordinated Motion

Non-coordinated motion is primarily used to control a single independent axis. There are two modes of non-coordinated motion: Single Axis, for linear positioning required to achieve point-to-point movement, and Velocity Mode, as might be used for some spindle motor drives.

**Single axis**
Single axis motion commands within a user program are interpreted by the CLC and sent to the DDS drive. The user program tells the DDS drive the speed, and/or distance and acceleration it should use to internally develop a velocity profile, which is then maintained and controlled within the intelligent DDS drive. Consequently, single axis motion does not require continuous calculation by the CLC and consumes minimum CLC resources.

**Velocity Mode**
Velocity mode controls only the speed of the axis, without any position information. The intelligent DDS-2 drive maintains torque and velocity loops internally, updating the internal loops every 250 microseconds.

**Ratioed Axes**
A special form of non-coordinated motion permits linking two axes by relating the number of revolutions of a slave axis to a master axis. For example, a ratio might be required when the positioning axis of a gantry robot, having a motor on each side of its supporting track, must travel along a circular track.
1.4.2. Coordinated Motion

The CLC defines multi-axis coordinated motion in terms of a path composed of standard straight line and circular geometry segments. Point positions, \((x, y, z)\), are used to establish the start, middle or end of a geometry segment. Two points define a line, three points define a circle. The path combines these standard geometry segments so that the start of the next segment begins at the end of the previous segment. A path, therefore, is nothing more than a collection of connected segments.

Since each segment has an end point specifying speed, acceleration, deceleration and jerk, each segment can have a unique rate profile curve. A special type of segment, called a blend segment, can be used to join two standard geometry segments. Blend segments provide the capability of continuous smooth motion from one standard segment to another without stopping. They reduce calculation cycle time as well as provide a means of optimal path shaping.

The CLC is capable of calculating a path in any of several different modes:

**Constant Speed**
Constant Speed mode is always active and tries to maintain a constant speed between any two connecting segments in the path. This mode is constrained by the system’s acceleration and deceleration. Constant speed is the optimum path motion for applying adhesives or paint, and welding and some forms of cutting such as laser or water-jet, etc.

**Linear Interpolation**
A coordinated motion straight line segment is defined by two points. The motion is calculated from the end point of the last segment, or the current position if the system is not in motion, to the new end point. Multi-axis coordinated motion is used when a relationship must be maintained between two or more axes during motion.

**Circular Interpolation**
A coordinated motion circular segment is defined by three points. Circular motion begins with the end point of the last segment executed, or the current system position if the system is not in motion, moves in a circular arc through an intermediate point, and terminates at the specified endpoint.

**Kinematics**
In addition to the standard linear and circular segments, the CLC has the capability of executing forward and inverse kinematic movement by using an application-specific library of kinematic functions. Kinematics must be developed by Indramat to customer specifications. Contact Indramat Applications Engineering to inquire about applications which could benefit from kinematics.
1.4.3. Electronic Line Shaft (ELS)

An Electronic Line Shaft is used to synchronize one or more slave axes to a master axis. An ELS master can be a real or virtual axis. A real master can be another axis in the system, or an external feedback device such as an encoder. A virtual master is a command generated by the CLC. (See ELS Icon, Chapter 6) Each slave axis can use either velocity, phase or cam synchronization. An ELS also includes the capability to jog each axis synchronously or independently, and to adjust phase offset and velocity while the program is running.

**Velocity synchronization** relates slave axes to a master in terms of rotational rate. It is used when axis velocities are most critical, as in paper processing operations in which two or more motors act on a single piece of fragile material.

**Phase synchronization** maintains the same relative position among axes, but adjusts the lead or lag of the slaves to the master in terms of degrees. It is used when the positions of axes are most critical. For example, to achieve proper registration in printing operations, the axis controlling the print head may be programmed for a particular phase offset relative to some locating device, such as a proximity switch.

**Cam synchronization** is used when custom position, velocity or acceleration profiles are needed at a slave axis. These special profiles are developed at the slave by sending position commands every SERCOS cycle.

A cam is an (x, y) table of positions that relate a master axis to a slave. Cams can be stored on the CLC or on the digital drive. CLC cams have more adjustment options and can work with any SERCOS drive. Drive cams are more efficient and can be applied to more axes. The same programming commands and utilities are used for both drive-resident and CLC-resident cams.

*See Appendix C - ELS Configuration for more information.*
CHAPTER 2. CLC INPUT/OUTPUT SYSTEMS

2.1. I/O Overview

CLC I/O (input/output) is used for operating functions such as System, Task and Axis control. I/O is also used to track system status and diagnostics. The CLC Executive, VisualMotion Program Instructions and remote I/O systems can all read and write I/O data.

The CLC uses an internal block of memory to manage I/O systems. The memory is arranged in a linear array of “Registers” which are 16 bits wide. Each bit can have a value of 1 or 0 which corresponds with its “On” or “Off” state. The CLC-V memory allocation allows for a maximum of 1024 I/O registers, while the CLC-D and CLC-P allow for a maximum of 512.

The following registers are reserved by the CLC for system functions:

<table>
<thead>
<tr>
<th>Register</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.6.1</td>
<td>Register 1: System Control</td>
</tr>
<tr>
<td>2.6.2</td>
<td>Registers 2-5: Task Control</td>
</tr>
<tr>
<td>2.6.4</td>
<td>Register 6: System Diagnostic Code</td>
</tr>
<tr>
<td>2.6.5</td>
<td>Registers 7-10: Task Jog Control</td>
</tr>
<tr>
<td>2.6.6</td>
<td>Registers 11-18, 209-240: Axis Control</td>
</tr>
<tr>
<td>2.6.7</td>
<td>Register 21: System Status</td>
</tr>
<tr>
<td>2.6.8</td>
<td>Registers 22-25: Task Status</td>
</tr>
<tr>
<td>2.6.9</td>
<td>Registers 27 and 28: Eagle Module Inputs/Outputs (CLC-V Only)</td>
</tr>
<tr>
<td>2.6.10</td>
<td>Register 29: ELS Master Control Register</td>
</tr>
<tr>
<td>2.6.11</td>
<td>Register 30: ELS Master Status Register</td>
</tr>
<tr>
<td>2.6.12</td>
<td>Registers 31-38, 309-340: Axis Status</td>
</tr>
<tr>
<td>2.6.13</td>
<td>Registers 40-89: DEA I/O</td>
</tr>
<tr>
<td>2.6.14</td>
<td>Registers 90 and 91: Latch and Unlatch</td>
</tr>
<tr>
<td>2.6.15</td>
<td>Registers 95-97: Teach Pendant Status Registers</td>
</tr>
<tr>
<td>2.6.16</td>
<td>Registers 98, 99: Teach Pendant Control - Task A-B, C-D</td>
</tr>
</tbody>
</table>

The remaining registers can be used for remote I/O systems and can function like the registers used in any programmable controller.
2.2. Remote I/O Systems

There are several hardware configurations which allow the CLC to read from and write to remote I/O systems. Remote I/O can be accessed through DDS-drive based I/O cards, VME based I/O subsystems (CLC-V only), or PC based I/O subsystems (CLC-P only). Remote I/O can also be accessed through the SERCOS loop using a Lutze I/O device. See Chapter 4 - I/O Setup under the Setup Menu for more information.

Since the CLC uses industry standard bus architecture, it can control virtually any bus-based I/O system, as long as the specific I/O device software drivers are available. The amount of I/O registers available for a specific CLC system design depends upon the manufacturer, the size of
the installed I/O system and the system specific configuration. *See the appropriate hardware Appendix in the CLC Start Up Guide - IAE 68010* for selecting I/O subsystems.

In addition to using program I/O instructions to change I/O lines, I/O activity may be related to motion by using high priority motion or time-related "Event" subroutines. Events behave in a manner similar to real-time hardware interrupts, and may be programmed to trigger at a specific distance along a commanded motion or at a specific time related to the motion. *See Chapter 5. Programming Concepts - Events,* for a description of event functions.

### 2.3. I/O Mapper

The I/O Mapper is a background function used by the CLC Executive to logically link or “map” the state of bits within the I/O registers. The logical relationship between these bits is defined in a file that contains a list of Boolean equations. These equations can be written in Visual Motion using either the ladder logic editor or the Boolean text editor.

The I/O Mapper allows the user to specify logical system relationships between physical I/O lines and CLC I/O bits. System builders can create custom I/O maps for individual applications that can be saved and transferred to any CLC. During normal operation the I/O Mapper is transparent to user programs. The I/O Mapper equations are executed within 250 μs every 4 ms, regardless of the SERCOS cycle time.

![Diagram of I/O Mapper and CLC Memory](image)

**Example:** When a physical start switch closes, the DEA card writes to Input Register 40, which is mapped to Register 2 - bit 6 (*Cycle Start Resume*). This initiates an action in the CLC program execution. When Axis 1 reaches its target position, the state of Register 31 bit 7 (*Axis in Position*) is mapped to Output Register 41 bit 1. The DEA card reads this data and a signal lamp is energized.
2.3.1. I/O Mapper Ladder Logic Format

The CLC I/O mapper equations can be represented in a ladder logic format using the VisualMotion Ladder Editor. In this format the previous example would be represented by the following diagrams:

If register 40 bit 1 received input from a physical I/O switch, its state (open or closed) would change the state of Register 2 bit 6 (on or off). This is similar to an electrical switch energizing or de-energizing a coil or a lamp.

For more information on the I/O Mapper and the Ladder Editor see Chapter 4. - I/O Mapper under the Data Menu.

2.3.2. I/O Mapper Boolean Format

The CLC I/O Mapper equations can also be represented using Boolean logic.

\[(\text{Control} \_ \text{register} \text{#} \- \text{bit} \text{#}) = (\text{Input} \_ \text{register} \text{#} \- \text{bit} \text{#}) \text{ (Operator)} (\text{Input} \_ \text{register} \text{#} \- \text{bit} \text{#}) \ldots \]
\[(\text{Output} \_ \text{register} \text{#} \- \text{bit} \text{#}) = (\text{Status} \_ \text{register} \text{#} \- \text{bit} \text{#}); \text{ Comments} \]

where: register - an integer value for an I/O register
bit - an integer value in the range 1 to 16 or a hex mask
Operator - one of the I/O Mapper logical operators (listed below)

The left side of the expression is set to the evaluated result of the right side of the expression. The string is parsed left to right. Operators are executed as they are parsed, with all operators having the same level of precedence.

A semi-colon at the end of an I/O Mapper string will allow comments to be entered.

Example:
2-1=100-2; sets Task A, Auto Mode (Register 2, bit 1) to the state of register 100 bit 2.
2.3.3. I/O Mapper Operators

Valid I/O Mapper operators include:

!  NOT, the complement operator. The bit is inverted.
&  AND, a bitwise and. If both bits are 1, the result is 1, else the result is 0.
|  OR, a bitwise inclusive or. If either bit is 1, the result is 1.

) Parentheses allow calculation of intermediate results or forcing of precedence. Up to 16 nested operations can be contained in one string.

Example:

\[
\begin{align*}
1-2 &= 100-5 \& 100-12 \\
107-1 &= 106-1 \& 106-3 | 106-5 \\
107-1 &= 106-1 \& (106-3 | 106-5)
\end{align*}
\]

Sets register 1, bit 2 TRUE if register 100 bit 5 and register 100 bit 12 are both True.

The string is processed from left to right if no parentheses are used, The “AND” and “OR” operators have equal precedence.

2.3.4. I/O Mapper Considerations

Directly changing a mapped bit, using VisualMotion or Direct ASCII Communication, may only result in a momentary change of state. The direct change will immediately modify the bit. However, at some time, within the CLC executive cycle time, the I/O Mapper may alter the bit state according to the user specified mapping.

The mapper must complete all operations within 4ms, leaving room for user tasks to run. This includes reading all CLC internal I/O registers, evaluating and mapping all logical expressions, then writing to all physical I/O registers.

The logic strings used to map I/O may also be accessed through the CLC’s serial port as a multi-step parameter list (C-0-3000) by using the CLC’s Direct ASCII Communication capability. See Chapter 3. Parameters for more related I/O Mapper information:

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>C-0-3001</td>
<td>I/O Mapper Options</td>
</tr>
<tr>
<td>C-0-3004</td>
<td>I/O Mapper File size</td>
</tr>
<tr>
<td>C-0-3003</td>
<td>I/O Mapper Total Operations</td>
</tr>
<tr>
<td>C-0-3005</td>
<td>I/O Mapper Executable Size</td>
</tr>
</tbody>
</table>
2.4. I/O Bit Forcing

The CLC provides a register forcing capability that allows a Host system to directly change the state of individual I/O register bits overriding both the physical I/O and the CLC I/O Mapper. Bit forcing circumvents the problem of the I/O Mapper re-mapping a bit that you need to change directly.

Two forcing values allow changing I/O register bits. A 16 bit forcing mask value enables forcing for individual bits in a selected register. A 16 bit forcing state value may then change the state of the enabled individual I/O bits.

The following algorithm mathematically describes how the final register state is set:
New register state = (old state & inverted forcing mask) | (forcing state & forcing mask)

Example:

| forcing mask | 0000 0000 0011 1000 |
| forcing state | 0000 0000 0000 1000 |
| inverted forcing mask | 1111 1111 1100 0111 |
| old register state | 0000 1101 1010 0001 |
| new register state | 0000 1101 1000 1001 |

When the forcing state changes bits in a CLC control register, all edge detection is reset. Forcing remains in effect until the forcing mask is cleared to zero. A 1 in a forcing mask and a 0 in the forcing state will force a zero in the new register state.

**NOTE**: I/O forcing is provided as a system configuration and debugging tool. Forcing can modify control register bits affecting the safe operation of the system. A failure of serial communication between the Host and CLC will prevent the Host from being able to clear a forcing mask, possibly during active motion in the system. USE CAUTION!

For more information on I/O bit forcing see Chapter 4 - Registers under the Data Menu.

2.5. Reading and Writing Physical I/O

The CLC’s physical I/O lines are read and written once during a 4ms system scan cycle (2 or 4ms). The fixed scan time provides a deterministic response and I/O sub-system independence from the user tasks.

Within each cycle, the I/O lines are always read before they are written. Near the beginning of the system scan time, the I/O system-specific drivers read data from the I/O lines and store the data to the CLC’s I/O register array. The data then remains accessible to user programmed tasks during
the CLC’s normal multitasking. Towards the end of the system scan time, the I/O drivers transfer the contents of the CLC’s I/O registers to the physical I/O registers.

Note that for very high-speed events it is possible to change an I/O register state between these two scans, without affecting the physical I/O line itself.

Although it is possible to read or write to I/O systems directly when the I/O ports reside in CLC accessible memory, this is discouraged. It is advisable to use the CLC’s I/O system to access I/O lines. Direct reading and writing of I/O requires a high level of expertise with the CLC, the specific I/O sub-system and the host bus system.

2.6. CLC Registers

The CLC reserves the first 100 registers for system and task, control and status functions. If Indramat drivers for Indramat supported I/O subsystems are used, registers 100 to 119 are typically reserved for inputs and registers 120 to 139 are typically reserved for outputs. A default I/O mapping is used to map physical I/O lines to the appropriate CLC system and task control register bits. However, as previously noted, all CLC physical I/O depends upon the user installation and system configuration.

The following sections describe the CLC reserved registers, the function of control and status bits within these registers, and the I/O mapping provided as a system default mapping.
2.6.1. Register 1: System Control

The first CLC register is reserved for system control. The System Control register bits are dedicated to system supervisory control functions.

<table>
<thead>
<tr>
<th>Register-Bit</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-1</td>
<td>Parameter Mode</td>
</tr>
<tr>
<td>1-3</td>
<td>nEmergency Stop</td>
</tr>
<tr>
<td>1-5</td>
<td>Clear All Errors</td>
</tr>
<tr>
<td>1-6</td>
<td>Pendant Live Man</td>
</tr>
<tr>
<td>1-8</td>
<td>Activate Program</td>
</tr>
<tr>
<td>1-9</td>
<td>Program Select LSB</td>
</tr>
<tr>
<td>1-10</td>
<td>Program Select Bit 2</td>
</tr>
<tr>
<td>1-11</td>
<td>Program Select Bit 3</td>
</tr>
<tr>
<td>1-12</td>
<td>Program Select MSB</td>
</tr>
<tr>
<td>1-14</td>
<td>Pendant Enable</td>
</tr>
<tr>
<td>1-15</td>
<td>Pendant Level LSB</td>
</tr>
<tr>
<td></td>
<td>Must be set to 1’s to enable teach pendant editing and function keys</td>
</tr>
<tr>
<td>1-16</td>
<td>Pendant Level MSB</td>
</tr>
<tr>
<td></td>
<td>Must be set to 1’s to enable teach pendant editing and function keys</td>
</tr>
</tbody>
</table>

**Bit 1: Parameter Mode/ nRun Mode**
When this bit changes from (0) to (1), Parameter Mode is selected. All user tasks are immediately stopped. The system is switched into parameter mode, and the drives are switched into SERCOS phase 2.

When this bit changes from (1) to (0), the system is re-initialized into Run Mode. Parameter initializations are performed and the drives are switched from phase 2 to phase 4. If there are no errors, the user tasks are ready to operate.

**Bit 3: nEmergency Stop**
This input is active low (0 = Emergency Stop). When it is set to (0), all user tasks are stopped except those selected to run during errors. All motion is stopped, and the drives are set to zero velocity and disabled. When it is set to (1), the emergency stop condition has been corrected, and the tasks can run if there are no other errors.

**Bit 5: Clear All Errors**
When this bit changes from (0) to (1), any existing errors are cleared. This includes system, task, and drive errors. If drives were previously enabled, they are restored to an enabled state. To run the tasks, another cycle start transition is required, unless the automatic start option is enabled. To clear errors, a transition of (0) to (1) is required.
Bit 6: Pendant Live Man
This bit should be wired to the teach pendant live man switch when the teach pendant is used. When it is (0), no motion can be initiated from the teach pendant and any motion in progress is immediately stopped. When it is (1) (live man closed), the teach pendant can jog, start, and stop motion.

Bit 8-12: Activate Program and Binary Program Select
The active program can optionally be changed using I-O bits 9-12 in the System Control Register. A transition from (0) to (1) on Activate Program bit 8 will activate the program based on bits 9-12. These bits correspond to the program number (from 1 to 10). If the Activate Program bit is high at power-up, the program selected with the Binary Program Select bits will be activated.

Note that the user interface (Visual Motion, Teach Pendant, etc.) selections take precedence over the bits. The actual active program is acknowledged in System Status Register bits 9-12.

Example:
To activate program 5, set the bits as follows:
Bit 9: 1
Bit 10: 0
Bit 11: 1
Bit 12: 0

Then Bit 8 requires a transition from 0 to 1.

Bit 14: Pendant Enable
This bit toggles control of tasks and jogging between the teach pendant and the I-O system. When it is set to (0), the system I-O mapper and control registers are in control. When it is set to (1), the teach pendant assumes control of system functions, forcing all relevant bits in the control registers.

Bits 15-16: Pendant Access Level
These bits provide access protection for teach pendant menus. The protection levels are defined per-menu to provide restricted access to data. If a menu’s protection level exceeds the value of these bits, the menu can be viewed but not edited.
2.6.2. Registers 2-5: Task Control

The CLC reserved registers 2, 3, 4 and 5 are dedicated to task control for Tasks A, B, C and D respectively. The state of the control register bits determine how the CLC executive executes the user task programs. User programs may read, set or clear bits in these registers to monitor and control each task’s status.

The task control bits permit independent control of the operations within each of the four tasks in a user program.

Setting Auto Mode enables execution of a task. The current version of the software restarts the task at the beginning of the program. Future revisions may provide a more flexible restart method.

Clearing Auto Mode switches the CLC into Manual, ramping down all axes used by a task and aborting the task without finishing the current instructions.

Cycle Start/Resume starts or resumes the selected task.

A Cycle Stop finishes current instruction. An immediate Cycle start resumes the program at the next instruction.

Single Step executes one instruction at a time.

<table>
<thead>
<tr>
<th>Bit</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Mode: Auto /nManual</td>
</tr>
<tr>
<td>2</td>
<td>Override Automatic Start</td>
</tr>
<tr>
<td>4</td>
<td>Single Step</td>
</tr>
<tr>
<td>6</td>
<td>Cycle Start/Resume</td>
</tr>
<tr>
<td>7</td>
<td>nTask Stop</td>
</tr>
<tr>
<td>9</td>
<td>Task Event Trigger</td>
</tr>
<tr>
<td>10</td>
<td>Trace Enable</td>
</tr>
<tr>
<td>11</td>
<td>Breakpoint Enable</td>
</tr>
<tr>
<td>12</td>
<td>Sequencer Single Step</td>
</tr>
<tr>
<td>13</td>
<td>Step Sequence Function</td>
</tr>
</tbody>
</table>

Registers 2 - 5 for Tasks A - D
Bit 1: Mode: Automatic/ nManual
This bit selects the mode of operation for a task. When it is (0), the task is in Manual Mode, the user program does not run, and manual jogging is enabled. When it is (1), the task is in Automatic Mode and is ready to begin execution.

Switch from Manual to Automatic (0 to 1):
The instruction pointer is reset to the beginning of the program, and all events become inactive. At the next (0-1) transition of the cycle start bit when the cycle stop bit is (1), the program starts running.

Switch from Automatic to Manual (1 to 0)
The user task and any events associated with it stop execution immediately. Any motion associated with the task is immediately decelerated to zero velocity. Coordinated, single-axis, and velocity axes are stopped using the maximum deceleration. The ELS master is stopped using the E-Stop deceleration. The instruction pointer is set to where the program was stopped, but is reset to the beginning of the program when the task is returned to Auto Mode.

Bit 2: Override Automatic Start
The ‘Automatically Start Task’ parameter option (T-0-0002, bit 4) allows a task to start immediately upon exit from parameter mode or clearing of an error. The “Override Automatic Start” bit can be used to temporarily disable this function. The bits in the task control register are enabled as long as bit 2 is high (1), and are ignored when the bit is low (0). This allows the automatic start option to be disabled in case the task needs to be stopped or the debugging bits need to be used.

Bit 4: Single Step Select
When Bit 4 = 1, the task is placed in Single Step Mode. Each positive (0 to 1) transition of Cycle Start bit executes one user task program instruction then pauses (providing that the system is in Automatic mode and nTask Stop is inactive). If this bit is set while a task is running the current instruction completes. The task then pauses and waits for a Cycle Start transition.

Event functions cannot be single-stepped. If one or more events have been started or queued by executing the current instruction the events will always continue to completion before pausing the user task program.

When Bit 4 = 0, the task is in Normal Run Mode. When Single Step is zero, normal cycling begins at the next positive transition of the Cycle Start bit (providing that automatic mode is true and nTask Stop is false).

Bit 6: Cycle Start/Resume
When this bit is changed from (0) to (1), the user task starts executing at the current instruction, if the task is in automatic mode, the nTask Stop bit is (1), and there are no errors. It is also used to resume the task after a task stop and to restart the task after entering automatic mode. If single-stepping is enabled, the next instruction is executed with each positive transition.
A (0 to 1) transition is required to start or resume the task

**Bit 7: nTask Stop**
When Bit 7 = 0, the nTask is stopped. A negative transition (1 to 0) stops the task program at the end of the current instruction. All types of motion are halted and can be resumed at the next Cycle Start. The nTask Stop bits function as a “Pause”. When the bit is set to 0, the CLC pauses execution of the task and the instruction pointer remains at the current instruction. When the bit is again = 1 and the cycle start bit is toggled, the task resume execution at the current instruction.

All types of events will execute during a cycle stop state. Only the main program flow in tasks A, B, C and D are affected.

A nTask Stop decelerates **Coordinated Motion** to zero velocity after the current instruction. Motion is then paused on the current segment. All distance and time-based events remain active. As long as the task is in automatic mode, the next 0 to 1 transition on the cycle start will resume motion and complete all pending segments.

**Single axis** motion halts each axis in the task by decelerating the axes to zero velocity while retaining target position. The previous state of the GO command is saved until the next cycle start. If the GO command was active, motion will be resumed at the next cycle start in automatic mode. All normal and repeating events remain active.

All **Velocity** mode axes are decelerated to zero velocity if ramping is enabled, or set to zero velocity if step command is selected. All events remain active.

The **ELS** master axis is decelerated to zero velocity. The previous state of the GO command is saved until the next cycle start. If the GO command was active, the master will be commanded the last programmed velocity. All events on the ELS slaves remain active. The slave axes remain synchronized to the master if synchronization was enabled. Because the CLC has no control over a real master, motion of the slaves of a real master cannot be changed.

**Ratioed** slave axes always follow the master axis. Events remain active and motion will continue until the master axis is stopped.

The operation of **Torque** mode axes during a nTask Stop are not defined at this time.

**NOTE:** nTask Stop does nothing to assure that the system is in a safe or known condition to stop. nTask Stop simply completes the current instruction, then ramps down motion in the task. See "Cycle Stop in User Programs" in 2.6.3. CLC Cycle Control Considerations, for more information.
**Bit 9: Task Event Trigger**
This bit is reserved as an Event Interrupt Input for each task. Each low-to-high (0 to 1) transition of this input will trigger an event to the corresponding task. This event type can be used to start a process or to respond to an external event.

In the event table, Type 6 selects an Interrupt Input event. The event/trigger (arm event) instruction enables the interrupt input. The event/done (disarm event) instruction is used to disable the input. The CLC scans the input every 4ms and queues an event upon a low-to-high transition. The event function will take priority over the user tasks, allowing quick response to an external input.

The I-O mapper can be used to reverse the logic of the interrupt input, or to direct other external inputs to it. Logic in the event function can then scan the multiple inputs to determine the source of the interrupt.

**Bit 11: Breakpoint Enable**
When this bit is set to (1), the breakpoint enabled in task parameter Tx.137 is active. When program flow reaches the breakpoint, the task is stopped. When this bit is set to (0), the program executes normally, without breakpoints.

**Bit 12: Sequencer Single Step**
This bit places the CLC into Sequence Single Step mode. As long as this bit is (1), a (0-1) transition on the cycle start bit causes the program to be stopped after each sequencer step is executed. If this bit is (0), the sequencer executes normally.

**Bit 13: Step Sequence Function**
This bit places the CLC into Function Single Step mode. As long as this bit is (1), a (0-1) transition on the cycle start bit causes the program to be stopped after each sequencer function is executed. If this bit is (0), the sequencer executes normally.

**2.6.3. CLC Cycle Control Considerations**

**Cycle Stop in User Program**
A cycle stop implies that a task’s motion cycle has completed and the system is at a safe place to halt. The nTask Stop bit cannot always be used for this purpose, since it only stops task instruction execution and commands the drives to ramp down.

A nTask Stop signal may lose track of user task activity that is related to axis or segment position or time. In addition, the CLC’s path planner may have several queued segments or events. Queued events always continue execution until completion. This may result in the
system position and I/O losing synchronization with your programmed sequence of task instructions. If you attempt to simply restart motion the results may not be predictable.

If each of your tasks require a cycle stop capability, a separate user task I/O bit should be user configured into the I/O system for each task needing a cycle stop. Your program then tests the associated I/O bit from within your task program. Use the condition of the I/O bit to branch to a program routine that halts motion and establishes a known system state. Since you are programming a unique system, only you can determine a safe system condition.

**System Parameter Mode**
A switch to Parameter Mode immediately disables all user tasks, and switches the DDS drives to SERCOS Phase 2.

**System Shutdown Errors**
A shutdown error disables all drives, stops coordinated and single-axis motion, and puts the task into manual mode. All control bits are left at their current state.

**Programmed End of Task**
A task program that reaches and executes the task/end instruction or the Finish Icon has the same effect as activating the $n$Task Stop I/O line.

### 2.6.4. Register 6: System Diagnostic Code

This status register shows the current CLC diagnostics code, in Motorola 16-bit format.
2.6.5. Registers 7-10: Task Jog Control

The reserved registers 7, 8, 9 and 10 are dedicated to jogging control for Tasks A, B, C and D respectively.

Axis Jogging
Any drive-controlled axis (single-axis, velocity, or ELS) may be jogged independently when its associated task is in manual mode. ELS axes can also be jogged with the master. See Chapter 4 - Coordinated Motion under the Setup Menu.

Coordinated Jogging
Coordinated axes may be jogged in any direction while a task is in manual mode with the Task Jog Register. The coordinate to jog, type of jog, and parameters to use are selected using bits in this register. Single-axis and velocity mode axes are jogged using the Axis Control Registers.

Motion is started with a low-to-high transition on either the jog forward or the jog reverse bit. Motion is stopped when both jog bits are low, when both jog bits are high, when the task mode selection changes, or when a travel limit or incremental distance has been reached.

Jog mode (continuous or incremental) and jog speed or distance are selected in the Task Jog Register for the task associated with the axis.

<table>
<thead>
<tr>
<th>Register-Bit</th>
<th>Task</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>7-1</td>
<td>A</td>
<td>Continuous/incremental</td>
</tr>
<tr>
<td>7-2</td>
<td></td>
<td>Coordinated Jog Forward</td>
</tr>
<tr>
<td>7-3</td>
<td></td>
<td>Coordinated Jog Reverse</td>
</tr>
<tr>
<td>7-4</td>
<td></td>
<td>Jog Type LSB</td>
</tr>
<tr>
<td>7-5</td>
<td></td>
<td>Jog Type MSB</td>
</tr>
<tr>
<td>7-6</td>
<td></td>
<td>Distance/Speed</td>
</tr>
<tr>
<td>7-9</td>
<td></td>
<td>Jog X Coord</td>
</tr>
<tr>
<td>7-10</td>
<td></td>
<td>Jog Y Coord</td>
</tr>
<tr>
<td>7-11</td>
<td></td>
<td>Jog Z Coordinate</td>
</tr>
<tr>
<td>7-12</td>
<td></td>
<td>Jog Joint 4</td>
</tr>
<tr>
<td>7-13</td>
<td></td>
<td>Jog Joint 5</td>
</tr>
<tr>
<td>7-14</td>
<td></td>
<td>Jog Joint 6</td>
</tr>
<tr>
<td>8-1 through 14</td>
<td>B</td>
<td>Same as Task A</td>
</tr>
<tr>
<td>9-1 through 14</td>
<td>C</td>
<td>Same as Task A</td>
</tr>
<tr>
<td>10-1 through 14</td>
<td>D</td>
<td>Same as Task A</td>
</tr>
</tbody>
</table>

Bit 1: Mode: Continuous/inStep
This bit is used to select the jog mode for both coordinated and single-axis jogging. The mode takes effect when the next jog is started with a transition on the jog forward or jog reverse bit.
Bit 1 = 0, nStep jogging. Motion stops after the large or small distance is reached, or when the jog bit is set to 0.

Bit 1 = 1, continuous jogging. Motion stops when the travel limit is reached on an axis or when the jog bit is set to 0.

**Bit 2 and 3: Coordinated Jog Forward (bit 2) and Reverse (bit 3)**

Coordinated Motion: A low-to-high (0-1) transition on this bit while a task is in manual mode causes motion to start in the positive (Bit 2) or negative (Bit 3) direction in the coordinate selected in bits 9 to 14. A high-to-low (1-0) transition immediately stops the motion.

ELS Motion: A low-to-high (0-1) transition on this bit while a task is in manual mode causes motion to start on the ELS master in the positive (Bit 2) or negative (Bit 3) direction. All axes with the enable synchronized jog bit set in the axis control register will follow the master. A high-to-low (1-0) transition immediately stops the motion.

**Bits 4 and 5: Jog Type**

Bits 4 and 5 select the type of coordinated jogging for the next jog motion.

**World Jog** jogs the axes in the world coordinate selected by bits 9, 10, and 11.

**Joint jog** jogs individual axes, with the joint number determined in bits 9 through 14.

**Tool jog** jog axes in the tool coordinates selected with bits 9, 10, and 11. It is available only in future 6-axis robot versions.

<table>
<thead>
<tr>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Jog Type Selected</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>World Jog</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Joint Jog</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Tool Jog (6-axis robot version)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>not used</td>
</tr>
</tbody>
</table>

Selecting an invalid jog type will issue a warning message.
Bit 6: Distance/Speed (Large/small, Fast/Slow)
When bit 6 is set to 0 before a continuous jog, the slow jog speed is selected. For an incremental jog, the small distance and slow speed are selected.

When bit 6 is set to 1 before a continuous jog, the fast jog speed is selected. For an incremental jog, the large distance and fast speed are selected.

Bits 9 through 14 select the coordinate or joint that will be jogged when the jog forward or jog reverse bits are activated. Only one of these bits should be set, since jogging is allowed in only one coordinate at a time.

<table>
<thead>
<tr>
<th>Bit number</th>
<th>World and Tool Jog</th>
<th>Joint Jog</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>X Coordinate</td>
<td>Joint 1</td>
</tr>
<tr>
<td>10</td>
<td>Y Coordinate</td>
<td>Joint 2</td>
</tr>
<tr>
<td>11</td>
<td>Z Coordinate</td>
<td>Joint 3</td>
</tr>
<tr>
<td>12</td>
<td>Roll Axis (SAR only)</td>
<td>Joint 4 (SAR only)</td>
</tr>
<tr>
<td>13</td>
<td>Pitch Axis (SAR only)</td>
<td>Joint 5 (SAR only)</td>
</tr>
<tr>
<td>14</td>
<td>Yaw Axis (SAR only)</td>
<td>Joint 6 (SAR only)</td>
</tr>
</tbody>
</table>

When jogging in world coordinates, motion will be generated parallel to the selected X, Y, or Z coordinate according to bits 9 through 11. For example, setting bit 9 high and bits 10 and 11 low enables jogging parallel to the X axis.

For jogging of individual joints, the axis to jog is selected in bits 9 through 14. For example, if axes 2, 3, and 4 are used in coordinated motion, bit 9 selects axis 2, and bit 11 selects axis 4. The jog speed used is a percent of maximum axis velocity.

If an invalid jog type is selected, a warning message is issued.
2.6.6. Registers 11-18, 209-240: Axis Control

Registers 11-18 are for Axes 1-8. Registers 209-240 for Axes 9-40. These Registers are reserved for CLC axis control. These axes correspond to the DDS drive positions on the SERCOS fiber-optic communication loop. Any drive-controlled axis (single-axis, velocity, or ELS) may be jogged independently when its associated task is in manual mode. ELS axes can also be jogged with the master. See Chapter 4 - Coordinated Motion under the Setup Menu.

Jog mode (continuous or incremental) and jog speed or distance must be selected in the Task Jog Register for the task associated with the axis. Motion starts with a low-to-high transition on either the jog forward or the jog reverse bit. Motion stops when both jog bits are low, when both jog bits are high, when the task mode selection changes (e.g., from manual to automatic mode), or when a travel limit or incremental distance has been reached.

<table>
<thead>
<tr>
<th>Bit</th>
<th>Axis</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>Disable Axis</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>Jog Forward</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>Jog Reverse</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>Synchronized Jog</td>
</tr>
</tbody>
</table>

**Bit 1: Disable Axis**

When this bit is set to (1), all motion is disabled for this axis. The drive is immediately set to zero velocity and the position loop is disabled. Motion commands in the user program and from the control have no effect. Note that the drive can produce torque to hold position. It is still in the SERCOS ring. I-O and some diagnostics are still enabled.

When it is set to (0), motion is enabled, provided other conditions allow it and there are no errors on the drive or the control.

**Bit 2 and 3: Jog Forward (bit 2) and Reverse (bit 3)**

A low-to-high (0-1) transition on this bit while an axis is in manual mode causes motion to start in the positive (bit 2) or negative (bit 3) direction. A high-to-low (1-0) transition immediately stops the motion.

Motion is stopped when both jog bits are low, when both jog bits are high, when the task mode selection changes, or when a travel limit or incremental distance has been reached. Jog mode (continuous or incremental) and jog speed or distance are selected in the Task Jog Register for the task associated with the axis.
**Bit 4: Synchronized Jog**
ELS slave axes may be jogged independently or synchronously when their associated task is in manual mode.

When the Synchronized Jog bit is set to (0), the axis can be jogged independently. It does not follow the master, and it may be jogged continuously or incrementally in single-axis mode. Axes can be jogged individually to set up phase offsets or to prepare the machine for operation.

When the Synchronized Jog bit is set to (1), the slave axis follows the real or virtual master. This jog mode can be used to initially thread the material into sections of the machine, maintaining the position or velocity relation to the master. The master is jogged using the coordinated jog bits in the Task Jog register.
2.6.7. Register 21: System Status

The System Status register at CLC register address 21, is a read-only register dedicated to system status. The status is indicated by a high level in the appropriate bit.

<table>
<thead>
<tr>
<th>Register-Bit</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>21-1</td>
<td>Parameter Mode</td>
</tr>
<tr>
<td>21-4</td>
<td>Service Chan Ready</td>
</tr>
<tr>
<td>21-5</td>
<td>Error</td>
</tr>
<tr>
<td>21-9</td>
<td>Active Program LSB</td>
</tr>
<tr>
<td>21-10</td>
<td>Active Program Bit 2</td>
</tr>
<tr>
<td>21-11</td>
<td>Active Program Bit 3</td>
</tr>
<tr>
<td>21-12</td>
<td>Active Program MSB</td>
</tr>
<tr>
<td>21-13</td>
<td>TP Password Active</td>
</tr>
<tr>
<td>21-14</td>
<td>Teach Pendant</td>
</tr>
</tbody>
</table>

Bit 1: Parameter Mode/ Initializing
0 = Run Mode, 1 = Parameter Mode or Initializing System

If this bit is (1), the CLC is in parameter mode or the system is being initialized into run mode. If parameter mode is selected in System Control bit 1, the drives are in phase 2, access to restricted parameters is allowed, and the user programs are stopped.

If this bit is (0), the CLC is in run mode, and the user program is ready for operation if there are no errors.

Bit 4: Service Channel Ready
This bit can be checked by a user interface before initiating communication with a drive. When it is (0), the SERCOS ring is disconnected or phases are being switched. When it is (1), the drives are ready for service channel communication.

Bit 5: Error
This is the global error indicator for the CLC system. If the system, any task, or any drive has an error, this bit is set to (1). If there are no errors present, it is set to (0).

Bits 9-12: Active Program
These bits indicate the currently active program as a binary program number. (Bit 12 = most significant bit)

Example: The bits below indicate that program 5 is active.

Bit 9: 1 Bit 10: 0
Bit 11: 1 Bit 12: 0
Bit 13: Teach Pendant Password Active
This bit is set if the teach pendant password is active. It allows the user program or I-O mapper to disable functions while the corresponding functions are disabled in the teach pendant.

Bit 14: Teach Pendant Connected
This bit is set if the teach pendant is connected. On the CLC-V, the CTS line on the RS-232/422 connector is indicated in this bit. On the CLC-D and CLC-P, the ZT line on the cable is indicated. These lines are wired by default so that this bit indicates a connection.
2.6.8. Registers 22-25: Task Status

The reserved read-only registers 22, 23, 24 and 25 are dedicated to task status for Tasks A, B, C and D respectively. The condition of the task status registers may be read by a user task program to determine the state of the system and the task’s program execution.

<table>
<thead>
<tr>
<th>Register-Bit</th>
<th>Task</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>22-1</td>
<td>A</td>
<td>Mode: Auto/Manual</td>
</tr>
<tr>
<td>22-4</td>
<td></td>
<td>Single Step</td>
</tr>
<tr>
<td>22-5</td>
<td></td>
<td>Task Error</td>
</tr>
<tr>
<td>22-6</td>
<td></td>
<td>Task Running</td>
</tr>
<tr>
<td>22-10</td>
<td></td>
<td>Trace Ready</td>
</tr>
<tr>
<td>22-11</td>
<td></td>
<td>Breakpoint Reached</td>
</tr>
<tr>
<td>23-1 through 6</td>
<td>B</td>
<td>Same as Task A</td>
</tr>
<tr>
<td>24-1 through 6</td>
<td>C</td>
<td>Same as Task A</td>
</tr>
<tr>
<td>25-1 through 6</td>
<td>D</td>
<td>Same as Task A</td>
</tr>
</tbody>
</table>

**Bit 1: Mode: Automatic/Manual**

This bit is set to (1) by the CLC when the task is in automatic mode and is ready for the program to be started. It is set to (0) when manual mode is selected, an error is preventing the task from starting, or the task has not been initialized into automatic mode.

**Bit 4: Single Stepping**

When this bit is set to (1), the program flow has been stopped after an instruction, sequencer function, or sequencer step in single-step mode. To resume program flow, a (0-1) transition on the cycle start bit is required.

**Bit 5: Task Error**

This bit is set to (1) by the CLC when a task has an error or warning condition. An error is shown in parameter T-0-0122. The bit is (0) when no errors exist in this task.

**Bit 6: Task Running**

This bit is set to (1) by the CLC when the user task is executing program instructions, either from the main task or from an event. It is (0) when the task is not running.

**Bit 10: Trace Ready**

**Bit 11: Breakpoint Reached**

When this bit is set to (1), the breakpoint enabled in task parameter T-0-0137 and activated with Task Control Register bit 11 has been reached. Program flow has been stopped. To resume program flow, a (0-1) transition on the cycle start bit is required.
2.6.9. Registers 27 and 28: Eagle Module Inputs/Outputs (CLC-V Only)

Register 27 on CLC-V
These 12 inputs reside on the Eagle Module of the CLC-V 2.3 through the 25-pin connector on the front panel. They can be mapped to system bits with the I-O mapper or can be read in the user program.

Bits 1-8: Inputs1-8
Bit 9-12: Reserved for future functions
Bits 13-16 Always set to (0)

Register 28 on CLC-V
These four outputs reside on the Eagle Module of the CLC-V 2.3 through the 25-pin connector on the front panel. They can be mapped to system bits with the I-O mapper or can be written by the user program.

Bits 1-4: Outputs1-4
Bits 9-16 Available as flags (not connected to I-O)
2.6.10. Register 29: ELS Master Control Register

This register is functional only in CLC versions GPx-02.10 and greater.

Bit 4: Real Master: Primary/Secondary
This bit allows the ELS master to be switched between two real masters. The changeover can be done while the master is running.

When it is set to (0), the ELS and cam slave axes follow the primary real master selected in the user program and in parameter C-0-1000.

When it is set to (1), the ELS and cam slave axes follow the primary real master selected in the user program and in parameter C-0-1001.

Conditions for Master Switching:
When the master is switched between primary and secondary, the switch is not completed until the position of the two masters are within the ELS master switching threshold.

NOTE:
- This bit has no effect when no real master is configured in the program or parameters.
- The master switch is acknowledged by the CLC via the real master bit in the ELS status register.

Bit 5: Master Type: Virtual/nReal
This bit allows the ELS master to be switched between a real master and a virtual master.

When it is set to (0), the ELS and cam slave axes follow the real master that is selected in the user program or through system parameters C-0-0151 and C-0-0164.

When it is set to (1), the ELS and cam slave axes follow the virtual master. The master may then be stopped, started, or jogged without regard to the real master’s motion.

Conditions for Master Switching:
When the master is switched from virtual to real, the switch is not completed until the position of the virtual master equals the position of the real master. The virtual master must catch the real master. If the masters are at the same speed, it is necessary to slow down or speed up the virtual master in the user program.

When the master is switched from real to virtual, the switch immediately takes place, and the virtual master position is set to that of the real master.
NOTE:

- This bit has no effect when no real master is configured in the program or parameters.
- The master switch is acknowledged by the CLC via the master type bit in the ELS status register.
- The virtual master can be initialized with a velocity and/or a stop position before the master is switched. This allows a controlled stop to be performed on the slave axes by toggling this control bit.
- Using the I-O Mapper, this control bit can be tied to the task Auto/Manual bit or Cycle Stop bit, so that the user program can be stopped and started while the real master is running.
2.6.11. Register 30: ELS Master Status Register
This register is functional only in CLC versions GPx-02.10 and greater.

**Bit 4: Real Master: Primary/Secondary**
This status bit acknowledges the real master in the ELS master control register. When it is (0), the primary real master is active. When it is (1), the secondary master is active.

If the state of this bit does not match the corresponding bit in the control register, the master switch is still in progress. When switching to a real master, the master positions must be equal before the new master is activated.

**Bit 5: Master Type: Virtual/nReal**
This status bit acknowledges the master selection in the ELS master control register. When it is (0), the real master is active. When it is (1), the virtual master is active.

If the state of this bit does not match the corresponding bit in the control register, the master switch is still in progress. When switching to a real master, the master positions must be equal before the new master is activated.

**Bit 6: Virtual Master Enabled**
This bit is set to (1) whenever the ELS virtual master is ready to accept motion commands. In the current software version, the virtual master is always enabled in any ELS mode.

**Bit 7: Virtual Master at Programmed Speed**
This bit is set to (1) when the current virtual master speed is equal to the programmed velocity (C-0-0153 = C-0-0158). Otherwise, it is set to (0).

**Bit 9: Master Direction: Positive/ nNegative**
When the currently active master’s speed is greater than or equal to 0, this bit is set to (0). Otherwise, it is set to (1).

**Bit 10: Master Stopped**
When this bit is (1), the ELS master is stopped. When it is (0), the master is running.

It is set to (1) when the absolute value of the active master’s current velocity is less than the master zero velocity window: (|C-0-0158 | < C-0-0159).

Registers 31-38 are for Axes 1-8. Registers 309-340 (read only) for Axes 9-40. These registers are reserved for CLC axis status. These axes correspond to the DDS drive positions on the SERCOS fiber-optic communication loop.

<table>
<thead>
<tr>
<th>Bit</th>
<th>Axis</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>(1-40)</td>
<td>Jogging Forward</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>Jogging Reverse</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>Phase Adjusted</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>ELS Enabled</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>ELS Secondary Mode</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>Axis In-Position</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>Axis Aligned</td>
</tr>
<tr>
<td>11</td>
<td></td>
<td>Axis Halted</td>
</tr>
<tr>
<td>12</td>
<td></td>
<td>Class 3 Status</td>
</tr>
<tr>
<td>13</td>
<td></td>
<td>Class 2 Warning Change</td>
</tr>
<tr>
<td>14</td>
<td></td>
<td>Shutdown Error</td>
</tr>
<tr>
<td>15</td>
<td></td>
<td>Drive Ready LSB</td>
</tr>
<tr>
<td>16</td>
<td></td>
<td>Drive Ready MSB</td>
</tr>
</tbody>
</table>

**Bit 2 and Bit 3: Jogging Forward (Bit 2) and Reverse (Bit 3)**
This status bit is set to (1) when the axis is jogging forward (Bit 2) or reverse (Bit 3) through either the teach pendant or the I-O bits. Otherwise, it is set to (0).

**Bit 4: Phase Adjusted**
For ELS and Cam axes, this bit indicates if a phase adjustment move has been completed. During the phase adjust, it is set to (0), indicating the adjustment is in progress. When the phase adjust is complete, or when first synchronizing, this bit is set to (1).

**Bit 5: ELS Enabled**
When this bit is set to (1), the axis is in ELS or cam mode. When it is (0), it is in velocity or single-axis mode, as indicated by bit 6.

**Bit 6: Secondary Mode**
For ELS or cam axes, this indicates the current secondary mode that is enabled when ELS/cam is disabled. If it is set to (0), the axis is in single-axis mode. If it is set to (1), the axis is in velocity mode.

**Bit 7: Single Axis in Position**
Bit 7 is set to one (1) when the target position is reached and the axis is at zero velocity in single-axis mode. The drive associated with the axis must have its in-position window and zero velocity window parameters set correctly for this bit to function properly.
Bit 8: Axis Aligned (CLC cam axes only)
This bit provides the status of CLC based cam alignment. It is set to (1) if the axis is aligned to the cam, and (0) if it is not aligned. This allows user program logic to determine if an alignment move or phase offset is needed. This bit is only for CLC cam axes. It is not checked for Drive Cams.

The following conditions set this bit to (0):
- The axis is not configured in the program to be a cam axis.
- A valid cam is not active for this axis.
- The absolute value of (position of the axis - slave position from cam equation) is greater than the in-position window (drive parameter S-0-0057).

The following conditions set this bit to (1):
- The axis is synchronized to the master
- The absolute value of (position of the axis - slave position from cam equation) is less than or equal to the in-position window (drive parameter S-0-0057).

Bit 11: Axis Halted
Bit 11 is set to one (1) when the drive’s restart/nhalt bit is set to zero (0) (drive halted) and the axis is at zero velocity in single-axis mode.

Bit 12: Class 3 Status
This bit corresponds to the Change in Class 3 Diagnostics bit in the drive status word. When the diagnostic condition changes, the drive changes this bit from a (0) to a (1). The program can then check this register bit for a (0) to (1) transition, instead of continually reading the status parameter through the service channel.

Bit 13: Class 2 Warning Change
This bit corresponds to the Change in Class 2 Diagnostics bit in the drive status word. Warnings are often temporary conditions. This bit allows the program to latch and take action on a warning. When the diagnostic condition changes, the drive changes this bit from a (0) to a (1).

Bit 14: Drive Shutdown Error
Bit 14 is set to one (1) when there is a Class 1 Diagnostic Shutdown Error in the drive. This bit corresponds to the same bit in the drive’s SERCOS status register.
Bit 15, 16: Ready to Operate

Bits 15 and 16 indicate when a drive is ready to operate. When both bits are one (1) the drive will respond to motion commands. These bits correspond to the drive’s SERCOS status register.

<table>
<thead>
<tr>
<th>Bit 15</th>
<th>Bit 16</th>
<th>Description</th>
<th>DDS LED Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Drive not ready for power up</td>
<td>error or P1 - P3</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Drive ready for power up</td>
<td>bb</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Drive control and power sections ready</td>
<td>Ab</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Drive ready to operate</td>
<td>AH or AF</td>
</tr>
</tbody>
</table>

2.6.13. Registers 40-89: DEA I/O

Each DDS can hold up to three DEA cards, each DEA card has 15 inputs and 16 outputs. DEA cards are hardware configured to be card 1, 2, or 3, there order in the drive rack does not matter. DEA cards in drives 1 through 8 have predefined base registers, for cards in drives 9 through 99 a base register must be entered.

<table>
<thead>
<tr>
<th>Reg #</th>
<th>Name</th>
<th>Reg #</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>DDS1_1IN</td>
<td>46</td>
<td>DDS2_1IN</td>
</tr>
<tr>
<td>41</td>
<td>DDS1_1OUT</td>
<td>47</td>
<td>DDS2_1OUT</td>
</tr>
<tr>
<td>42</td>
<td>DDS1_2IN</td>
<td>48</td>
<td>DDS2_2IN</td>
</tr>
<tr>
<td>43</td>
<td>DDS1_2OUT</td>
<td>49</td>
<td>DDS2_2OUT</td>
</tr>
<tr>
<td>44</td>
<td>DDS1_3IN</td>
<td>50</td>
<td>DDS2_3IN</td>
</tr>
<tr>
<td>45</td>
<td>DDS1_3OUT</td>
<td>51</td>
<td>DDS2_3OUT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...87</td>
<td>DDS8_3OUT</td>
</tr>
</tbody>
</table>

2.6.14. Registers 90 and 91: Latch and Unlatch

Reserved Registers

Register 90 provides 16 latches that can be set to (1) directly or by the I-O mapper. The bits in register 90 can be reset to (0) only by writing a (1) to the corresponding bit in register 91. Register 91 immediately clears its bits to (0), so that a pulse is not needed on the unlatch bits.
### 2.6.15. Registers 95-97: Teach Pendant Status Registers

The bits in these registers are set when the corresponding keys are pressed on the teach pendant. They can be scanned in the user program or I-O mapper to detect system operations or to extend the teach pendant functionality to control the user program.

<table>
<thead>
<tr>
<th>95-1</th>
<th>96-1</th>
<th>97-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1 Key</td>
<td>Esc Key</td>
<td>Enter Key</td>
</tr>
<tr>
<td>95-2</td>
<td>96-2</td>
<td>97-2</td>
</tr>
<tr>
<td>F2 Key</td>
<td>4 Key</td>
<td>Decimal Key</td>
</tr>
<tr>
<td>95-3</td>
<td>96-3</td>
<td>97-3</td>
</tr>
<tr>
<td>F3 Key</td>
<td>5 Key</td>
<td>Zero Key</td>
</tr>
<tr>
<td>95-4</td>
<td>96-4</td>
<td>97-4</td>
</tr>
<tr>
<td>F4 Key</td>
<td>6 Key</td>
<td>Plus Minus Key</td>
</tr>
<tr>
<td>95-5</td>
<td>96-5</td>
<td>97-5</td>
</tr>
<tr>
<td>F5 Key</td>
<td>A Plus Key</td>
<td>C Plus Key</td>
</tr>
<tr>
<td>95-6</td>
<td>96-6</td>
<td>97-6</td>
</tr>
<tr>
<td>F6 Key</td>
<td>A Minus Key</td>
<td>C Minus Key</td>
</tr>
<tr>
<td>95-7</td>
<td>96-7</td>
<td>97-7</td>
</tr>
<tr>
<td>F7 Key</td>
<td>X Plus Key</td>
<td>Z Plus Key</td>
</tr>
<tr>
<td>95-8</td>
<td>96-8</td>
<td>97-8</td>
</tr>
<tr>
<td>F8 Key</td>
<td>X Minus Key</td>
<td>Z Minus Key</td>
</tr>
<tr>
<td>95-9</td>
<td>96-9</td>
<td></td>
</tr>
<tr>
<td>Help Key</td>
<td>Edit Key</td>
<td></td>
</tr>
<tr>
<td>95-10</td>
<td>96-10</td>
<td></td>
</tr>
<tr>
<td>7 Key</td>
<td>1 Key</td>
<td></td>
</tr>
<tr>
<td>95-11</td>
<td>96-11</td>
<td></td>
</tr>
<tr>
<td>8 Key</td>
<td>2 Key</td>
<td></td>
</tr>
<tr>
<td>95-12</td>
<td>96-12</td>
<td></td>
</tr>
<tr>
<td>9 Key</td>
<td>3 Key</td>
<td></td>
</tr>
<tr>
<td>95-13</td>
<td>96-13</td>
<td></td>
</tr>
<tr>
<td>Teach Key</td>
<td>B Plus Key</td>
<td></td>
</tr>
<tr>
<td>95-14</td>
<td>96-14</td>
<td></td>
</tr>
<tr>
<td>Task Key</td>
<td>B Minus Key</td>
<td></td>
</tr>
<tr>
<td>95-15</td>
<td>96-15</td>
<td></td>
</tr>
<tr>
<td>Axis Plus Key</td>
<td>Y Plus Key</td>
<td></td>
</tr>
<tr>
<td>95-16</td>
<td>96-16</td>
<td></td>
</tr>
<tr>
<td>Axis Minus Key</td>
<td>Y Minus Key</td>
<td></td>
</tr>
</tbody>
</table>
### 2.6.16. Registers 98, 99: Teach Pendant Control - Task A-B, C-D

The bits in the register can disable teach pendant control of the selected function for the corresponding tasks A - B, C-D

<table>
<thead>
<tr>
<th>98-1</th>
<th>Block Task A Manual</th>
<th>99-1</th>
<th>Block Task C Manual</th>
</tr>
</thead>
<tbody>
<tr>
<td>98-2</td>
<td>Block Task A Auto</td>
<td>99-2</td>
<td>Block Task C Auto</td>
</tr>
<tr>
<td>98-3</td>
<td>Block Task A Step</td>
<td>99-3</td>
<td>Block Task C Step</td>
</tr>
<tr>
<td>98-4</td>
<td>Block Task A Jog</td>
<td>99-4</td>
<td>Block Task C Jog</td>
</tr>
<tr>
<td>98-5</td>
<td>Block Task A Entry</td>
<td>99-5</td>
<td>Block Task C Entry</td>
</tr>
<tr>
<td>98-6</td>
<td>Block Task A Teach</td>
<td>99-6</td>
<td>Block Task C Teach</td>
</tr>
<tr>
<td>98-10</td>
<td>Block Task B Auto</td>
<td>99-10</td>
<td>Block Task D Auto</td>
</tr>
<tr>
<td>98-11</td>
<td>Block Task B Step</td>
<td>99-11</td>
<td>Block Task D Step</td>
</tr>
<tr>
<td>98-12</td>
<td>Block Task B Jog</td>
<td>99-12</td>
<td>Block Task D Jog</td>
</tr>
<tr>
<td>98-13</td>
<td>Block Task B Entry</td>
<td>99-13</td>
<td>Block Task D Entry</td>
</tr>
<tr>
<td>98-14</td>
<td>Block Task B Teach</td>
<td>99-14</td>
<td>Block Task D Teach</td>
</tr>
</tbody>
</table>
2.6.17. CLC Reserved Register Tables

Indramat reserves the first 99 registers for managing the CLC system, tasks, axes and DDS drives. User available I/O registers begin with register 100.

The following tables define the Indramat reserved registers and bits, and their associated default CLC label names.

<table>
<thead>
<tr>
<th>Register</th>
<th>Register Label Name</th>
<th>Bit</th>
<th>Bit Label Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>System_Control</td>
<td>1</td>
<td>Parameter_Mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>Bit_02</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>nEmergency_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>Bit_04</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>Clear_All_Errors</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6</td>
<td>Pendant_Live_Man (switch)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>7</td>
<td>Bit_07</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8</td>
<td>Activate_Program</td>
</tr>
<tr>
<td></td>
<td></td>
<td>9</td>
<td>Program_Select_LSB</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10</td>
<td>Program_Select_Bit_2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>11</td>
<td>Program_Select_Bit_3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>12</td>
<td>Program_Select_MSB</td>
</tr>
<tr>
<td></td>
<td></td>
<td>13</td>
<td>Bit_13</td>
</tr>
<tr>
<td></td>
<td></td>
<td>14</td>
<td>Pendant_Enable</td>
</tr>
<tr>
<td></td>
<td></td>
<td>15</td>
<td>Pendant_Level_LSB</td>
</tr>
<tr>
<td></td>
<td></td>
<td>16</td>
<td>Pendant_Level_MSB</td>
</tr>
<tr>
<td>Register</td>
<td>Register Label Name</td>
<td>Bit</td>
<td>Bit Label Name</td>
</tr>
<tr>
<td>----------</td>
<td>----------------------</td>
<td>-----</td>
<td>-------------------------</td>
</tr>
<tr>
<td>02</td>
<td>TaskA_Co ntrol</td>
<td>1</td>
<td>Mode_Auto_nManual</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>Bit_02</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>Bit_03</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>Single_Step</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>Bit_05</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6</td>
<td>Cycle_Start/Resume</td>
</tr>
<tr>
<td></td>
<td></td>
<td>7</td>
<td>nTask_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8</td>
<td>Bit_08</td>
</tr>
<tr>
<td></td>
<td></td>
<td>9</td>
<td>Task_Event_Trigger</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10</td>
<td>Bit_10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>11</td>
<td>Breakpoint_Enable</td>
</tr>
<tr>
<td></td>
<td></td>
<td>12</td>
<td>Step_Seq_Step</td>
</tr>
<tr>
<td></td>
<td></td>
<td>13</td>
<td>Step_Seq_Function</td>
</tr>
<tr>
<td></td>
<td></td>
<td>14</td>
<td>Bit_14</td>
</tr>
<tr>
<td></td>
<td></td>
<td>15</td>
<td>Bit_15</td>
</tr>
<tr>
<td></td>
<td></td>
<td>16</td>
<td>Bit_16</td>
</tr>
<tr>
<td>03</td>
<td>TaskB_C o ntrol</td>
<td></td>
<td>(Same as Task A)</td>
</tr>
<tr>
<td>04</td>
<td>TaskC_C o ntrol</td>
<td></td>
<td>(Same as Task A)</td>
</tr>
<tr>
<td>05</td>
<td>TaskD_C o ntrol</td>
<td></td>
<td>(Same as Task A)</td>
</tr>
<tr>
<td>06</td>
<td>System_Diagnostic_C ode</td>
<td>1-16</td>
<td>Bit_1-Bit_16</td>
</tr>
<tr>
<td>07</td>
<td>TaskA_Jog</td>
<td>1</td>
<td>Continuous_nStep</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>Coord_Jog_Forward</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>Coord_Jog_Reverse</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>Jog_Type_LSB</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>Jog_Type_MSB</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6</td>
<td>Distance_Speed</td>
</tr>
<tr>
<td></td>
<td></td>
<td>7</td>
<td>Bit_07</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8</td>
<td>Bit_08</td>
</tr>
<tr>
<td></td>
<td></td>
<td>9</td>
<td>Jog_X_Coord</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10</td>
<td>Jog_Y_Coord</td>
</tr>
<tr>
<td></td>
<td></td>
<td>11</td>
<td>Jog_Z_Coord</td>
</tr>
<tr>
<td></td>
<td></td>
<td>12</td>
<td>Jog_Joint_4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>13</td>
<td>Jog_Joint_5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>14</td>
<td>Jog_Joint_6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>15</td>
<td>Bit_15</td>
</tr>
<tr>
<td></td>
<td></td>
<td>16</td>
<td>Bit_16</td>
</tr>
<tr>
<td>08</td>
<td>TaskB_Jog</td>
<td></td>
<td>(Same as Task A)</td>
</tr>
<tr>
<td>09</td>
<td>TaskC_Jog</td>
<td></td>
<td>(Same as Task A)</td>
</tr>
<tr>
<td>10</td>
<td>TaskD_Jog</td>
<td></td>
<td>(Same as Task A)</td>
</tr>
<tr>
<td>Register</td>
<td>Register Label Name</td>
<td>Bit</td>
<td>Bit Label Name</td>
</tr>
<tr>
<td>----------</td>
<td>---------------------</td>
<td>-----</td>
<td>----------------------------</td>
</tr>
<tr>
<td>11</td>
<td>Axis1 Control</td>
<td>1</td>
<td>Disable Axis</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>Jog Forward</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>Jog Reverse</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>Synchronized Jog</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>Bit 05</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6</td>
<td>Bit 06</td>
</tr>
<tr>
<td></td>
<td></td>
<td>7</td>
<td>Bit 07</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8</td>
<td>Bit 08</td>
</tr>
<tr>
<td></td>
<td></td>
<td>9</td>
<td>Bit 09</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10</td>
<td>Bit 10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>11</td>
<td>Bit 11</td>
</tr>
<tr>
<td></td>
<td></td>
<td>12</td>
<td>Bit 12</td>
</tr>
<tr>
<td></td>
<td></td>
<td>13</td>
<td>Bit 13</td>
</tr>
<tr>
<td></td>
<td></td>
<td>14</td>
<td>Bit 14</td>
</tr>
<tr>
<td></td>
<td></td>
<td>15</td>
<td>Bit 15</td>
</tr>
<tr>
<td></td>
<td></td>
<td>16</td>
<td>Bit 16</td>
</tr>
<tr>
<td>12</td>
<td>Axis2 Control</td>
<td></td>
<td>(Same as Axis 1)</td>
</tr>
<tr>
<td>13</td>
<td>Axis3 Control</td>
<td></td>
<td>(Same as Axis 1)</td>
</tr>
<tr>
<td>14</td>
<td>Axis4 Control</td>
<td></td>
<td>(Same as Axis 1)</td>
</tr>
<tr>
<td>15</td>
<td>Axis5 Control</td>
<td></td>
<td>(Same as Axis 1)</td>
</tr>
<tr>
<td>16</td>
<td>Axis6 Control</td>
<td></td>
<td>(Same as Axis 1)</td>
</tr>
<tr>
<td>17</td>
<td>Axis7 Control</td>
<td></td>
<td>(Same as Axis 1)</td>
</tr>
<tr>
<td>18</td>
<td>Axis8 Control</td>
<td></td>
<td>(Same as Axis 1)</td>
</tr>
<tr>
<td>Register</td>
<td>Register Label Name</td>
<td>Bit</td>
<td>Bit Label Name</td>
</tr>
<tr>
<td>----------</td>
<td>---------------------</td>
<td>-----</td>
<td>------------------------------</td>
</tr>
<tr>
<td>21</td>
<td>System_Status</td>
<td>1</td>
<td>Parameter_Mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>Bit_02</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>Bit_03</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>Service_Chan_Ready</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>Error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6</td>
<td>Bit_06</td>
</tr>
<tr>
<td></td>
<td></td>
<td>7</td>
<td>Bit_07</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8</td>
<td>Bit_08</td>
</tr>
<tr>
<td></td>
<td></td>
<td>9</td>
<td>Active_Program_LSB</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10</td>
<td>Active_Program_Bit_2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>11</td>
<td>Active_Program_Bit_3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>12</td>
<td>Active_Program_Bit_MSB</td>
</tr>
<tr>
<td></td>
<td></td>
<td>13</td>
<td>TP_Password_Active</td>
</tr>
<tr>
<td></td>
<td></td>
<td>14</td>
<td>Teach_Pendant</td>
</tr>
<tr>
<td></td>
<td></td>
<td>15</td>
<td>Bit_15</td>
</tr>
<tr>
<td></td>
<td></td>
<td>16</td>
<td>Bit_16</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Register</th>
<th>Register Label Name</th>
<th>Bit</th>
<th>Bit Label Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>TaskA_Status</td>
<td>1</td>
<td>Mode:Auto/nManual</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>Bit_02</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>Bit_03</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>Single_Step</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>Task_Error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6</td>
<td>Task_Running</td>
</tr>
<tr>
<td></td>
<td></td>
<td>7</td>
<td>Bit_07</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8</td>
<td>Bit_08</td>
</tr>
<tr>
<td></td>
<td></td>
<td>9</td>
<td>Bit_09</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10</td>
<td>Trace_Ready</td>
</tr>
<tr>
<td></td>
<td></td>
<td>11</td>
<td>Breakpoint_Reached</td>
</tr>
<tr>
<td></td>
<td></td>
<td>12</td>
<td>Bit_12</td>
</tr>
<tr>
<td></td>
<td></td>
<td>13</td>
<td>Bit_13</td>
</tr>
<tr>
<td></td>
<td></td>
<td>14</td>
<td>Bit_14</td>
</tr>
<tr>
<td></td>
<td></td>
<td>15</td>
<td>Bit_15</td>
</tr>
<tr>
<td></td>
<td></td>
<td>16</td>
<td>Bit_16</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Register</th>
<th>Register Label Name</th>
<th>Bit</th>
<th>Bit Label Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>TaskB_Status</td>
<td></td>
<td>(Same as Task A)</td>
</tr>
<tr>
<td>24</td>
<td>TaskC_Status</td>
<td></td>
<td>(Same as Task A)</td>
</tr>
<tr>
<td>25</td>
<td>TaskD_Status</td>
<td></td>
<td>(Same as Task A)</td>
</tr>
<tr>
<td>Register</td>
<td>Register Label Name</td>
<td>Bit</td>
<td>Bit Label Name</td>
</tr>
<tr>
<td>----------</td>
<td>----------------------------</td>
<td>-------</td>
<td>---------------------------------------</td>
</tr>
<tr>
<td>27</td>
<td>Eagle Module Inputs</td>
<td>1-8</td>
<td>Input1-8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>9-12</td>
<td>Future Speed Function</td>
</tr>
<tr>
<td>28</td>
<td>Eagle Module Outputs</td>
<td>1-4</td>
<td>Output1-4</td>
</tr>
<tr>
<td>29</td>
<td>ELS Master Control</td>
<td>4</td>
<td>Real Master: Primary/Secondary</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>Master Type: Virtual/nReal</td>
</tr>
<tr>
<td>30</td>
<td>ELS Master Status</td>
<td>4</td>
<td>Real Master: Primary/Secondary</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>Master Type: Virtual/nReal</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6</td>
<td>Virtual Master Enabled</td>
</tr>
<tr>
<td></td>
<td></td>
<td>7</td>
<td>Virtual Master at Programmed Speed</td>
</tr>
<tr>
<td></td>
<td></td>
<td>9</td>
<td>Master Direction: Positive/nNegative</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10</td>
<td>Master Stopped</td>
</tr>
<tr>
<td>31</td>
<td>Axis1_Status</td>
<td>1</td>
<td>Bit _01</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>Jogging_Forward</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>Jogging_Reverse</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>Phase_Adjusted</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>ELS_Enabled</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6</td>
<td>ELS_Secondary_Mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td>7</td>
<td>Axis_in_Position</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8</td>
<td>Axis_Aligned</td>
</tr>
<tr>
<td></td>
<td></td>
<td>9</td>
<td>Bit _09</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10</td>
<td>Bit _10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>11</td>
<td>Axis_Halted</td>
</tr>
<tr>
<td></td>
<td></td>
<td>12</td>
<td>Class _3_Status</td>
</tr>
<tr>
<td></td>
<td></td>
<td>13</td>
<td>Class _2_Warning</td>
</tr>
<tr>
<td></td>
<td></td>
<td>14</td>
<td>Shutdown_Error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>15</td>
<td>Drive_Ready_LSB</td>
</tr>
<tr>
<td></td>
<td></td>
<td>16</td>
<td>Drive_Ready_MSB</td>
</tr>
<tr>
<td>32</td>
<td>Axis2_Status</td>
<td></td>
<td>(Same as Axis 1)</td>
</tr>
<tr>
<td>33</td>
<td>Axis3_Status</td>
<td></td>
<td>(Same as Axis 1)</td>
</tr>
<tr>
<td>34</td>
<td>Axis4_Status</td>
<td></td>
<td>(Same as Axis 1)</td>
</tr>
<tr>
<td>35</td>
<td>Axis5_Status</td>
<td></td>
<td>(Same as Axis 1)</td>
</tr>
<tr>
<td>36</td>
<td>Axis6_Status</td>
<td></td>
<td>(Same as Axis 1)</td>
</tr>
<tr>
<td>37</td>
<td>Axis7_Status</td>
<td></td>
<td>(Same as Axis 1)</td>
</tr>
<tr>
<td>38</td>
<td>Axis8_Status</td>
<td></td>
<td>(Same as Axis 1)</td>
</tr>
<tr>
<td>Register</td>
<td>Register Label Name</td>
<td>Bit</td>
<td>Bit Label Name</td>
</tr>
<tr>
<td>----------</td>
<td>---------------------</td>
<td>-------</td>
<td>----------------</td>
</tr>
<tr>
<td>40</td>
<td>DDS1 _1IN</td>
<td>1</td>
<td>Bit _11</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>16</td>
<td>Bit _16</td>
</tr>
<tr>
<td>41</td>
<td>DDS1 _1OUT</td>
<td></td>
<td>(same)</td>
</tr>
<tr>
<td>42</td>
<td>DDS1 _2IN</td>
<td></td>
<td>(same)</td>
</tr>
<tr>
<td>43</td>
<td>DDS1 _2OUT</td>
<td></td>
<td>(same)</td>
</tr>
<tr>
<td>44</td>
<td>DDS1 _3IN</td>
<td></td>
<td>(same)</td>
</tr>
<tr>
<td>45</td>
<td>DDS1 _3OUT</td>
<td></td>
<td>(same)</td>
</tr>
</tbody>
</table>

Registers 40 through 45 access inputs and outputs provided by Indramat DDS I/O cards installed in the DDS drive 1 enclosure.

<table>
<thead>
<tr>
<th>Register</th>
<th>Register Label Name</th>
<th>Bit</th>
<th>Bit Label Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>46</td>
<td>DDS2 _1IN</td>
<td>1</td>
<td>Bit _11</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>16</td>
<td>Bit _16</td>
</tr>
<tr>
<td>47</td>
<td>DDS2 _1OUT</td>
<td></td>
<td>(same)</td>
</tr>
<tr>
<td>48</td>
<td>DDS2 _2IN</td>
<td></td>
<td>(same)</td>
</tr>
<tr>
<td>49</td>
<td>DDS2 _2OUT</td>
<td></td>
<td>(same)</td>
</tr>
<tr>
<td>50</td>
<td>DDS2 _3IN</td>
<td></td>
<td>(same)</td>
</tr>
<tr>
<td>51</td>
<td>DDS2 _3OUT</td>
<td></td>
<td>(same)</td>
</tr>
</tbody>
</table>

Registers 46 through 51 access inputs and outputs provided by Indramat DDS I/O cards installed in the DDS drive 2 enclosure.
### Register Table

<table>
<thead>
<tr>
<th>Register</th>
<th>Register Label Name</th>
<th>Bit</th>
<th>Bit Label Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>52</td>
<td>DDS3 _IIN</td>
<td>1</td>
<td>Bit_01</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>16</td>
<td>Bit_16</td>
</tr>
<tr>
<td>53</td>
<td>DDS3 -IOUT</td>
<td></td>
<td>(same)</td>
</tr>
<tr>
<td>54</td>
<td>DDS3 _IIN</td>
<td></td>
<td>(same)</td>
</tr>
<tr>
<td>55</td>
<td>DDS3 2OUT</td>
<td></td>
<td>(same)</td>
</tr>
<tr>
<td>56</td>
<td>DDS3 3IN</td>
<td></td>
<td>(same)</td>
</tr>
<tr>
<td>57</td>
<td>DDS3 3OUT</td>
<td></td>
<td>(same)</td>
</tr>
</tbody>
</table>

Registers 52 through 57 access inputs and outputs provided by Indramat DDS I/O cards installed in the DDS drive 3 enclosure.

<table>
<thead>
<tr>
<th>Register</th>
<th>Register Label Name</th>
<th>Bit</th>
<th>Bit Label Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>58</td>
<td>DDS 4 _IIN</td>
<td>1</td>
<td>Bit_01</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>16</td>
<td>Bit_16</td>
</tr>
<tr>
<td>59</td>
<td>DDS 4-IOUT</td>
<td></td>
<td>(same)</td>
</tr>
<tr>
<td>60</td>
<td>DDS 4 2IN</td>
<td></td>
<td>(same)</td>
</tr>
<tr>
<td>61</td>
<td>DDS 4 2OUT</td>
<td></td>
<td>(same)</td>
</tr>
<tr>
<td>62</td>
<td>DDS 4 3IN</td>
<td></td>
<td>(same)</td>
</tr>
<tr>
<td>63</td>
<td>DDS 4 3OUT</td>
<td></td>
<td>(same)</td>
</tr>
</tbody>
</table>

Registers 58 through 63 access inputs and outputs provided by Indramat DDS I/O cards installed in the DDS drive 4 enclosure.
<table>
<thead>
<tr>
<th>Register</th>
<th>Register Label Name</th>
<th>Bit</th>
<th>Bit Label Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>DDS 5_1IN</td>
<td>1</td>
<td>Bit_01</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>65</td>
<td>DDS 5_1OUT</td>
<td></td>
<td>(same)</td>
</tr>
<tr>
<td>66</td>
<td>DDS 5_2IN</td>
<td></td>
<td>(same)</td>
</tr>
<tr>
<td>67</td>
<td>DDS 5_2OUT</td>
<td></td>
<td>(same)</td>
</tr>
<tr>
<td>68</td>
<td>DDS 5_3IN</td>
<td></td>
<td>(same)</td>
</tr>
<tr>
<td>69</td>
<td>DDS 5_3OUT</td>
<td></td>
<td>(same)</td>
</tr>
</tbody>
</table>

Registers 64 through 69 access inputs and outputs provided by Indramat DDS I/O cards installed in the DDS drive 5 enclosure.

<table>
<thead>
<tr>
<th>Register</th>
<th>Register Label Name</th>
<th>Bit</th>
<th>Bit Label Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>70</td>
<td>DDS 6_1IN</td>
<td>1</td>
<td>Bit_01</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>71</td>
<td>DDS 6_1OUT</td>
<td></td>
<td>(same)</td>
</tr>
<tr>
<td>72</td>
<td>DDS 6_2IN</td>
<td></td>
<td>(same)</td>
</tr>
<tr>
<td>73</td>
<td>DDS 6_2OUT</td>
<td></td>
<td>(same)</td>
</tr>
<tr>
<td>74</td>
<td>DDS 6_3IN</td>
<td></td>
<td>(same)</td>
</tr>
<tr>
<td>75</td>
<td>DDS 6_3OUT</td>
<td></td>
<td>(same)</td>
</tr>
</tbody>
</table>

Registers 70 through 75 access inputs and outputs provided by Indramat DDS I/O cards installed in the DDS drive 6 enclosure.
<table>
<thead>
<tr>
<th>Register</th>
<th>Register Label Name</th>
<th>Bit</th>
<th>Bit Label Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>76</td>
<td>DDS 7_1IN</td>
<td>1</td>
<td>Bit_01</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>16</td>
<td>Bit_16</td>
</tr>
<tr>
<td>77</td>
<td>DDS 7-1OUT</td>
<td></td>
<td>(same)</td>
</tr>
<tr>
<td>78</td>
<td>DDS 7_2IN</td>
<td></td>
<td>(same)</td>
</tr>
<tr>
<td>79</td>
<td>DDS 7_2OUT</td>
<td></td>
<td>(same)</td>
</tr>
<tr>
<td>80</td>
<td>DDS 7_3IN</td>
<td></td>
<td>(same)</td>
</tr>
<tr>
<td>81</td>
<td>DDS 7_3OUT</td>
<td></td>
<td>(same)</td>
</tr>
</tbody>
</table>

Registers 76 through 81 access inputs and outputs provided by Indramat DDS I/O cards installed in the DDS drive 7 enclosure.

<table>
<thead>
<tr>
<th>Register</th>
<th>Register Label Name</th>
<th>Bit</th>
<th>Bit Label Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>82</td>
<td>DDS 8_1IN</td>
<td>1</td>
<td>Bit_01</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>16</td>
<td>Bit_16</td>
</tr>
<tr>
<td>83</td>
<td>DDS 8-1OUT</td>
<td></td>
<td>(same)</td>
</tr>
<tr>
<td>84</td>
<td>DDS 8_2IN</td>
<td></td>
<td>(same)</td>
</tr>
<tr>
<td>85</td>
<td>DDS 8_2OUT</td>
<td></td>
<td>(same)</td>
</tr>
<tr>
<td>86</td>
<td>DDS 8_3IN</td>
<td></td>
<td>(same)</td>
</tr>
<tr>
<td>87</td>
<td>DDS 8_3OUT</td>
<td></td>
<td>(same)</td>
</tr>
</tbody>
</table>

Registers 82 through 87 access inputs and outputs provided by Indramat DDS I/O cards installed in the DDS drive 8 enclosure.
<table>
<thead>
<tr>
<th>Register</th>
<th>Register Label Name</th>
<th>Bit</th>
<th>Bit Label Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>90</td>
<td>Latch</td>
<td>1</td>
<td>Bit_01</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>16</td>
<td>Bit_16</td>
</tr>
<tr>
<td>91</td>
<td>Unlatch</td>
<td></td>
<td>(same)</td>
</tr>
<tr>
<td>95</td>
<td>Teach Pendant Status</td>
<td>1</td>
<td>F1 Key</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>F2 Key</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>F3 Key</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>F4 Key</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>F5 Key</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6</td>
<td>F6 Key</td>
</tr>
<tr>
<td></td>
<td></td>
<td>7</td>
<td>F7 Key</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8</td>
<td>F8 Key</td>
</tr>
<tr>
<td></td>
<td></td>
<td>9</td>
<td>Help Key</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10</td>
<td>7 Key</td>
</tr>
<tr>
<td></td>
<td></td>
<td>11</td>
<td>8 Key</td>
</tr>
<tr>
<td></td>
<td></td>
<td>12</td>
<td>9 Key</td>
</tr>
<tr>
<td></td>
<td></td>
<td>13</td>
<td>Teach Key</td>
</tr>
<tr>
<td></td>
<td></td>
<td>14</td>
<td>Task Key</td>
</tr>
<tr>
<td></td>
<td></td>
<td>15</td>
<td>Axis Plus Key</td>
</tr>
<tr>
<td></td>
<td></td>
<td>16</td>
<td>Axis Minus Key</td>
</tr>
<tr>
<td>96</td>
<td>Teach Pendant Status</td>
<td>1</td>
<td>Esc Key</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>4 Key</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>5 Key</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>6 Key</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>A Plus Key</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6</td>
<td>A Minus Key</td>
</tr>
<tr>
<td></td>
<td></td>
<td>7</td>
<td>X Plus Key</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8</td>
<td>X Minus Key</td>
</tr>
<tr>
<td></td>
<td></td>
<td>9</td>
<td>Edit Key</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10</td>
<td>1 Key</td>
</tr>
<tr>
<td></td>
<td></td>
<td>11</td>
<td>2 Key</td>
</tr>
<tr>
<td></td>
<td></td>
<td>12</td>
<td>3 Key</td>
</tr>
<tr>
<td></td>
<td></td>
<td>13</td>
<td>B Plus Key</td>
</tr>
<tr>
<td></td>
<td></td>
<td>14</td>
<td>B Minus Key</td>
</tr>
<tr>
<td></td>
<td></td>
<td>15</td>
<td>Y Plus Key</td>
</tr>
<tr>
<td></td>
<td></td>
<td>16</td>
<td>Y Minus Key</td>
</tr>
<tr>
<td>Register</td>
<td>Register Label Name</td>
<td>Bit</td>
<td>Bit Label Name</td>
</tr>
<tr>
<td>----------</td>
<td>----------------------</td>
<td>-----</td>
<td>----------------------</td>
</tr>
<tr>
<td>97</td>
<td>Teach Pendant Status</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>Enter Key</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>Decimal Key</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>Zero Key</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>Plus Minus Key</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>C Plus Key</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6</td>
<td>C Minus Key</td>
</tr>
<tr>
<td></td>
<td></td>
<td>7</td>
<td>Z Plus Key</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8</td>
<td>Z Minus Key</td>
</tr>
<tr>
<td>98</td>
<td>Teach Pendant Control</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>Block Task A Manual</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>Block Task A Auto</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>Block Task A Step</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>Block Task A Jog</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>Block Task A Entry</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6</td>
<td>Block Task A Teach</td>
</tr>
<tr>
<td></td>
<td></td>
<td>7</td>
<td>Bit 7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8</td>
<td>Bit 8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>9</td>
<td>Block Task B Manual</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10</td>
<td>Block Task B Auto</td>
</tr>
<tr>
<td></td>
<td></td>
<td>11</td>
<td>Block Task B Step</td>
</tr>
<tr>
<td></td>
<td></td>
<td>12</td>
<td>Block Task B Jog</td>
</tr>
<tr>
<td></td>
<td></td>
<td>13</td>
<td>Block Task B Entry</td>
</tr>
<tr>
<td></td>
<td></td>
<td>14</td>
<td>Block Task B Teach</td>
</tr>
<tr>
<td></td>
<td></td>
<td>15</td>
<td>Bit 15</td>
</tr>
<tr>
<td></td>
<td></td>
<td>16</td>
<td>Bit 16</td>
</tr>
<tr>
<td>99</td>
<td>Teach Pendant Control</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>Block Task C Manual</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>Block Task C Auto</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>Block Task C Step</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>Block Task C Jog</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>Block Task C Entry</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6</td>
<td>Block Task C Teach</td>
</tr>
<tr>
<td></td>
<td></td>
<td>7</td>
<td>Bit 7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8</td>
<td>Bit 8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>9</td>
<td>Block Task D Manual</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10</td>
<td>Block Task D Auto</td>
</tr>
<tr>
<td></td>
<td></td>
<td>11</td>
<td>Block Task D Step</td>
</tr>
<tr>
<td></td>
<td></td>
<td>12</td>
<td>Block Task D Jog</td>
</tr>
<tr>
<td></td>
<td></td>
<td>13</td>
<td>Block Task D Entry</td>
</tr>
<tr>
<td></td>
<td></td>
<td>14</td>
<td>Block Task D Teach</td>
</tr>
<tr>
<td></td>
<td></td>
<td>15</td>
<td>Bit 15</td>
</tr>
<tr>
<td></td>
<td></td>
<td>16</td>
<td>Bit 16</td>
</tr>
<tr>
<td>Register</td>
<td>Register Label Name</td>
<td>Bit</td>
<td>Bit Label Name</td>
</tr>
<tr>
<td>----------</td>
<td>---------------------</td>
<td>-----</td>
<td>---------------</td>
</tr>
<tr>
<td>100</td>
<td>User Inputs Reg 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>101</td>
<td>User Inputs Reg 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>These registers are reserved if an Indramat supported I/O subsystem and physical I/O drivers are used. For specific assignments see the appropriate I/O system section. If Indramat I/O drivers are not used, the registers are available to the user.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Register</th>
<th>Register Label Name</th>
<th>Bit</th>
<th>Bit Label Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>120</td>
<td>User Outputs Reg 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>121</td>
<td>User Outputs Reg 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>These registers are reserved if an Indramat supported I/O subsystem and physical I/O drivers are used. For specific assignments see the appropriate I/O system section. If Indramat I/O drivers are not used, the registers are available to the user.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
CHAPTER 3. PARAMETERS

3.1. Overview

The CLC card uses system-builder parameters to adapt to a specific application. Basic communication and initialization parameters for the CLC and DDS digital drives must be entered before it is possible to operate or program a CLC system.

System-builders must enter certain parameters describing the mechanical characteristics of their unique application. Parameters may be specific to an application and/or a single machine installation. These parameters specify the machine’s limitations such as maximum velocity and acceleration. Other parameters are used to specify the mechanical characteristics of the system, such as the ratio between motor revolutions and shaft rotation or slide travel.

The CLC system divides parameters into four functionally related classes: System (Card), Task, Axis and Drive. The System parameters contain setup and status parameters for the CLC card. The Task and Axis parameters are associated with user programs. The System, Task and Axis parameters are downloaded and stored in the CLC card. Digital Drive parameters are stored in each SERCOS-compatible digital drive and are needed to configure a motion system.

Each parameter within a Class also has a set and number identifier. All parameter elements (data, name, units) can be accessed using the Visual Motion Windows application or from a user specified interface such as an ASCII terminal (see Appendix B. Direct ASCII Communication). Parameters in this chapter are described using the following format:

Parameter Identification
All parameters on the CLC have an identification number (IDN): X-s-nnnn

Where X = parameter type:
C = CLC System Parameter
T = CLC Task Parameter
A = CLC Axis Parameter
S = Drive SERCOS Standard Parameter
P = Drive Product-specific Parameter
s = parameter set, if drive has different sets, a number between 0 and 7
nnnn = parameter number from 1 to 4095.

Examples:
C-0-0001, is CLC system parameter 1
P-1-0001, is a product-specific drive 1, parameter 1.
Class 'C' includes the system setup and status parameters for the CLC card. Certain status parameters are stored on the CLC card in non-volatile memory, while other status parameters are only temporary indicators and are lost when the system is powered off. Class 'C' parameters have only one set and are indicated by the number '0'. (Examples: C-0-0nnn, etc.)

Class 'T' parameters include setup and status information for each task. One of the four task parameter sets is indicated by an ASCII character 'A' through 'D' corresponding to one of the user program tasks: A, B, C or D. (Examples: Task A 01 T-0-nnnn, Task C 03 T-0-nnnn, etc.)

Class 'A' includes the parameters used to configure and display information about the CLC Axes. A maximum of 40 different sets of parameters are possible. A set corresponding to one of the eight axes is identified by a single digit in the range 1-40. (Examples: Axis 01 A-0-nnnn, Axis 03 A-0-nnnn, etc.)

Class 'D' parameters, the DDS drive parameters, are accessed through the SERCOS Service Channel. The drive parameter sets 1 through 8 directly correspond to the SERCOS drive address 1 through 8 and the CLC Axes 1-8. (Examples: Drive 01 S-0-nnnn, Drive 03 P-0-nnnn)

Most servo and position loop parameters are contained on the SERCOS-compatible digital drive. These parameters include: Feed Constant, Kv Factor, In-Position Window, Monitor Window, and all homing parameters. Acceleration, Deceleration, and Jerk are defined in the user program and limits for these are set in the drive. A list of most often changed drive parameters is included at the end of this section. For a complete description of the parameters, refer to the Drive Manual.

Some parameters may be accessed by the user program through the Visual Motion parameter transfer commands or the text programming language Parameter commands. Visual Motion also allows access to specific groups of parameters for editing and archiving. See the relevant chapters on Icon Programming or Text Language Programming, and the section on Direct ASCII Communication for information on transferring parameters.

Lists of all parameters and all non-volatile parameters are available through the CLC’s serial communication port. Transferring parameters using direct communication follows the same format, with various subclasses for names, units, limits, and data. See the chapter on Direct ASCII Communication for more information on the protocol and syntax for direct communication.
3.2. Parameter Transfer Commands

A parameter transfer capability is built into the CLC's programming language. Parameter transfers allow the monitoring of system values and SERCOS command execution. The System, Axis, and Task parameter descriptions in this section specify the type of data (Float, Integer, etc.) set or returned by the PARAMETER/BIT, PARAMETER/GET, PARAMETER/INIT, PARAMETER/SET text language commands. See Chapter 7 - Text Language Programming, Chapter 6 - Icon Programming and the Parameter Transfer Icon.

All CLC and drive parameters of the floating point type may be read into a CLC floating point variable. All other parameters, unless they are a String or List type, can be read into a CLC integer variable. The CLC issues a runtime error if values are transferred to/from the wrong type of variable. The parameters classified as "Read/write at any time" may be changed by a user program.

To transfer SERCOS parameters not listed in this section, refer to the DDS and SERCOS manuals for a specific parameter's type of data. In general: feedback, tuning, and measurement values are floating point; procedure commands are integer. Use the List Parameter capability to obtain a list containing specific parameters, then find the types of CLC variables that can contain specific parameters.

The full capability of the SERCOS drive is accessible through a user program. Procedural commands such as homing can be performed from a user program by initiating the SERCOS homing sequence, then testing the appropriate parameter to determine when the operation has completed. The SERCOS probing functions may be used to implement registration to a mark, or for accurate position measurements. In addition, some of the limits and offsets can be set up at program initialization, or a user program may be written to dynamically tune a drive based on feedback of the system's performance.
3.3. List of Parameters

This chapter includes specific information on the CLC System, Drive, Axis, and Task parameters listed below. Additional information on SERCOS and DDS parameters may be found in the respective SERCOS and DDS manuals.

System (Card) Parameters - Class C, Set 0
CLC system parameters include System Setup, Jogging and Display, Program Management, System Status, Electronic Line Shaft, VME, Pendant, and Parameter Lists.

System Setup Parameters
C-0-0001 Language Selection
C-0-0002 Unit Number
C-0-0003 Host Serial Port Setup
C-0-0004 Pendant Serial Port Setup
C-0-0005 I/O Device Selection
C-0-0006 I/O Device Setup
C-0-0007 I/O Device Direction
C-0-0008 I/O Device Base Address
C-0-0009 Error Reaction Mode
C-0-0010 System Options
C-0-0012 Serial Port B Device Type
C-0-0013 Serial Port A Mode
C-0-0014 Serial Port B Mode
C-0-0016 Communication Time-out Period
C-0-0020 Transmitter Fiber Optic Length

Jogging and Display Parameters
C-0-0042 World Large Increment
C-0-0043 World Small Increment
C-0-0045 World Fast Jog Speed
C-0-0046 World Slow Jog Speed
C-0-0052 Axis Large Increment
C-0-0053 Axis Small Increment
C-0-0055 Axis Fast Jog Velocity
C-0-0056 Axis Slow Jog Velocity

Program Management Parameters
C-0-0090 Download Block Size
C-0-0091 Total Program Memory
C-0-0092 Available Program Memory
C-0-0093 Contiguous Program Memory
C-0-0094 Maximum Executable Program Size
C-0-0098  Initialization Delay
C-0-0099  Minimum SERCOS cycle Time

**System Status Parameters**
- C-0-0100  CLC Firmware Version
- C-0-0101  CLC Hardware Version
- C-0-0102  CLC Version Date
- C-0-0103  Maximum Number of Axes Allowed
- C-0-0111  Data Backup Selection
- C-0-0112  Data Backup Command
- C-0-0120  Operating Mode
- C-0-0121  SERCOS Communication Phase
- C-0-0122  Diagnostic Message
- C-0-0123  Diagnostic Code
- C-0-0124  Extended Diagnostic
- C-0-0125  System Timer Value
- C-0-0126  Date and Time

**Electronic Line Shaft Parameters** *(See also C-0-1000 to C-0-1568)*
- C-0-0150  Master 1 Type
- C-0-0151  Master 1 Drive Address
- C-0-0152  ELS Master Task Assignment
- C-0-0153  Virtual Master Programmed Velocity
- C-0-0154  Virtual Master Programmed Acceleration
- C-0-0155  Virtual Master Programmed Deceleration
- C-0-0156  Virtual Master E-Stop Deceleration
- C-0-0157  ELS Master Current Position
- C-0-0158  ELS Master Current Velocity
- C-0-0159  Master 1 Zero Velocity Window
- C-0-0160  Virtual Master Maximum Jog Velocity
- C-0-0161  Master 1 Ratio Input
- C-0-0162  Master 1 Ratio Output
- C-0-0164  Master 1 Encoder Type
- C-0-0170  PLS 1 Mask Register

**VME Parameters**
- C-0-0200  Bus Arbitration Mode
- C-0-0201  Bus Release Mode
- C-0-0202  VME / CLC Interface
- C-0-0203  VME Arbiter Status
- C-0-0210  Unit 0 Short Address Page
- C-0-0211  Unit 0 Access
- C-0-0212  Unit 0 Window Address Page
C-0-0215  Unit 1 Short Address Page
C-0-0216  Unit 1 Access
C-0-0217  Unit 1 Window Address Page
C-0-0220  Unit 2 Short Address Page
C-0-0221  Unit 2 Access
C-0-0222  Unit 2 Window Address Page
C-0-0225  Unit 3 Short Address Page
C-0-0226  Unit 3 Access
C-0-0227  Unit 3 Window Address Page
C-0-0230  Unit 4 Short Address Page
C-0-0231  Unit 4 Access
C-0-0232  Unit 4 Window Address Page
C-0-0235  Unit 5 Short Address Page
C-0-0236  Unit 5 Access
C-0-0237  Unit 5 Window Address Page
C-0-0240  Unit 6 Short Address Page
C-0-0241  Unit 6 Access
C-0-0242  Unit 6 Window Address Page
C-0-0245  Unit 7 Short Address Page
C-0-0246  Unit 7 Access
C-0-0247  Unit 7 Window Address Page
C-0-0250  Unit 8 Short Address Page
C-0-0251  Unit 8 Access
C-0-0252  Unit 8 Window Address Page
C-0-0255  Unit 9 Short Address Page
C-0-0256  Unit 9 Access
C-0-0257  Unit 9 Window Address Page
C-0-0260  Unit 10 Short Address Page
C-0-0261  Unit 10 Access
C-0-0262  Unit 10 Window Address Page
C-0-0265  Unit 11 Short Address Page
C-0-0266  Unit 11 Access
C-0-0267  Unit 11 Window Address Page
C-0-0270  Unit 12 Short Address Page
C-0-0271  Unit 12 Access
C-0-0272  Unit 12 Window Address Page
C-0-0275  Unit 13 Short Address Page
C-0-0276  Unit 13 Access
C-0-0277  Unit 13 Window Address Page
C-0-0280  Unit 14 Short Address Page
C-0-0281  Unit 14 Access
C-0-0282  Unit 14 Window Address Page
C-0-0285  Unit 15 Short Address Page
C-0-0288  Unit 15 Access
C-0-0289  Unit 15 Window Address Page
C-0-0286  Unit 15 Access
C-0-0287  Unit 15 Window Address Page

**Pendant**
C-0-0801  Pendant Protection Level 1 Password
C-0-0802  Pendant Protection Level 2 Password
C-0-0803  Pendant User Accessible Floats Section
C-0-0804  Pendant User Accessible Integers Section
C-0-0805  Pendant Start of User Accessible Registers
C-0-0806  Pendant End of User Accessible Registers
C-0-0807  Pendant Password Timeout
C-0-0810  Task ID

**Electronic Line Shaft Parameters** *(See also C-0-0150 to C-0-0164)*
C-0-1000  ELS Primary Master
C-0-1001  ELS Secondary Master
C-0-1010  Master Switching Threshold
C-0-1011  Diff. Between Real Master Positions
C-0-1015  Virtual Master Current Position
C-0-1016  Virtual Master Current Velocity
C-0-1508  Master 1 Filter Cutoff Frequency
C-0-1509  Master 1 Filter Feed Forward Constant
C-0-1517  Master 1 Current Position
C-0-1518  Master 1 Current Velocity
C-0-1550  Master 2 Type
C-0-1551  Master 2 Drive Address
C-0-1552  Master 2 Encoder Type
C-0-1554  Master 2 Ratio Input
C-0-1555  Master 2 Ratio Output
C-0-1556  Master 2 Zero Velocity Window
C-0-1567  Master 2 Current Position
C-0-1568  Master 2 Current Velocity

**System Parameter Lists**
C-0-2000  List of All Parameters
C-0-2001  List of Required Parameters
C-0-2010  List of SERCOS Devices
C-0-2011  List of SERCOS Drives
C-0-2012  List of SERCOS I-O Stations
C-0-2013  SERCOS I-O Configuration List
C-0-2020  Diagnostic Log List
C-0-2021  Diagnostic Log Options
C-0-2501  Oscilloscope signal 1 type.
C-0-2502 Oscilloscope signal 2 type.
C-0-2503 Oscilloscope signal 3 type.
C-0-2504 Oscilloscope signal 1 number.
C-0-2505 Oscilloscope signal 2 number.
C-0-2506 Oscilloscope signal 3 number.
C-0-2507 Oscilloscope signal 1 axis.
C-0-2508 Oscilloscope signal 2 axis.
C-0-2509 Oscilloscope signal 3 axis.
C-0-2510 Oscilloscope sampling rate.
C-0-2511 Oscilloscope signal 1 list, 512 samples.
C-0-2512 Oscilloscope signal 2 list, 512 samples.
C-0-2513 Oscilloscope signal 3 list, 512 samples.
C-0-2514 Oscilloscope sample count, 1 to 512.
C-0-2515 Oscilloscope samples after trigger count.
C-0-2516 Oscilloscope trigger signal type.
C-0-2517 Oscilloscope trigger signal number.
C-0-2518 Oscilloscope trigger signal axis.
C-0-2519 Oscilloscope trigger signal value.
C-0-2520 Oscilloscope trigger mode.
C-0-2521 Oscilloscope trigger source.
C-0-2522 Oscilloscope trigger control word.
C-0-2523 Oscilloscope trigger status word.
C-0-3000 I-O Mapper Program
C-0-3001 I-O Mapper Options
C-0-3003 I-O Mapper Total Operations
C-0-3004 I-O Mapper File Size
C-0-3005 I-O Mapper Executable Size
C-0-3100 Cam Tags
C-0-3101 Cam Table 1
C-0-3102 Cam Table 2
C-0-3103 Cam Table 3
C-0-3104 Cam Table 4
C-0-3105 Cam Table 5
C-0-3106 Cam Table 6
C-0-3107 Cam Table 7
C-0-3108 Cam Table 8
Task Parameters - Class T
CLC user task parameters include Task Setup, Task Status, Coordinated Motion and Robotics, Coordinated Status, and Parameter Lists.

Task Setup Parameters
T-0-0001  Task Motion Type
T-0-0002  Task Options

Coordinated Motion
T-0-0005  World Position Units
T-0-0010  Kinematic Number
T-0-0011  Coordinated X-axis
T-0-0012  Coordinated Y-axis
T-0-0013  Coordinated Z-axis
T-0-0020  Maximum Path Speed
T-0-0021  Maximum Path Acceleration
T-0-0022  Maximum Path Deceleration
T-0-0023  Look Ahead Distance
T-0-0024  Velocity Override
T-0-0025  Maximum Jog Increment
T-0-0026  Maximum Jog Velocity

Robotics
T-0-0035  Relative Point Used for Origin
T-0-0036  Relative Point Used for Tool Frame
T-0-0050  Kinematic Value 1
T-0-0051  Kinematic Value 2
T-0-0052  Kinematic Value 3
T-0-0053  Kinematic Value 4
T-0-0054  Kinematic Value 5
T-0-0055  Kinematic Value 6
T-0-0056  Kinematic Value 7
T-0-0057  Kinematic Value 8
T-0-0058  Kinematic Value 9
T-0-0059  Kinematic Value 10

Coordinated Motion Status
T-0-0100  Target Point Number
T-0-0101  Segment Status
T-0-0102  Rate Limit Status
T-0-0111  Current X Position
T-0-0112  Current Y Position
T-0-0113  Current Z Position

Task Status
T-0-0120  Task Operating Mode
T-0-0122  Task Diagnostic Message
T-0-0123  Task Status Message
T-0-0130  Current Instruction Pointer
T-0-0131  Current Instruction
T-0-0132  Instruction Pointer at Error
T-0-0133  Composite Instruction Pointer
T-0-0135  Current Subroutine
T-0-0136  Stack Variable Data
T-0-0137  Subroutine Breakpoint
T-0-0138  Sequencer Information
T-0-0200  Last Active Event Number

Task Parameter Lists
T-0-2000  List of All Parameters
T-0-2001  List of Required Parameters
Axis Parameters - Class A
CLC axis parameters include Setup Parameters, Status Parameters, Electronic Line Shaft, Feedback Capture, Optional SERCOS Data, and Parameter lists.

Axis Setup Parameters
A-0-0001 Task Assignment
A-0-0002 Type of Positioning
A-0-0003 Axis Motion Type
A-0-0004 Axis Options
A-0-0005 Linear Position Units
A-0-0006 Reference Options
A-0-0007 Configuration Mode
A-0-0009 Drive PLS Register
A-0-0010 Start of Drive I/O
A-0-0011 Drive I-O Card 1 Setup
A-0-0014 Drive I-O Card 2 Setup
A-0-0017 Drive I-O Card 3 Setup
A-0-0020 Maximum Velocity
A-0-0021 Maximum Acceleration
A-0-0022 Maximum Deceleration
A-0-0023 Jog Acceleration
A-0-0025 Maximum Jog Increment
A-0-0026 Maximum Jog Velocity
A-0-0030 Ratio Mode Master Axis
A-0-0031 CLC Cam/Ratio Master Factor (N)
A-0-0032 CLC Cam/Ratio Slave Factor (M)
A-0-0033 CLC Cam Stretch Factor (H)
A-0-0034 CLC Cam Currently Active
A-0-0035 CLC Cam Linear Scaling (L)
A-0-0036 Ratio Mode Encoder Type
A-0-0037 Ratio Mode Step Rate

Axis Status Parameters
A-0-0100 Target Position
A-0-0101 Commanded Position
A-0-0102 Feedback Position
A-0-0110 Programmed Velocity
A-0-0111 Commanded Velocity
A-0-0112 Feedback Velocity
A-0-0120 Programmed Acceleration
A-0-0131 SERCOS Control Word
A-0-0132 SERCOS Status Word
A-0-0133 AT Error Count
A-0-0140  Mfg. Class 3 Status Word  
A-0-0141  Torque Mode Commanded Torque  
A-0-0142  Torque Feedback (cyclic)  
A-0-0145  Current Motion Type  

**Axis Parameters - Electronic Line Shaft**  
A-0-0150  Programmed Ratio Adjust  
A-0-0151  Programmed Phase Offset  
A-0-0153  CLC Cam Adjust Average Velocity  
A-0-0155  CLC Cam Adjust Time Constant  
A-0-0157  CLC Cam Current Master Adjust  
A-0-0159  Ratio Adjust Step Rate  
A-0-0160  Commanded Ratio Adjust  
A-0-0161  CLC Cam Programmed Slave Adjust  
A-0-0162  CLC Cam Current Slave Adjust (Sph)  
A-0-0163  CLC Cam Output Position  
A-0-0164  ELS Options  

**Axis Feedback Capture (Registration)**  
A-0-0170  Feedback Capture Status  
A-0-0171  Probe 1 Positive Captured Position  
A-0-0172  Probe 1 Negative Captured Position  
A-0-0173  Probe 2 Positive Captured Position  
A-0-0174  Probe 2 Negative Captured Position  

**Optional SERCOS Data**  
A-0-0180  Optional Command ID #1  
A-0-0181  Optional Command ID #2  
A-0-0182  Optional Command ID #3  
A-0-0185  Optional Feedback ID #1  
A-0-0186  Optional Feedback ID #2  
A-0-0190  Command Data #1  
A-0-0191  Command Data #2  
A-0-0192  Command Data #3  
A-0-0195  Feedback Data #1  
A-0-0196  Feedback Data #2  

**Axis Parameter Lists**  
A-0-2000  List of All Parameters  
A-0-2001  List of Required Parameters
3.4. System Parameters

Class C, Set 1

The system parameter set includes setup parameters for system configuration, program options, serial interface, and I/O options. Also included are status parameters such as operating mode, and diagnostic messages.

3.4.1. System Setup

C-0-0001 Language Selection

<table>
<thead>
<tr>
<th>Selections:</th>
<th>0 = German (future)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 = English</td>
</tr>
<tr>
<td></td>
<td>2 = French</td>
</tr>
<tr>
<td></td>
<td>3 = Spanish (Future)</td>
</tr>
</tbody>
</table>

Default: 1

Attributes: Integer, Read/write at any time

Parameter C-0-0001 selects the language in which CLC diagnostic and parameter messages are sent. The Language parameter for each Indramat DDS drive is also set automatically.

C-0-0002 Unit Number

<table>
<thead>
<tr>
<th>Selections:</th>
<th>0-15</th>
</tr>
</thead>
</table>

Attributes: Integer, Read-only (set by switch on board)

The CLC permits cards to communicate over the VME backplane. This parameter will set the network address of each card so that one Host can communicate with up to 15 cards. When changing the unit number in this parameter be sure to also change the card number in the card selection setup menu. The CLC card will then need to be reset by power cycling.

Note: This applies to the CLC-V only. For CLC-D, it is software selectable since multiple cards can be addressed with an RS-485 connection.
C-0-0003 Host Serial Port Setup

Format: Baud Rate + Option
Valid Baud Rates: 300, 1200, 2400, 4800, 9600, 19200, 38400
Valid Options: 1 = checksum on
2 = Send CLC Status
Default: 9601 (9600 baud, checksum on)
Attributes: Integer, Read/write at any time

This parameter sets the baud rate and options for Serial Port A (X27 on CLC/D and CLC/P), which communicates with a PC, a terminal, or any device that follows the CLC ASCII Host Protocol. The port always operates with 8 data bits, 1 stop bit, and no parity.

If the checksum option is enabled, the CLC will send a checksum as described in the Host Protocol Description, and will check the checksum in data received from the host.

If the Send CLC Status option is enabled, the CLC will send the C-0-0122 Diagnostic Message whenever the status changes. The message is sent in the format ":1 Message\r\n".

The options are added together with the baud rate. For example, to set 9600 baud, checksum on, the parameter would be set to (9600 + 1) = 9601.

C-0-0004 Pendant Serial Port Setup

Format: Baud Rate + Options
Valid Baud Rates: 1200, 2400, 4800, 9600
Options for Host Protocol: 1 = checksum on
2 = send CLC status
Default: 9601 (9600 + checksum on)
Attributes: Integer, Read/write at any time

Versions previous to GPS-01.20: This parameter sets the baud rate and options for the Pendant Serial Port, which always communicates with the teach pendant. The pendant is shipped with a default of 9600 baud, so this parameter should not need to be changed.

Versions starting with GPS-01.20: This parameter sets the baud rate and options for Serial Port B (X28 on CLC/D and CLC/P). The device connected to this port is selected in parameter C-0-0012 Serial Port B Device Type. The port always operates with 8 data bits, 1 stop bit, and no parity. See the description of parameter C-0-0003 Host Serial Port Setup for Host Protocol options.

Note that the baud rate of this port is limited to 9600. For high-speed host communications, port A should be used.
C-0-0005 I/O Device Selection

Selections:  
0 = no parallel I/O  
1 = Xycom XVME-202 PAMUX  
2 = Pentland MPV922 VME  
3 = Xycom XVME-201 Digital  
4 = Indramat Demo  
5 = Xycom XVME-244 Digital

Default: 0  
Attributes: Integer, Read/write at any time

The CLC can communicate with one parallel I/O device in addition to I/O on the DDS digital drives. Each device is mapped into a set of I/O registers. *See Chapter 2. CLC I/O Systems* for more information.

The Indramat Demo selects a XVME-201 I/O card and inverts the signal states. This selection is not recommended for field use because the I/O lines are set to a high state on reset.

C-0-0006 I/O Device Setup

Selections: LS bit selects lowest address  
Default: 0000000000000000  
Attributes: Binary Word, Read/write at any time

Sixteen binary bits select the address or other setup information for the device. The format of this parameter depends on the device selected.

For the Xycom PAMUX, the 16 bits correspond to stations 0x0000-0x000F.

For Xycom Digital, they correspond to the two I/O connectors on up to 8 cards.

*See Chapter 2. CLC I/O Systems* for more information.

C-0-0007 I/O Device Direction

Selections: LS bit selects direction for lowest address  
Default: 0000000000000000  
Attributes: Binary Word, Read/write at any time

The amount of I/O can be configured for each device. For Xycom I/O devices, a "0" selects a 24 bit station to have 16 inputs and 8 outputs, a "1" selects 16 outputs and 8 inputs.

*See Chapter 2. CLC I/O Systems* for more information.
C-0-0008 I/O Device VME Start Address

Selections: VME Short Address for selected I/O device
Default: 0x0000
Attributes: Hex Word, Read/write at any time

This parameter selects the VME bus base address for the specified parallel I/O card. VME parallel I/O is accessed using the 16-bit short address space. This parameter sets the starting address for an I/O interface and must correspond to the address settings of the interface card. Refer to the appropriate CLC I/O Systems and VME memory sections for more information.

C-0-0009 Error Reaction Mode

Codes: 0 = immediately disable drives
        1 = decelerate to zero velocity, then disable drives
Default: 0
Attributes: Integer, write only in parameter mode

This parameter selects the CLC’s reaction to a fatal shutdown error on the card or the drive. Fatal errors include Emergency Stop, Zone Violation, or Drive Class 1 Shutdowns.

When this parameter is set to 0 (default), the drives are immediately disabled. Velocity is immediately set to zero and the brakes are engaged.

When this parameter is set to 1, all motion will come to a controlled stop before the drives are disabled. In the case of coordinated and single-axis motion, the maximum task or axis deceleration will be used. For an ELS virtual master, the E-Stop deceleration will be used.

This Parameter works with drive error reaction mode parameters: P-0-0117, P-0-0118, P-0-0119.

C-0-0010 System Options

Units: none
Default: 0000000000000000
Attributes: Binary, Write in parameter mode only
Note: Bit 1 is the rightmost bit

This parameter sets several options for the CLC system and for the SERCOS ring.

**Bit 1:** Simulation Mode

0 = normal drive operation
1 = simulation mode, do not scan for drives
When this bit is set to 1, the axes will be simulated. The SERCOS ring will not be scanned for drives, and the drive enable bits will be ignored. This mode is useful for simulating coordinated motion when the CLC is not connected to the actual system, or when a program does not contain any axes. All axis and task status parameters are simulated. Drive parameters and I-O since they require a SERCOS drive, are not simulated. Any drive-controlled motion (homing, single-axis, etc.) is also not simulated.

**Bits 2-3: Phase 2 SERCOS I-O Update**

<table>
<thead>
<tr>
<th>bit3</th>
<th>bit2</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>= 500ms update</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>= 10 second update</td>
</tr>
<tr>
<td>1</td>
<td>x</td>
<td>= no update</td>
</tr>
</tbody>
</table>

In SERCOS Phase 2 (Parameter Mode), all drive-resident I-O cards are scanned every 500ms by default (bit=0). This can slow down communications when downloading parameter lists from the drives, depending upon how many I-O cards there are. If it is not critical to have fast updates of I-O in Phase 2, set these bits to a nonzero value. The I-O will be scanned every 10 seconds or not at all, which speeds up the communication to the user interface.

**Bit 4: SERCOS synchronization clock**

*On CLC-V 2.3 hardware only*

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SERCOS clock is independent</td>
</tr>
<tr>
<td>1</td>
<td>Use clock generated by Card 0</td>
</tr>
</tbody>
</table>

When this bit is set to (0), each CLC card has an independent SERCOS ring. The clocks between these rings are not synchronized.

When this bit is set to (1), synchronization of multiple SERCOS rings across the VME backplane is accomplished by using a common SERCOS clock driven by one card. The card with Address 0 always drives the clock. This option should be enabled for applications that require synchronization between all axes in the system. All cards that use the common clock must have this bit set to (1).

**Notes:**

1. If card 0 does not exist or has a faulty interface board, the SERCOS rings for all other cards on the backplane will stay in phase 0 and the CLC will issue a SERCOS Controller Error.
2. All cards on the bus that use a common synchronization clock must have the same SERCOS cycle time set in Parameter C-0-0099 CLC Hardware Cycle Time or Minimum SERCOS Cycle Time. Automatic configuration of cycle time is not performed by the CLC, but a diagnostic indicates the cycle time that should be set in this parameter.

3. The LED that blinks on the CLC front panel is not synchronized with the SERCOS ring and is used for diagnostic purposes only. The LED’s on the CLC’s will not blink in unison.

Bits 5-11: reserved for future use

Bit 12: Prioritized Service Channel

*On CLC software versions 5.0 and greater*

- 0 = Don’t prioritize the service channel (default)
- 1 = User tasks have priority over lists from user interface

Only one task or user interface can use a drive’s SERCOS service channel at a time. When drive parameter lists, long text strings, or oscilloscope data are transferred from one task, the other task could be suspended from 100ms to 5 seconds.

If the timing for user task service channel access is critical, this bit should be set to (1). The user tasks will suspend any SERCOS transmission of any text strings or lists from the user interface, and the communication error "!78 Service channel in use" will be issued.

If user task service channel access is not critical, parameter lists and oscilloscope are seldom used during normal operation, or nuisance "78" errors occur while viewing parameters, this bit should be set to (0).

**Note:** Even with prioritization, service channel access can vary between 10 and 100 ms. Therefore, any time-critical parameter transfers should be from the cyclic data if possible; or should be put into a non-critical section of the user program.

Bit 13: Ignore Drive Warnings

*On CLC software versions 3.0 and greater*

- 0 = Drive warnings cause the CLC to issue a shutdown error
- 1 = The CLC ignores drive warnings

Default: 0

By default, the CLC issues the error 498 Drive D Shutdown Error when the drives issue a Class 2 Diagnostic Warning. Sometimes it is necessary to disable this error to allow warning checks to be done in the user program before the system is shut down, or to prevent nuisance faults from being issued during testing. This is a global option for all axes in the system.
Bit 14: Ignore Axis Ready Status in Program Commands

On CLC software versions 01.30 and greater

0 = Error is issued if axis is not ready
1 = No error is issued if axis is not ready

In some applications, it is necessary to add and remove drives from the system by setting the disable bit in the axis control register. If this bit is (0) (default), the CLC issues an error if the drive is ready before any commands that start motion. These commands include the start command, the homing command, and the operation mode switch. If this bit is (1), the CLC does not issue an error if the drive is not ready.

Bit 15: Use Parameter C-0-00099 for SERCOS Cycle Time

On CLC software versions 01.20 to 01.30 only

0 = automatically calculate SERCOS Cycle Time
1 = set SERCOS cycle time using parameter C-0-0099

See the description of C-0-0099 CLC Hardware Cycle Time or Minimum SERCOS Cycle Time.

Bit 16: Disable AT Timing Check

On CLC software versions 02.10 and greater

0 = Check SERCOS AT timing (RECOMMENDED)
1 = Do not check AT timing
Default: 0

An option to disable drive telegram time checking is necessary for older versions of DDS 2.1 SERCOS interface cards (DSS and DSI cards). When Bit 16 in System parameter C-0-0010 is (0) (default, recommended), the AT time check is enabled. When it is (1), the AT time check is disabled. If SERCOS disconnect errors occur with this firmware version and not with previous versions, the AT time check should be disabled or the DSS card should be upgraded.
C-0-0012 Serial Port B Device Type

Selections:  
0 = off
1 = ASCII Host Protocol
2 = TPT Teach Pendant

Default: 2 (teach pendant)

Attributes: Integer, Read/write in parameter mode
Must Reset CLC for option to take effect

This parameter selects the type of device that is connected to serial port B. This selection takes effect at the next reset (from power-up or reset switch) of the CLC. The baud rate and options of this port are configured in parameter C-0-0004 Pendant Serial Port Setup.

C-0-0013 Serial Port A Mode

Selections:  
232 = RS-232
422 = RS-422
485 = RS-485

Default: 232

Attributes: Integer, Read/write at any time (on CLC-D or P)
Read-only on CLC-V (always 232)

This parameter selects the serial port A communication mode. This selection takes effect immediately. The baud rate and options of this port are configured in parameter C-0-0003 Host Serial Port Setup. On the CLC-V, only RS-232 is allowed.

RS-232
Connect to one device at a time according to RS-232 standard. Maximum cable length is 20 meters.

RS-422
Connect to one device at a time according to RS-422 standard. Maximum cable length is 200 meters.

RS-485
The CLC is a slave on a multi-drop ring with a host and up to 16 slaves, using the RS-485 standard. Maximum cable length is 200 meters.
C-0-0014 Serial Port B Mode

<table>
<thead>
<tr>
<th>Selections</th>
<th>232 = RS-232</th>
</tr>
</thead>
<tbody>
<tr>
<td>422 = RS-422</td>
<td></td>
</tr>
<tr>
<td>485 = RS-485</td>
<td></td>
</tr>
</tbody>
</table>

Default: 232

Attributes: Integer, Read/write at any time

(on CLC-D or P) Read-only on CLC-V (always 232)

This parameter selects the serial port B communication mode. This selection takes effect immediately. The baud rate and options of this port are configured in parameter C-0-0004 Pendant Serial Port Setup. Connection modes include RS-232, RS-422, and RS-485.

C-0-0016 Communication Time-out Period

<table>
<thead>
<tr>
<th>Units</th>
<th>millisec</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>2000</td>
</tr>
</tbody>
</table>

Attributes: Integer, Read/write at any time

This parameter adjusts the communication time-out period. The state of the communication error timer is set to enabled/disabled by start/stop commands from the serial device. A CLC Shutdown Error occurs if the communication error timer changes states from the enabled to the timed-out state. Only a Fault Clear can move the communication timer from the timed-out to the disabled state. If the communication timer is enabled, the timer is reset after each valid Reset Timer Command. The communication timer is reset by both a Timer Reset command and a change of state from disabled to enabled (i.e., Start Timer command).

C-0-0020 Transmitter Fiber Optic Length

<table>
<thead>
<tr>
<th>Units</th>
<th>meters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>0 = minimum transmit power</td>
</tr>
<tr>
<td>Attributes</td>
<td>Integer, Read/write at any time</td>
</tr>
</tbody>
</table>

(takes effect next shift down in phase)

This parameter adjusts the intensity of the output from the CLC’s SERCOS transmitter, based on the length of the cable in meters. The length values are for plastic fiber. For glass fiber, a smaller length must be entered for the equivalent power. In future releases, another parameter will be added to select glass or plastic.
3.4.2. Jogging and Display

C-0-0042 World Large Increment

Units: Percent
Limits: 1-99
Default: 50
Attributes: Integer, Read/write at any time

This parameter sets the incremental coordinated jogging distance that is used when Large Increment is selected. It is based on the value entered in task parameter T-0-0025 Maximum Jog Increment.

C-0-0043 World Small Increment

Units: Percent
Limits: 1-99
Default: 50
Attributes: Integer, Read/write at any time

This parameter sets the incremental coordinated jogging distance used when Small Increment is selected. It is based on the value entered in task parameter T-0-0025 Maximum Jog Increment.

C-0-0045 World Fast Jog Speed

Units: Percent
Limits: 1-99
Default: 50
Attributes: Integer, Read/write at any time

This parameter sets the coordinated jogging speed used when Fast Jog is selected. It is based on the value entered in task parameter T-0-0026 Maximum Jog Velocity.

C-0-0046 World Slow Jog Speed

Units: Percent
Limits: 1-99
Default: 50
Attributes: Integer, Read/write at any time

This parameter sets the coordinated jogging speed used when Slow Jog is selected. It is based on the value entered in task parameter T-0-0026 Maximum Jog Velocity.
C-0-0052 Axis Large Increment

<table>
<thead>
<tr>
<th>Units</th>
<th>Percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Limits</td>
<td>1-99</td>
</tr>
<tr>
<td>Default</td>
<td>50</td>
</tr>
<tr>
<td>Attributes</td>
<td>Integer, Read/write at any time</td>
</tr>
</tbody>
</table>

This parameter sets the incremental jogging distance used when Large Increment is selected. It is based on the value entered in axis parameter A-0-0025 Maximum Jog Increment.

C-0-0053 Axis Small Increment

<table>
<thead>
<tr>
<th>Units</th>
<th>Percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Limits</td>
<td>1-99</td>
</tr>
<tr>
<td>Default</td>
<td>50</td>
</tr>
<tr>
<td>Attributes</td>
<td>Integer, Read/write at any time</td>
</tr>
</tbody>
</table>

This parameter sets the incremental jogging distance used when Small Increment is selected. It is based on the value entered in axis parameter A-0-0025 Maximum Jog Increment.

C-0-0055 Axis Fast Jog Velocity

<table>
<thead>
<tr>
<th>Units</th>
<th>Percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Limits</td>
<td>1-99</td>
</tr>
<tr>
<td>Default</td>
<td>50</td>
</tr>
<tr>
<td>Attributes</td>
<td>Integer, Read/write at any time</td>
</tr>
</tbody>
</table>

This parameter sets the axis jogging speed used when Fast Jog is selected. It is based on the value entered in axis parameter A-0-0026 Maximum Jog Velocity.

C-0-0056 Axis Slow Jog Velocity

<table>
<thead>
<tr>
<th>Units</th>
<th>Percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Limits</td>
<td>1-99</td>
</tr>
<tr>
<td>Default</td>
<td>50</td>
</tr>
<tr>
<td>Attributes</td>
<td>Integer, Read/write at any time</td>
</tr>
</tbody>
</table>

This parameter sets the axis jogging speed used when Slow Jog is selected. It is based on the value entered in axis parameter A-0-0026 Maximum Jog Velocity.
3.4.3. Program Management

C-0-0090 Download Block Size

<table>
<thead>
<tr>
<th>Data:</th>
<th>1-200</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>16</td>
</tr>
<tr>
<td>Attributes:</td>
<td>Integer, Read/write at any time</td>
</tr>
</tbody>
</table>

This parameter selects the block size that will be used for user program downloads to the CLC. Refer to the section on Direct ASCII Communication for more information.

C-0-0091 Total Program Memory

<table>
<thead>
<tr>
<th>Units:</th>
<th>Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attributes:</td>
<td>Integer, Read-only</td>
</tr>
</tbody>
</table>

This is the total file memory on the CLC card that can be used for programs and text messages. CLC point, variable, I/O, and event tables are included in this number.

C-0-0092 Available Program Memory

<table>
<thead>
<tr>
<th>Units:</th>
<th>Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attributes:</td>
<td>Integer, Read-only</td>
</tr>
</tbody>
</table>

This is the amount of memory the CLC currently has available for storage of programs and text messages.

C-0-0093 Contiguous Program Memory

<table>
<thead>
<tr>
<th>Units:</th>
<th>Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attributes:</td>
<td>Integer, Read-only</td>
</tr>
</tbody>
</table>

The CLC dynamically allocates the non-volatile memory used for user programs and data. This is the largest unit of storage (i.e. the largest program or cam) that can be stored on the card. If this number is much smaller than C-0-0092 Available Program Memory, Available Program Memory, the memory is fragmented. If the message “Insufficient Program Space” is sent from the CLC, upload programs, delete them, then download them again to de-fragment the memory.
C-0-0094 Maximum Executable Program Size

Units: Bytes
Attributes: Integer, Read-only

The CLC reserves a fixed amount of memory to store the currently active executable program. The executable program contains the instructions but not the points, events, variables, and labels. This parameter indicates the largest executable program that can be stored on the CLC. The program size can be checked against this number before the program is downloaded. If the CLC receives a user program that exceeds this number, communication error “!60 Executable program is too large” is issued.

C-0-0098 Initialization Delay

Units: seconds
Default: 0 (no delay)
Limits: 0-30
Attributes: Integer, Non-volatile, Read/write in parameter mode

This parameter causes the CLC to delay for the specified number of seconds before it initializes the SERCOS ring. This prevents the CLC from issuing a ‘No drives were found on ring’ error if I-O stations or drives take a long time to initialize the SERCOS ring.

C-0-0099 CLC Hardware Cycle Time or Minimum SERCOS Cycle Time

Units: microseconds
Default: 2000 for CLC-GPS
        4000 for CLC-SAR
Attributes: Integer, Read/write in parameter mode

For some applications, such as coordinated motion using multiple tasks or cam motion using multiple cams, the application program can be made faster by increasing the SERCOS cycle time.

This parameter’s operation depends on the hardware and software versions.

CLC-V 2.1 (with DFI/DAS2 SERCOS interface)
all software versions
This parameter sets the cycle time of the SERCOS ring. It must be set to match the clock frequency of the DFI board (2MHz = 2000 microseconds, 4MHz = 4000 microseconds). For current CLC applications with more than six axes the 4000 microsecond hardware is required. This parameter takes effect only after a reset.
CLC-D/P and CLC-V 2.3
with software versions before GPS-02.00
CLC parameter C-0-0010 System Options has an option to automatically or manually set the SERCOS cycle time. When manual cycle time selection is enabled, parameter C-0-0099 sets the cycle time. When automatic cycle time selection is enabled, parameter C-0-0099 sets the minimum SERCOS cycle time. The CLC issues an extended diagnostic message when the selected cycle time is too small for the number of drives or the amount of data on the ring. This parameter takes effect only after a reset.

CLC-D/P and CLC-V 2.3
with software versions GPS-02.00 and greater
and separate SERCOS clocks
The CLC automatically sets the SERCOS cycle time based on the number of drives and the amount of data on the ring. Parameter C-0-0099 sets the minimum SERCOS cycle time. This parameter takes effect when the SERCOS phase is decreased (when switching into parameter mode or activating a program).

CLC-V 2.3
with software versions GPS-02.00 and greater
and common SERCOS clock
When a common SERCOS clock through the VME backplane is enabled with CLC C-0-0010 System Options, the cycle time is not automatically configured by the CLC. Parameter C-0-0099 must be set equal for all cards using the same SERCOS clock. The CLC issues an extended diagnostic message when the selected cycle time is too small for the number of drives or the amount of data on the ring. This parameter takes effect when the SERCOS phase is decreased (when switching into parameter mode or activating a program).
3.4.4. System Status

System status messages are available through the serial ports and the user program, and provide the current status of the CLC system.

C-0-0100 CLC Firmware Version

Type: String
Attributes: Read-only

The CLC displays the firmware version number for the card, which corresponds to the Indramat standard part number:

| CLC*DP-GPS-03V10 |
| Control Type (GPS = general purpose) |
| Hardware Type |

C-0-0101 CLC Hardware Version

Type: String
Attributes: Read-only

The CLC displays the hardware version type code:

| CLC-D 1.1 250233 A03 |
| board revision |
| board part number |
| Hardware revision number |
| Hardware platform (CLC-D, CLC-P, CLC-D) |

C-0-0102 CLC Version Date

Type: String
Attributes: Read-only

This parameter shows the release date of the CLC firmware:

Mar 29 1996

C-0-0103 Maximum Number of Axes Allowed

Type: Integer
Attributes: Read-only
This indicates the maximum axis number and the maximum number of axes allowed in this software version. It can be used to distinguish between standard and economy GPS versions, which otherwise keep the same version number.

**C-0-0111 Data Backup Selection**

<table>
<thead>
<tr>
<th>CLC-D/P</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>no selection</td>
</tr>
<tr>
<td>1</td>
<td>save communication parameters to EEPROM</td>
</tr>
<tr>
<td>2</td>
<td>restore comm. parameters from EEPROM</td>
</tr>
<tr>
<td>3</td>
<td>erase comm. parameters from EEPROM</td>
</tr>
</tbody>
</table>

**Attributes:** Integer, read/write in parameter mode only

**Default:** 0 at power-up

The communication parameters on the CLC-D and CLC-P can optionally be saved in a 256-byte EEPROM on the card. This allows communication to take place even after the RAM backup battery has failed. The parameters stored in RAM are used unless those stored in EEPROM are retrieved or the battery fails. The jumpers S1 and S2 override both the RAM and EEPROM values to allow service personnel to connect at a known baud rate (9600) and mode (RS-232).

This parameter selects the command that will be performed by parameter C-0-0112 Data Backup Command. On the CLC-D and CLC-P, this command saves and restores communication parameters to and from EEPROM.

When a (1) is entered in this parameter, the values set in the communication parameters are stored into EEPROM when command C-0-0112 is executed.

When a (2) is entered, the values stored in EEPROM are written to the communication parameters when command C-0-0112 is executed. The restored parameters do not take effect until the power is cycled on the CLC.

A (3) erases the communication parameters from EEPROM when command C-0-0112 is executed. When the battery is lost, the defaults for each parameter are loaded.

**Parameters Saved**

The following communication parameters can be saved to EEPROM:

- C-0-0002 Unit Number
- C-0-0003 Host Serial Port Setup (includes baud rate)
- C-0-0004 Pendant Serial Port Setup (includes baud rate)
- C-0-0012 Serial Port B Device Type
- C-0-0013 Serial Port A Mode
- C-0-0014 Serial Port B Mode
C-0-0112 Data Backup Command

Attributes: Binary, Read/write in parameter mode  
Default: 0 at power-up  
Note: Bit 1 is the rightmost bit, bit 16 the leftmost.

On the CLC-D and CLC-P, this command parameter saves communication parameters to EEPROM or restores them from EEPROM, according to the selection in parameter C-0-0111 Data Backup Selection.

Bit 1: Set Command  
Bit 2: Enable Command  
These bits follow the SERCOS specification for procedure commands. The CLC executes a backup when it sees a (0-1) edge in bits 1 and 2. The backup option must first be selected in parameter C-0-0111. To start the backup, restore, or erase, the following sequence must be used:

<table>
<thead>
<tr>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Data Backup Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Disable Command</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Enable and Start Command (Save or Restore Operation is started)</td>
</tr>
</tbody>
</table>

Procedure for Saving Communication Defaults:
1. Set the communication parameters to the desired values.  
2. Set parameter C-0-0111 to (1).  
3. Set parameter C-0-0112 to (0000000000000000).  
4. Set parameter C-0-0112 to (0000000000000011).

C-0-0120 Operating Mode

Codes: 0 = Initializing CLC  
1 = Parameter Mode  
2 = Run Mode  
Attributes: Integer, Read-only

Provides a code corresponding to the current CLC operating mode.

C-0-0121 SERCOS Communication Phase

Codes: SERCOS Phase (0-4)  
Attributes: Integer, Read-only
This is the current phase (0-4) of the CLC SERCOS master. Note that the drives may have switched to a lower phase if there was an error.

**C-0-0122 Diagnostic Message**

<table>
<thead>
<tr>
<th>Type:</th>
<th>String</th>
</tr>
</thead>
<tbody>
<tr>
<td>Codes:</td>
<td>000-199: Status message</td>
</tr>
<tr>
<td></td>
<td>200-399: Warning error</td>
</tr>
<tr>
<td></td>
<td>400-599: Shutdown error</td>
</tr>
<tr>
<td>Attributes:</td>
<td>read-only</td>
</tr>
</tbody>
</table>

This is the current system status code and message. *See Chapter 8. CLC Diagnostics for listings of status and error codes.*

Example: 412 No drives were found on ring

```
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
</table>
|   | Message
|   | Error code
```

**C-0-0123 Diagnostic Code**

<table>
<thead>
<tr>
<th>Type:</th>
<th>Integer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Codes:</td>
<td>000-199: Status message</td>
</tr>
<tr>
<td></td>
<td>200-399: Warning error</td>
</tr>
<tr>
<td></td>
<td>400-599: Shutdown error</td>
</tr>
<tr>
<td>Attributes:</td>
<td>read-only</td>
</tr>
</tbody>
</table>

This parameter returns a status code. *See Chapter 8. CLC Diagnostics for listings of status and error codes.*

**C-0-0124 Extended Diagnostic**

Attributes: String, read-only

This is a dynamic system message used to provide additional diagnostic information for a status warning or error message (parameter C-0-0122 Diagnostic Message).

**C-0-0125 System Timer Value**

<table>
<thead>
<tr>
<th>Units:</th>
<th>milliseconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attributes:</td>
<td>Integer, Read/Write at any time</td>
</tr>
</tbody>
</table>

This is a general-purpose timer that continuously counts milliseconds while the CLC is running. It is a global system variable and can be read into an integer variable to provide timing for a section...
of a CLC program, or its incremental value can be used to time a process. It is a 31 bit counter with a maximum count is $2,147,483,648 \times (2^{31})$, after which it rolls over to 0 and continues counting. It can be set to any value by the user program or the user interface.

**C-0-0126 Date and Time**

<table>
<thead>
<tr>
<th>Type:</th>
<th>String</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attributes:</td>
<td>Read/Write any time (write requires SET command)</td>
</tr>
</tbody>
</table>

This is the current time and date, which the CLC uses for Flash identification and the diagnostic log. It also provides a time stamp for parameters when the parameter list is uploaded.

The date is printed Month-Day-Year, and the time is Hour-Minute-Second in 24-hour format (example below is 1:05 PM on November 3).

11-03-1994 13:05:24

The host can set the date at any time. To prevent the date from being changed inadvertently when a parameter file is downloaded, the CLC requires ‘SET’ before the new time and date:

>0 CP 1.126 SET 11-04-94 15:33:00

The CLC/V has a real time clock which keeps time even when the power is off. On the CLC/D and CLC/P, the host device must set the time and date when the CLC is powered on, since there is no real time clock.
3.4.5. Electronic Line Shaft Parameters 1 of 2

For C-0-1000 to C-0-1568 See 3.4.8. Electronic Line Shaft Parameters 2 of 2

C-0-0150 Master 1 Type

Codes: 0 = ELS disabled
1 = Virtual Master
2 = Real master to each slave
3 = Real master through SERCOS
4 = Follow axis feedback
Attributes: Integer, automatically set by ELS/INIT command at program activation

This provides a status of the type of ELS master. See the descriptions of Electronic Line Shafting and the ELS/INIT command.

C-0-0151 Master 1 Drive Address

Selections: 0 if real master is disabled
1-max_axis, real master encoder address
Attributes: Integer, automatically set by ELS/INIT command at program activation

This parameter is set to the address of the drive in which the encoder for a real master resides. See the descriptions of Electronic Line Shafting and the ELS/INIT command.

C-0-0152 ELS Master Task Assignment

Codes: 0 = No task selected
1 = Task A
2 = Task B
3 = Task C
4 = Task D
Attributes: Integer, automatically set by ELS/INIT command at program activation.

This is the task with which the ELS virtual or real master is associated. When this task goes into manual mode, the virtual master is ramped to zero velocity using the E-Stop Deceleration (C 1.156). The task assignment is set with the first task (A, B, C, D in order) that includes an ELS/INIT command with a virtual master.
C-0-0153 Virtual Master Programmed Velocity

Units: RPM
Default: 0.0
Attributes: Floating point, read/write at any time

This is the target velocity for the ELS virtual master. A negative value causes motion in the negative direction. It is set using the AXIS/INITIALIZE command or VELOCITY icon in the user program.

C-0-0154 Virtual Master Programmed Acceleration

Units: radians/sec^2
Default: E-stop deceleration (C-0-0156 Virtual Master E-Stop Deceleration)
Attributes: Floating point, read/write at any time

This is the acceleration rate for the ELS virtual master. The master accelerates to a greater speed using this parameter. It is set using the AXIS/INITIALIZE command or ACCEL icon in the user program.

C-0-0155 Virtual Master Programmed Deceleration

Units: radians/sec^2
Default: E-stop deceleration (C-0-0156 Virtual Master E-Stop Deceleration)
Attributes: Floating point, Read/Write at any time

This is the deceleration rate for the ELS virtual master. The master decelerates to a smaller speed using this parameter. It is set using the ELS/DECEL command or DECEL icon in the user program.

C-0-0156 Virtual Master E-Stop Deceleration

Units: radians/sec^2
Default: 200.0
Attributes: Floating point, non-volatile, read/write at any time

The CLC Virtual Master uses this parameter to decelerate to zero velocity at an Emergency Stop, a Shutdown error, or when switching into manual mode without a cycle stop.
C-0-0157 ELS Master Current Position

<table>
<thead>
<tr>
<th>Units:</th>
<th>degrees (modulo 360)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>0.0 at power-up or reset</td>
</tr>
<tr>
<td>Attributes:</td>
<td>Floating point, Read/Write at any time</td>
</tr>
</tbody>
</table>

This run time status parameter is the current position of the active ELS master (real or virtual) in degrees.

It is possible to write a value to this parameter to reset the virtual master position, but care should be taken that all axes are stopped.

C-0-0158 ELS Master Current Velocity

<table>
<thead>
<tr>
<th>Units:</th>
<th>RPM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attributes:</td>
<td>Floating point, read only</td>
</tr>
</tbody>
</table>

This run time status parameter is the current velocity of the CLC Virtual or Real Master in revolutions per minute.

C-0-0159 Master 1 Zero Velocity Window

<table>
<thead>
<tr>
<th>Units:</th>
<th>RPM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>1.0</td>
</tr>
<tr>
<td>Attributes:</td>
<td>Floating point, read/write at any time</td>
</tr>
</tbody>
</table>

This parameter determines when the real master is stopped for diagnostic purposes or for a mode change. When the master's speed is below this value, it is considered at zero velocity.

C-0-0160 Virtual Master Maximum Jog Velocity

<table>
<thead>
<tr>
<th>Units:</th>
<th>RPM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>100.0</td>
</tr>
<tr>
<td>Attributes:</td>
<td>Floating point, read/write at any time</td>
</tr>
</tbody>
</table>

This parameter defines the maximum velocity used for jogging the ELS virtual master. Both the fast and slow axis percent parameters are multiplied by this value and used as the master's jog velocity.
C-0-0161 Master 1 Ratio Input

Units:None
Limits:1.0 to 1*10^9
Default:1.0
Attributes:Floating point, read/write in parameter mode only

See C-0-0162 Master 1 Ratio Output.

C-0-0162 Master 1 Ratio Output

Units:None
Limits:-1 * 10^9 to 1*10^9
Default:1.0
Attributes:Floating point, read/write in parameter mode only

These parameters define a gear ratio for a real master encoder for ELS or cam axes when Real Master through SERCOS or Follow Axis Feedback options are used. The position difference read from the master encoder is multiplied by (C-0-0162 / C-0-0161 Master 1 Ratio Input). The master position is always rotary with a modulo of 360. A ratio output of "-1" changes the direction of the position readout for the master encoder.

Master Position = (Ratio Output / Ratio Input) * Master Encoder Value

C-0-0164 Master 1 Encoder Type

Attributes:Read/write at any time, integer
Units:Milliseconds
Selections:0 if not used or primary feedback is used
1 if primary feedback is used
2 if secondary feedback is used.

This option selects the type of master used for the ELS ‘Follow Axis Feedback’ mode. The primary encoder or secondary encoder can be used. Options (0) and (1) select the master drive’s primary feedback, which is the value read from drive parameter S-0-0051. Option (2) selects the secondary feedback, read from drive parameter S-0-0053. This parameter is set automatically from the els/init instruction.
C-0-0170 PLS 1 Mask Register

Attributes: Read/write at any time, integer
Limits: 0 = no mask is applied to PLS
        1 - 1024 = register to be used for PLS mask

Individual bits of the Programmable Limit Switch may be disabled by setting the bits in an optional mask register. On CLC GPx version 03, this register is selected in parameter C-0-0170. For example, if C-0-0170 is set to 110, and register 110 is set to 1111111111110000, the PLS will control only the first 4 bits of the PLS register. The bits that are masked can be controlled via the I-O mapper or the user program. If C-0-0170 is set to 0, no mask will be applied to the PLS.

3.4.6. VME Bus

C-0-0200 VME Bus Arbitration Mode

Codes:

<table>
<thead>
<tr>
<th>Bit 1</th>
<th>Bit 0</th>
<th>Arbitration mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Prioritized</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Round-Robin</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Prioritized Round-Robin</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bit 2</th>
<th>FAIR Arbitration</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>disabled</td>
</tr>
<tr>
<td>1</td>
<td>enabled</td>
</tr>
</tbody>
</table>

Attributes: Byte, Write-only in Parameter Mode

The CLC/VME can be configured to act as the VME bus Arbiter by enabling the slot 1 System Controller functions (SW5-8). The Arbiter supports three standard VME algorithms for bus arbitration: prioritized, round-robin, and prioritized round-robin. Parameter 200 is used to select the desired arbitration mode. This parameter is also used to select if the fair arbitration algorithm will be utilized.

Example:

```
xxxxxx101
|    |    |    | Enables Round-Robin arbitration
|    |    |    | Enables FAIR arbitration algorithm
|    |    |    | Most significant bit
```
C-0-0201 Bus Release Mode

Codes:

<table>
<thead>
<tr>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
<th>Release Inhibit Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.5 microsecond</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1 µs</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2 µs</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>4 µs</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>8 µs</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>16 µs</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>32 µs</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>64 µs</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bit 3</th>
<th>Release on Bus Clear</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>disabled</td>
</tr>
<tr>
<td>1</td>
<td>enabled</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bit 4</th>
<th>Release Every Cycle</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>disabled</td>
</tr>
<tr>
<td>1</td>
<td>enabled</td>
</tr>
</tbody>
</table>

Attributes: Byte, Write-only in Parameter Mode

As the VME bus master, the CLC provides three optional methods for releasing the bus:
- ROR - release on request
- RBCLR - release on bus clear
- REC - release every cycle

Release on request is always enabled. The Release Inhibit Time may be set to insure a minimum time that the CLC will hold the bus as master.

Example:

```
xxx10101
```

| x | x | x | 1 | 1 | _ sets the release inhibit time to 16 µs |
| x | x | _ | _ | _ disables release on bus clear |
| x | _ | _ | _ enables release every cycle |
| _ | _ | _ most significant bit |


C-0-0202 VME/CLC Interface

Codes:

<table>
<thead>
<tr>
<th>Bit 1</th>
<th>Bit 0</th>
<th>VME bus access time-out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>64000 µs</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1000 µs</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>64 µs</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>16 µs</td>
</tr>
</tbody>
</table>

Bit 2 | VME read/write error
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>VME bus error not fatal to CLC</td>
</tr>
<tr>
<td>1</td>
<td>VME bus error fatal to CLC</td>
</tr>
</tbody>
</table>

Bit 3 | VME SYSFAIL error
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>VME SYSFAIL ignored by CLC</td>
</tr>
<tr>
<td>1</td>
<td>VME SYSFAIL fatal to CLC</td>
</tr>
</tbody>
</table>

Bit 4 | Read-Modify-Write cycles
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Do not allow external RMW cycles to local RAM</td>
</tr>
<tr>
<td>1</td>
<td>External RMW to local RAM allowed</td>
</tr>
</tbody>
</table>

Attributes: Byte, Write-only in Parameter Mode

This parameter specifies how the CLC interfaces with the VME bus. Read-Modify-Write cycles are not permitted as the default. Enabling this function will increase the CLC’s access time to its local RAM if another card contends for access.

Example:

```
xxx01010
```

- Sets the VME access time to 64 µs
- Read/write error not fatal to CLC
- SYSFAIL error fatal to CLC
- RMW cycles allowed from external cards
- Most significant bit
C-0-0203 VME Arbiter Status

Codes:

<table>
<thead>
<tr>
<th>Bits 1</th>
<th>Bit 0</th>
<th>VME bus request level (SW5-6 &amp; 7)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>BR0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>BR1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>BR2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>BR3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bit 2</th>
<th>VME Slot 1 controller status (SW5-8)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>VME bus Slot 1 functions disabled</td>
</tr>
<tr>
<td>1</td>
<td>VME bus Slot 1 functions enabled</td>
</tr>
</tbody>
</table>

Attributes: Byte, Read-only

This read only parameter permits reading the settings of the VME bus request level and Slot 1 controller circuit board switch (SW5) settings.

Example:

```
xxxxx010
```

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>_ CLC bus requests use priority level BR2</td>
</tr>
<tr>
<td></td>
<td>_ CLC not the VME system controller (Slot 1 disabled)</td>
<td></td>
</tr>
<tr>
<td>_</td>
<td>most significant bit</td>
<td></td>
</tr>
</tbody>
</table>

C-0-0210 Unit 0 Short Address Page
See C-0-0285 Unit 15 Short Address Page for description.

C-0-0211 Unit 0 Access
See C-0-0286 Unit 15 Access for description.

C-0-0212 Unit 0 Window Address Page
See C-0-0287 Unit 15 Window Address Page for description.

C-0-0215 Unit 1 Short Address Page
See C-0-0285 Unit 15 Short Address Page for description.

C-0-0216 Unit 1 Access
See C-0-0286 Unit 15 Access for description.

C-0-0217 Unit 1 Window Address Page
See C-0-0287 Unit 15 Window Address Page for description.
C-0-0220 Unit 2 Short Address Page
See C-0-0285 Unit 15 Short Address Page for description.
C-0-0221 Unit 2 Access
See C-0-0286 Unit 15 Access for description.
C-0-0222 Unit 2 Window Address Page
See C-0-0287 Unit 15 Window Address Page for description.

C-0-0225 Unit 3 Short Address Page
See C-0-0285 Unit 15 Short Address Page for description.
C-0-0226 Unit 3 Access
See C-0-0286 Unit 15 Access for description.
C-0-0227 Unit 3 Window Address Page
See C-0-0287 Unit 15 Window Address Page for description.

C-0-0230 Unit 4 Short Address Page
See C-0-0285 Unit 15 Short Address Page for description.
C-0-0231 Unit 4 Access
See C-0-0286 Unit 15 Access for description.
C-0-0232 Unit 4 Window Address Page
See C-0-0287 Unit 15 Window Address Page for description.

C-0-0235 Unit 5 Short Address Page
See C-0-0285 Unit 15 Short Address Page for description.
C-0-0236 Unit 5 Access
See C-0-0286 Unit 15 Access for description.
C-0-0237 Unit 5 Window Address Page
See C-0-0287 Unit 15 Window Address Page for description.

C-0-0240 Unit 6 Short Address Page
See C-0-0285 Unit 15 Short Address Page for description.
C-0-0241 Unit 6 Access
See C-0-0286 Unit 15 Access for description.
C-0-0242 Unit 6 Window Address Page
See C-0-0287 Unit 15 Window Address Page for description.

C-0-0245 Unit 7 Short Address Page
See C-0-0285 Unit 15 Short Address Page for description.
C-0-0246 Unit 7 Access
See C-0-0286 Unit 15 Access for description.
C-0-0247 Unit 7 Window Address Page
See C-0-0287 Unit 15 Window Address Page for description.
C-0-0250 Unit 8 Short Address Page
See C-0-0285 Unit 15 Short Address Page for description.
C-0-0251 Unit 8 Access
See C-0-0286 Unit 15 Access for description.
C-0-0252 Unit 8 Window Address Page
See C-0-0287 Unit 15 Window Address Page for description.

C-0-0255 Unit 9 Short Address Page
See C-0-0285 Unit 15 Short Address Page for description.
C-0-0256 Unit 9 Access
See C-0-0286 Unit 15 Access for description.
C-0-0257 Unit 9 Window Address Page
See C-0-0287 Unit 15 Window Address Page for description.

C-0-0260 Unit 10 Short Address Page
See C-0-0285 Unit 15 Short Address Page for description.
C-0-0261 Unit 10 Access
See C-0-0286 Unit 15 Access for description.
C-0-0262 Unit 10 Window Address Page
See C-0-0287 Unit 15 Window Address Page for description.

C-0-0265 Unit 11 Short Address Page
See C-0-0285 Unit 15 Short Address Page for description.
C-0-0266 Unit 11 Access
See C-0-0286 Unit 15 Access for description.
C-0-0267 Unit 11 Window Address Page
See C-0-0287 Unit 15 Window Address Page for description.

C-0-0270 Unit 12 Short Address Page
See C-0-0285 Unit 15 Short Address Page for description
C-0-0271 Unit 12 Access
See C-0-0286 Unit 15 Access for description.
C-0-0272 Unit 12 Window Address Page
See C-0-0287 Unit 15 Window Address Page for description.

C-0-0275 Unit 13 Short Address Page
See C-0-0285 Unit 15 Short Address Page for description
C-0-0276 Unit 13 Access
See C-0-0286 Unit 15 Access for description.
C-0-0277 Unit 13 Window Address Page
See C-0-0287 Unit 15 Window Address Page for description.
C-0-0280  Unit 14 Short Address Page
See C-0-0285  Unit 15 Short Address  Page for description

C-0-0281  Unit 14 Access
See C-0-0286  Unit 15 Access for description.

C-0-0282  Unit 14 Window Address Page
See C-0-0287  Unit 15 Window Address Page for description.

C-0-0285  Unit 15 Short Address  Page

<table>
<thead>
<tr>
<th>Type</th>
<th>Hex Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attributes</td>
<td>Read/Write in Parameter mode</td>
</tr>
<tr>
<td>Range</td>
<td>0x0001 to 0x00FF.</td>
</tr>
</tbody>
</table>

These parameters set the short VME address page for the CLC card. All handshaking through the CLC mailbox structures will be accessed from this page. Mailboxes are realized as offsets into the defined short VME area. There are 256 pages possible, each consisting of 256 bytes.

C-0-0286  Unit 15 Access

<table>
<thead>
<tr>
<th>Type</th>
<th>Byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attributes</td>
<td>Read/Write in Parameter mode</td>
</tr>
</tbody>
</table>

These parameters specify the VME bus address, width and data width for access to the CLC local RAM. The CLC can be accessed with either Standard or Extended addressing. All of the data widths are supported.

<table>
<thead>
<tr>
<th>Bits 1-0</th>
<th>Data Width</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>D8</td>
</tr>
<tr>
<td>01</td>
<td>D16</td>
</tr>
<tr>
<td>10</td>
<td>D32</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bit 2</th>
<th>Address Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Standard Address Range</td>
</tr>
<tr>
<td></td>
<td>(A24)</td>
</tr>
<tr>
<td>1</td>
<td>Extended Address Range</td>
</tr>
<tr>
<td></td>
<td>(A32)</td>
</tr>
</tbody>
</table>
C-0-0287  Unit 15 Window Address Page

Type: Hex Number
Attributes: Read/Write in Parameter mode

These parameters are used to specify the address page where the CLC’s local RAM will be located in VME address space. The specific addresses used to access CLC back plane communications and IO registers are calculated using these parameters.

The base address must not be set lower than the size of the CLC memory. The lowest address for a 2 Megabyte CLC card is 0x00200000 (0x0020 ). The lowest address for a 4 Mb card is 0x00400000 (0x0040 ).

3.4.7. Teach Pendant

C-0-0801 Pendant Protection Level 1 Password

Default: 0
Limits: 0-9999
Attributes: Integer, Read/write at any time

This parameter defines a four-digit numeric password that prevents entry into protected menus unless it is entered. If it is 0, the password is disabled.

C-0-0802 Pendant Protection Level 2 Password  (Not Currently Used)

C-0-0803 Pendant User Accessible Floats Section

Default: -1
Selections: -1 = all are accessible
            0 = none are accessible
            n  - where n = number of program integers defines range

Attributes: Integer, Read/write at any time

This parameter defines the maximum allowable range for program floats to be user accessible from the teach pendant. The operator can view all the program floats, but the operator can only access the program floats, up to number set in this parameter. If the operator needs to change a program float greater than the number in this parameter then the operator can either enter a password or set the pendant level protection bits (System Control Register 1, bits 15 & 16).
If this parameter is set to 0, no floats are accessible by the operator. If it is set to (-1), all floats are accessible.

Example: User Accessible Program Float Section = 10
When the operator selects Table Edit Menu/Float Table Menu, the operator can only access the first ten floats. The programmer is responsible for structuring the program floats properly.

C-0-0804 Pendant User Accessible Integers Section

Default: -1
Selections: -1 = all are accessible
0 = none are accessible
n - where n = number of program integers defines range
Attributes: Integer, Read/write at any time

This parameter defines the maximum allowable range for program integers to be user accessible from the teach pendant. The operator can view all the program integers, but the operator can only access the program integers, up to number set in this parameter. If the operator needs to change a program integer greater than the number in this parameter then the operator can either enter a password or set the pendant level protection bits (System Control Register 1, bits 15 & 16).

If this parameter is set to 0, no integers are accessible by the operator. If it is set to (-1), all integers are accessible.

Example: User Accessible Program Integer Section = 10
When the operator selects Table Edit Menu/Integer Table Menu, the operator can only access the first ten integers. The programmer is responsible for structuring the program integers properly.

C-0-0805 Pendant Start of User Accessible Registers

Default: 1
Limits: 1 - maximum I-O register number (512 on CLC-D/P, 2048 on CLC-V)
Attributes: Integer, Read/write at any time

See C-0-0806 Pendant End of User Accessible Registers

C-0-0806 Pendant End of User Accessible Registers

Default: maximum I-O register number (512 on CLC-D/P, 2048 on CLC-V)
Limits: 1 - maximum I-O register number
Attributes: Integer, Read/write at any time
These two parameters (C-0-0805 and C-0-0806) define a range of registers accessible to the operator. The operator can view all the registers, but the operator can only access the registers within this window. If the operator needs to change a bit in a register outside the window of this parameter, then the operator can either enter a password or set the pendant level protection bits (System Control Register 1, bits 15 & 16). When the Register I/O Menu is selected on the teach pendant, the first register to be displayed is the number stored in the Start of User Accessible Registers parameter.

**C-0-0807 Pendant Password Timeout**

- **Units:** seconds
- **Default:** 30 seconds
- **Limits:** 0 (disabled) to 3600 (one hour)
- **Attributes:** Integer, Read/write at any time

This parameter sets a timeout on the teach pendant password. After the password is entered (C-0-0801 Pendant Protection Level 1 Password), the user can enter any screens requiring the password for the time set in this parameter. When a key is pressed, the timer is reset. After the timer expires, the password is again required. If the timeout is set to 0, the password is always required.

**C-0-0810 Task ID Number**

Gets Task Status and allows for the message to be displayed on the first line of the TPT.

- 0 Disabled
- 1 Task A status message to line 1 of TPT
- 2 Task B status message to line 1 of TPT
- 3 Task C status message to line 1 of TPT
- 4 Task D status message to line 1 of TPT
3.4.8. Electronic Line Shaft Parameters 2 of 2

For C-0-0150 to C-0-0164 see 3.4.5. Electronic Line Shaft Parameters 1 of 2

C-0-1000 ELS Primary Master

Units: none
Selections: 0, 1, or 2
Default: 0
Attributes: Integer, Read/write in parameter mode only, set automatically by els/init instruction.

This parameter selects the real master to be used as the primary real master for ELS. If it is set to 0, only a virtual master is used. If it is set to 1, real master 1 is used. If it is set to 2, real master 2 is used.

C-0-1001 ELS Secondary Master

Units: none
Selections: 0, 1, or 2
Default: 0
Attributes: Integer, Read/write in parameter mode only

This parameter selects the real master to be used as the secondary real master for ELS. If it is set to 0, only a virtual master is used. If it is set to 1, real master 1 is used. If it is set to 2, real master 2 is used.

C-0-1010 Master Switching Threshold

Units: degrees
Limits: 0.00001 to 10000000.0
Default: 0.1
Attributes: Floating Point, Read/write at any time

When the ELS master is switched from virtual to real or from real 1 to real 2, the positions must match before the switch takes place. This parameter defines the window around which the two master positions are considered equal. If abs(master 1 - master 2) < C-0-1010, then master is switched. Setting this parameter to a large value switches the masters immediately.
C-0-1011 Difference Between Real Master Positions

Units: degrees
Attributes: Floating Point, Read only

This parameter displays the position difference between the primary real master and the secondary real master. This can be read in the user program for diagnostic purposes.

C-0-1015 Virtual Master Current Position

Units: degrees
Attributes: Floating Point, Read/write at any time

This parameter displays the current virtual master position at all times, even while a real master is active.

It is possible to write a value to this parameter to reset the virtual master position, but care should be taken that all axes are stopped or are not synchronized to the virtual master.

C-0-1016 Virtual Master Current Velocity

Units: RPM
Attributes: Floating Point, Read only

This parameter displays the current velocity of the virtual master.

C-0-1508 Master 1 Filter Cutoff Frequency

Units: Break Freq. Hz.
Attributes: Read/write at any time, float

When a filter is chosen for the ELS real master the cutoff frequency for the filter must be entered. The cutoff frequency is the frequency where the signal is reduced 3db. When set to 0 the filter is disabled. (See PID descriptions - VM Menu Commands and Icon Programming)

See C-0-1509 Master 1 Filter Type
C-0-1509 Master 1 Filter Type

Attributes: Read/write at any time, integer

In some systems, the master could cause disturbances due to contact with a part. These disturbances cause oscillations in the master encoder signal. To minimize the oscillations, the CLC provides an optional filter.

This parameter selects the type of filter for the ELS real master. These selection are the same as are available in the CLC PID function. The filters below are described in in the PID description. (See PID descriptions - VM Menu Commands and Icon Programming)

0 = no filter
1 = 1st order low pass
2 = 2nd order low pass (default if C-0-1508 Master 1 Filter Cutoff Frequency > 0)
3 = 3rd order low pass
4 = 2nd order Butterworth
5 = 3rd order Butterworth
6 = 2nd order low pass with velocity feedforward
7 = 3rd order low pass with velocity and acceleration feedforward

See C-0-1508 Master 1 Filter Cutoff Frequency.

C-0-1517 Master 1 Current Position

Units: degrees
Attributes: Floating Point, Read/write at any time

This parameter displays the current position of the first real master.

It is possible to write a value to this parameter to reset the master position, but care should be taken that all axes are stopped or are not synchronized to this master.

C-0-1518 Master 1 Current Velocity

Units: RPM
Attributes: Floating Point, Read only

This parameter displays the velocity of the first real master.
C-0-1550 Master 2 Type

Codes: 0 = disabled
        2 = Real master to each slave
        3 = Real master through SERCOS
        4 = Follow axis feedback

Attributes: Integer

This provides a status of the type of second real master. See the descriptions of Electronic Line Shafting and the ELS/INIT command.

C-0-1551 Master 2 Drive Address

Selections: 0 - if real master is disabled
            1 - max_axis, real master encoder address

Attributes: Integer, automatically set by ELS/INIT command at program activation

This parameter is set to the address of the drive in which the encoder for a real master resides. See the descriptions of Electronic Line Shafting and the ELS/INIT command.

C-0-1552 Master 2 Encoder Type

Attributes: Read/write in parameter mode only, integer
Units: None
Selections: 0 if not used or primary feedback is used
            1 if primary feedback is used
            2 if secondary feedback is used.

This option selects the type of master used for the ELS ‘Follow Axis Feedback’ mode. The primary encoder or secondary encoder can be used. Options (0) and (1) select the master drive’s primary feedback, which is the value read from drive parameter S-0-0051. Option (2) selects the secondary feedback, read from drive parameter S-0-0053.

C-0-1554 Master 2 Ratio Input

Units: None
Limits: 1.0 to 1*10^9
Default: 1.0
Attributes: Floating point, read/write in parameter mode only

See C-0-1555 Master 2 Ratio Output
C-0-1555 Master 2 Ratio Output

Units: None
Limits: -1 * 10^9 to 1*10^9
Default 1.0
Attributes: Floating point, read/write in parameter mode only

These parameters define a gear ratio for the real master encoder for ELS or cam axes. The position difference read from the master encoder is multiplied by (C-0-1555 / C-0-1554). The master position is always rotary with a modulo of 360. The ratio output can be positive or negative, allowing a direction change independent of the value read from the drive.

Master Position = (Ratio Output / Ratio Input) * Master Encoder Value

C-0-1556 Master 2 Zero Velocity Window

Units: RPM
Default: 1.0
Attributes: Floating point, read/write at any time

This parameter determines when the real master is stopped for the master stopped status bit, the axis/wait command, or a mode change. When the master’s speed is below this value, it is considered at zero velocity.

C-0-1558 Master 2 Filter Cutoff Frequency
See C-0-1508 Master 1 Filter Cutoff Frequency for description.

C-0-1559 Master 2 Filter Type
See C-0-1509 Master 1 Filter Type for description.

C-0-1567 Master 2 Current Position

Units: degrees
Attributes: Floating Point, Read/write at any time

This parameter displays the current position of the second real master. It is possible to write a value to this parameter to reset the master position, but care should be taken that all axes are stopped or are not synchronized to this master.
C-0-1568 Master 2 Current Velocity

Units: RPM
Attributes: Floating Point, Read only

This parameter displays the current velocity of the second real master.

3.4.9. System Parameter Lists

C-0-2000 List of All Parameters

Type: Parameter List
Attributes: Read-only

This is a multiple-step List Parameter that requests a list of all the parameter numbers in this parameter set. The 'D' subclass of the serial port protocol is used to list these parameters.

C-0-2001 List of Required Parameters

Type: Parameter List
Attributes: Read-only

This is a multiple-step List Parameter that requests a list of all the parameters stored in nonvolatile CLC RAM that must be set for proper CLC operation. The 'D' subclass of the serial port protocol is used to list these parameters.

C-0-2010 List of SERCOS Devices

Type: Parameter List
Attributes: Variable-length list, read-only

When the CLC switches into Communication Phase 1 it scans the SERCOS ring for all connected devices, even if they are not selected in the program. This includes drives and I-O stations. The first step (0) of the list is the number of devices found on the ring. Steps (1) to (number of devices) list the addresses of the drives in sequential order.
C-0-2011 List of SERCOS Drives

Type: Parameter List
Attributes: Variable-length list, read-only

This is a list of all SERCOS devices found on the ring that are drives, even if they aren’t selected in the program.

C-0-2012 List of SERCOS I-O Stations

Type: Parameter List
Attributes: Variable-length list, read only

This is a list of all SERCOS devices found on the ring that are I-O stations. The CLC automatically allocates registers for up to five I-O stations with twenty modules per station.

C-0-2013 SERCOS I-O Configuration List

Attributes: Variable-length list, read only

The CLC handles SERCOS I-O from DDS drives and SERCOS I-O stations. DDS drive I-O is configured in axis parameters A-0-0011 Drive I/O Card 1 Setup, A-0-0014 Drive I/O Card 2 Setup, and A-0-0017 Drive I/O Card 3 Setup. SERCOS I-O stations such as Lutze DIOFACE are automatically configured, and no parameters need to be set (see SERCOS I-O and Lutze documentation).

Each step of this list presents the SERCOS I-O configuration in the following format:

Register SERCOS_address Parameter_IDN Direction Number_of_registers

> CD 1.2013.3 100 11 2066 I 1
  | | | | _number of registers (1, 2, 3, or 4)
  | | | _T = input, ‘O’ = output
  | | _SERCOS IDN (corresponds to card or module number)
  | _SERCOS device address
  _Register number

Example:
> CD 1.2013.0 4 $E1 ;4 elements in list
> CD 1.2013.1 40 1 32850 I 1 ;drive 1 DEA 4.1 input at register 40, one register
> CD 1.2013.2 41 1 32849 O 1 ;drive 1 DEA 4.1 output at register 41, one register
> CD 1.2013.3 100 11 2066 I 1 ;I-O station 11 module 0, input register 100, one register
> CD 1.2013.4 102 11 2086 O 1 ;I-O station 11 module 2, output register 102, one register
> CD 1.2013.5 !19 List is finished;end of list
C-0-2020 Diagnostic Log List

Attributes: Variable-length list (from 0-100) of strings, read only

The CLC keeps a log of the last 100 shutdown errors. Each diagnostic log string includes the date and time that the error occurred, the shutdown error code, and any other error codes. The most recent error is listed first. The list is retained in battery-backed RAM.

The extended error code is the data that varies in a message, such as the drive number in “Drive D Shutdown Error”. If there is a task or drive error, it is printed as a secondary error code. Error codes are not supported on all versions of drives.

Emergency Stop and warnings are normally not included in this list. To log these errors, set options in parameter C-0-2021 Diagnostic Log Options.

Format

```
> CD 1.2020.1 11-03 12:15:47 420 1 28 0
| _extended secondary error code
| _secondary error code (28 = excessive deviation)
| _extended error code (1 = drive 1)
| _error code (420 = Drive D shutdown error)
| _time that error occurred
| _date that error occurred
```

Example:
```
> CD 1.2020.0 16 $B0 ;16 errors total
> CD 1.2020.1 11-03 12:09:03 400 0 0 0 ;Last error was E-Stop
> CD 1.2020.2 11-01 08:58:59 422 65 6 0 ;Next to last was Param Transfer Task A, code 06

> CD 1.2020.16 !19 List is finished ;end of list
```

Emergency Stop and warnings are normally not included in this list. To log these errors, set options in parameter C-0-2021 Diagnostic Log Options.
C-0-2021 Diagnostic Log Options

Format: Binary Word
Storage: non-volatile, Read/write in parameter mode
Default: 00000000000000100
Note: Bit 1 is the rightmost bit.

This parameter sets options for the diagnostic log list in parameter C-0-2020 Diagnostic Log List.

**Bit**

1. Include Emergency Stop in the diagnostic log
2. Include Warnings in the diagnostic log
3. Include Drive Undervoltage in diagnostic log

By default, the Emergency Stop error (code 400) is not stored in the diagnostic log, since an E-Stop is usually part of normal operation. To store E-stop errors in the diagnostic log, set bit 1 to (1).

To store warnings (codes 200-399) in the diagnostic log, set bit 2 to (1). To exclude warnings, set bit 2 to (0). To store drive undervoltage (drive error 26) in the diagnostic log, set bit 3 to (1). To exclude drive undervoltage, set bit 3 to (0). This bit defaults to (1).

**C-0-2501 Oscilloscope signal 1 type**

See C-0-2503 Oscilloscope signal 3 type description.

**C-0-2502 Oscilloscope signal 2 type**

See C-0-2503 Oscilloscope signal 3 type description.

**C-0-2503 Oscilloscope signal 3 type**

Where signal type is:

0 = none.
1 = program float.
2 = program integer.
3 = global float.
4 = global integer.
5 = axis parameter.
6 = unsigned register.
7 = signed register.

**C-0-2504 Oscilloscope signal 1 number**

See C-0-2506 Oscilloscope signal 3 number description.
C-0-2505 Oscilloscope signal 2 number
See C-0-2506 Oscilloscope signal 3 number description.

C-0-2506 Oscilloscope signal 3 number

Number, variable number, parameter number, or register number.

C-0-2507 Oscilloscope signal 1 axis
See C-0-2509 Oscilloscope signal 3 axis description

C-0-2508 Oscilloscope signal 2 axis
See C-0-2509 Oscilloscope signal 3 axis description

C-0-2509 Oscilloscope signal 3 axis

Axis number if signal type is “axis parameter”(1-40) or bit number if signal type is “unsigned register”(1-16).

C-0-2510 Oscilloscope sampling rate

Multiples of SERCOS Cycle time, maximum of 64.

C-0-2511 Oscilloscope signal 1 list
See C-0-2513 Oscilloscope signal 3 list for description.

C-0-2512 Oscilloscope signal 2 list
See C-0-2513 Oscilloscope signal 3 list for description.

C-0-2513 Oscilloscope signal 3 list

These are the lists of captured data, range 1-512 samples.

C-0-2514 Oscilloscope sample count

This sets the sample count in C-0-2511, C-0-2512 and C-0-2513 Oscilloscope signal 3 list, range 1 to 512.

C-0-2515 Oscilloscope samples after trigger count

Range 1 to sample count. Count equal to sample count (C-0-2514 Oscilloscope sample count) for no pretrigger.
C-0-2516 Oscilloscope trigger signal type

Same as signal type above (C-0-2501 Oscilloscope signal 1 type).

C-0-2517 Oscilloscope trigger signal number

Variable number, parameter number, or register number.

C-0-2518 Oscilloscope trigger signal axis

axis number if signal type is “axis parameter”, or bit number if signal type is “unsigned register”(1-16).

C-0-2519 Oscilloscope trigger signal value

The value that is compared to C-0-2516 Oscilloscope trigger signal type, C-0-2517 Oscilloscope trigger signal number and C-0-2518 Oscilloscope trigger signal axis for the internal trigger.

C-0-2520 Oscilloscope trigger mode

1 = in positive direction, 2 = in negative direction, 3 = either positive or negative direction.

C-0-2521 Oscilloscope trigger source.

1 = user triggered, 2 = internal trigger.

C-0-2522 Oscilloscope trigger control word.

Bit 0 - Start recording.
Bit 1 - Trigger start.
Bit 2 - Enable (0->1 Allocates memory and configures system , 1->0 frees memory and disables system).

C-0-2523 Oscilloscope trigger status word.

Bit 0 - Trigger activated.
Bit 1 - Data acquisition in progress.
Bit 2 - System Ready( 1 indicates memory allocated and system ready, 0 not.
Bit 3 - Capture complete( 1 indicates the requested sample count is captured
C-0-3000  I-O Mapper

Type: Parameter List
Attributes: Write in parameter mode only

This variable length list is used to map system control and status to the pre-defined I-O devices. At the start of the list, the number of Boolean strings is sent to/from the CLC. This program is handled as a standard parameter list. If a count of zero is sent, the mapping is deleted. See the Chapter 2. CLC I/O Systems for more information.

Example of sending mapping parameter list:
Host tells the CLC that there are 3 Boolean Strings, and CLC responds:
> 1 CP 1.3000.0 3

Host sends the first string:
> 1 CP 1.3000.1 120-1 = !1-5

Host closes list after the last string and CLC acknowledges:
> 1 CP 1.3000.4 0
> 1 CP 1.3000.4 !19 List is finished
C-0-3001  I-O Mapper Options

Default: 0000000000000001
Attributes: Binary Word, Write in parameter mode only

This parameter selects I-O mapper compilation options. Changing these options causes the CLC to recompile the object code.

Bit
1-2  Forcing Options
3-4  reserved
5  Scan Time (!4ms/8ms)

Bits 1-2: Forcing Options
bit2 bit1
0 0  = Disable forcing
0 1  = Force only the reserved control registers (default)
1 1  = Force all registers

I-O forcing allows the results of mapper equations to be ignored if forcing is enabled. Forcing can cause the I-O mapper to execute twice as slow. As options, the CLC can disable forcing, enable forcing for all I-O registers, or enable forcing only for the reserved control registers. Control registers include System control register 1, Axis control register 11, etc. Note: If the teach pendant is used, forcing of the control registers must be enabled.

Bit 5: Scan Time
The I-O mapper on the CLC can be run at either 4 or 8 millisecond scan rates. This allows more strings to be executed with less time taken from user programs if higher-speed I-O isn’t necessary.

C-0-3003  I-O Mapper Total Operations

Attributes: Integer, Read Only

This is the total number of operations used by the I-O mapper currently executing on the CLC. Each operand in a boolean equation or each contact in a ladder diagram corresponds to one operation.

The maximum number of operations allowed on the CLC can be printed using the 'H' (upper limit) subclass of the serial protocol.
C-0-3004  I-O Mapper File Size

Attributes:    Integer, Read Only

This is the space consumed in bytes by the I-O mapper text strings on the CLC. The maximum I-O mapper file size allowed on the CLC can be printed using the 'H' (upper limit) subclass of the serial protocol.

C-0-3005  I-O Mapper Executable Size

Attributes:    Integer, Read Only

This is the space used by the compiled I-O mapper currently running on the CLC.

The maximum executable size of the I-O mapper can be printed using the 'H' (upper limit) subclass of the serial protocol.

C-0-3100 Cam Tags

Type: Parameter List
Attributes: Read-write at any time if cam is not active

This variable length list allows cams to be named with up to 20 characters. The name can then be displayed or changed by the user interface to identify the cam table or the motion for which it is associated. Each step of the list corresponds to a cam number.

Example of sending cam tags list:
Host tells the CLC that there are 16 names, and CLC responds:
>1 CP 1.3100.0 16

Host sends name for Cam 1:
>1 CP 1.3100.1 cam one

Host closes list after the last string and CLC acknowledges:
>1 CP 1.3100.4 17
>1 CP 1.3100.4 !19 List is finished
C-0-3101 Cam Table 1
See C-0-3108 Cam Table 8 for description.

C-0-3102 Cam Table 2
See C-0-3108 Cam Table 8 for description.

C-0-3103 Cam Table 3
See C-0-3108 Cam Table 8 for description.

C-0-3104 Cam Table 4
See C-0-3108 Cam Table 8 for description.

C-0-3105 Cam Table 5
See C-0-3108 Cam Table 8 for description.

C-0-3106 Cam Table 6
See C-0-3108 Cam Table 8 for description.

C-0-3107 Cam Table 7
See C-0-3108 Cam Table 8 for description.

C-0-3108 Cam Table 8

Type: Parameter List
Attributes: Read-write at any time if cam is not active

This variable length list is used to store and retrieve a cam table file. The list starts with step 0 set to the number of entries in the table to be sent to/from the CLC. The rows of the table are then sent as degrees of master, degrees of output, as they would be stored as rows in a space-delineated spreadsheet file.

If a count of zero is sent, the cam is deleted. Otherwise, the count can be from 10 to 1024 table entries. At the end of the download, the CLC expands the table and stores it as 1024 points for accuracy and efficiency. When a table is uploaded from the CLC, the expanded table as stored on the card is sent to the host.

Cam tables may be stored or deleted at any time as long as the cam table number is not active. If the cam is already active, the CLC responds with a communication error “Cam is already active for axis ‘x’”. To download a new table, either switch into parameter mode or deactivate the cam for all axes using it.
Example of sending cam parameter list:
Host tells the CLC that there are 100 table entries:
>1 CD 1.3101.0 100 ;100 entries
>1 CD 1.3101.0 ;CLC acknowledgment

Host sends the first two entries:
>1 CD 1.3101.1 0.0 0.0 ;space delineated x,y of table (x1,y1) = (0.0, 0.0)
>1 CD 1.3101.1 ;CLC acknowledgment
>1 CD 1.3101.2 2.0 3.5 ;(x2,y2) = (2.0, 3.5)
>1 CD 1.3101.2 ;CLC acknowledgment

Host continues for 100 entries, then closes list:
>1 CD 1.3101.101 0.0 0.0 ;n+1 closes list
>1 CD 1.3101.101 !19 List is finished ;CLC acknowledgment
3.5. Task Parameters

Class T, Sets A, B, C, D

Each task has a set of parameters that selects options and displays status information. Sets A through D correspond to CLC tasks A through D. These parameters set up and display information for tasks and coordinated motion.

3.5.1. Task Setup

T-0-0001 Task Motion Type

Codes:  
0 = No coordinated motion  
1 = Normal coordinated motion  
2 = Minimum-time coordinated motion  
3 = Constant velocity coordinated motion  

Attributes: Integer, Read-only

This parameter is set automatically by the user program TASK and PATH commands, and provides the type of task or coordinated motion. Until a valid program is activated, all tasks are non-coordinated by default.

T-0-0002 Task Options

Default: 0000000000000000
Attributes: Binary Word, non-volatile, Read/write in Parameter Mode only

*Note that bit 1 is the rightmost bit.*

Several options can be selected for the CLC User Tasks A-D. These options are selected via bits in this parameter for each task.

Bit 1: Run Task During Errors

Some CLC User Tasks are used for motion, and others for communication or I-O. The motion tasks need to be shut down during drive or axis errors to prevent damage or injury. The communication tasks, however, need to convey diagnostic information to an external device at all times. These tasks cannot be shut down when an error occurs.

If this bit is set to a (0), the task will be shut down by any CLC or drive shutdown error. If it is set to a (1), the task will run during errors unless the error is issued by an instruction in the task. System errors such as E-Stop and drive shutdown errors do not stop the task or cause it to execute the error handler.
The task requires automatic mode to be selected, and cycle start and !stop high for it to run. The I-O mapper can be used to start a task at the exit from parameter mode. The tasks do not run in Parameter Mode.

This option cannot be selected for a task which has axes associated with it.

**Bit 2: Task Errors do not Shut Down other Tasks**
In some tasks that are used for communications, an error should not shut down the motion tasks.

If this bit is set to a (0), errors issued by this task will generate CLC errors, shutting down all tasks affected by errors and causing a controlled stop of all motion.

If it is set to a (1), errors issued by this task will allow all other tasks to run. An error message is displayed in the system diagnostics, but all other tasks remain in operation.

**Bit 3: Parameter Transfer Errors are Warnings**
This option allows the task to continue running if a parameter transfer error occurs. The message “205 Parameter Transfer Warning: see Task A diag.” is issued. If a parameter value is critical to the operation of the task, the task error bit can be tested after the parameter transfer, or this option should be disabled.

**Bit 4: Automatically Start Tasks**
This option automatically starts tasks and keeps them running. All task control bits are ignored. At exit from parameter mode or when an error is cleared, the task is placed into automatic mode and started. The task is stopped when parameter mode is selected. This option can be used for supervisory or communications tasks.

The Override_Auto_Start bit, bit 2 in the Task Control register, can be used to temporarily disable this function. The bits in the control register are enabled as long as bit 2 is high (1), and are ignored when the
3.5.2. Coordinated Motion

T-0-0005 World Position Units

Selections: 0 = inches
1 = millimeters
2 = radians

Default: 0
Attributes: Integer, Write in Parameter Mode only

This parameter selects the display units for coordinated motion position, speed, and acceleration data. No unit conversions are performed when changing this parameter. This parameter automatically sets the Axis Position Units parameter (A-0-0005 Linear Position Units) for all coordinated axes in the specified task.

T-0-0010 Kinematic Number

Attributes: Integer, Read-only

The Kinematic Number represents a library routine which identifies the kinematic to be used for coordinated motion. Kinematic routines are application specific and unique to hardware configurations. Consult your Indramat sales office for kinematics to drive your hardware.

T-0-0011 Coordinated X Axis

Codes: 20 if no axis
1 to maximum axis number

Motion Type: Coordinated
Attributes: Integer, Read-only

This parameter provides the axis number (drive number) corresponding to the X axis of this task (selected in the user program TASK/AXIS command). If the number is zero, no coordinated axes are assigned to this task.
T-0-0012 Coordinated Y Axis

Codes: 0 if no axis
1 to maximum axis number
Motion Type: Coordinated
Attributes: Integer, Read-only

This parameter provides the axis number (drive number) corresponding to the Y axis of this task (selected in the user program TASK/AXIS command). If the number is zero, only one coordinated axis is assigned to this task.

T-0-0013 Coordinated Z Axis

Codes: 0 if no axis
1 to maximum axis number
Motion Type: Coordinated
Attributes: Integer, Read-only

This parameter provides the axis number (drive number) corresponding to the Z axis of this task (selected in the user program TASK/AXIS command). If it is zero, only two coordinated axes are assigned to this task.

T-0-0020 Maximum Path Speed

Units: World speed (units/min)
Default: 1000.0
Motion Type: Coordinated
Attributes: Floating point, Write in Parameter Mode Only

This is the maximum speed allowed for this task's coordinated motion. The speed entries in the CLC’s Absolute and Relative point tables are percentages of this value.

T-0-0021 Maximum Path Acceleration

Units: World Accel (units/sec^2)
Default: 200.0
Motion Type: Coordinated
Attributes: Floating point, Write in Parameter Mode Only

This is the maximum acceleration allowed for this task's coordinated motion. The acceleration entries in the CLC’s Absolute and Relative point tables are percentages of this value.
T-0-0022 Maximum Path Deceleration

Units: World accel (units/sec^2)
Default: 200.0
Motion Type: Coordinated
Attributes: Floating point, Write in Parameter Mode Only

This is the maximum deceleration allowed for this task's coordinated motion. The deceleration entries in the CLC's Absolute and Relative point tables are percentages of this value.

T-0-0023 Look Ahead Distance

Units: World position
Default: 10
Motion Type: Coordinated
Attributes: Floating point, Read/write at any time

This parameter sets the minimum look ahead distance that the CLC's path planner uses to calculate a path. The difference between the current target position and the actual position is called the Look Ahead Distance.

The look-ahead distance should never be set to zero. Generally, the look-ahead distance should be twice the length of the longest blend distance. If all blend distances are zero, the look-ahead should be equal to the shortest geometry segment.

Setting the look-ahead to a large value can improve the overall system performance. The length of the look-ahead determines how many intermediate positions the path planner will calculate ahead of the currently commanded position. Of course, if the parameter is set too large the path planner may process many statements before the physical motion takes place. The path planner can't know about potential real-time events that may stop motion or require a program branch, this can result in much wasted calculation and CLC resources.

Decreasing the look-ahead value causes the path planner to process fewer coordinated motion program statements ahead of the current commanded position. This results in a lower potential for wasted calculations and a lighter load on CLC resources. However; motion geometry segments using blending may be missed if the look-ahead value is set too small.
**T-0-0024 Velocity Override**

Units: percent  
Default: 100.0 (disabled)  
Motion Types: All  
Attributes: Floating point, Read/write at any time

The velocity override provides a method to slow down all motion in a task equally. When a Velocity Override factor is specified for coordinated motion, all velocities in the point table are multiplied by this factor as they are used. When a Velocity Override is specified for non-coordinated motion that is generated by the DDS drive, each velocity command is multiplied by this factor before the command is executed.

**T-0-0025 Maximum Jog Increment**

Units: World Position  
Default: 1.0  
Attributes: Floating point, Read/write at any time

This parameter defines the maximum distance that is used for incremental coordinated jogging. The Large Increment and Small Increment percent parameters are based on this value.

**T-0-0026 Maximum Jog Velocity**

Units: World Velocity (units per minute)  
Default: 100.0  
Attributes: Floating point, Read/write at any time

This parameter defines the maximum velocity used for coordinated jogging. The Fast and Slow percent parameters are based on this value.
3.5.3. Robotics

T-0-0035 Relative Point Used for Origin

Units: none
Default: 0 (default origin is used)
Attributes: Integer, Non-volatile, Read/write at any time

This is the relative point in which the robot origin is stored. The point index in the REL table is stored here at the execution of robot/origin instruction so that the origin stays the same when jogging in manual mode. It can also be edited directly in parameter mode so that the origin can be set up before the execution of the instruction. **NOTE:** *Takes effect at robot/origin instruction execution and at exit from parameter mode.*

T-0-0036 Relative Point Used for Tool Frame

Units: none
Default: 0 (default origin is used)
Attributes: Integer, Non-volatile, Read/write at any time

This is the relative point in which the robot tool frame is stored. The point index in the REL table is stored here at the execution of robot/tool instruction so that the origin stays the same when jogging in manual mode. It can also be edited directly in parameter mode so that the tool frame can be set up before the execution of the instruction. **NOTE:** *Takes effect at robot/tool instruction execution and at exit from parameter mode.*

T-0-0050 Kinematic Value 1

*See T-0-0059 Kinematic Value 10 for description.*

T-0-0051 Kinematic Value 2

*See T-0-0059 Kinematic Value 10 for description.*

T-0-0052 Kinematic Value 3

*See T-0-0059 Kinematic Value 10 for description.*

T-0-0053 Kinematic Value 4

*See T-0-0059 Kinematic Value 10 for description.*

T-0-0054 Kinematic Value 5

*See T-0-0059 Kinematic Value 10 for description.*
T-0-0055 Kinematic Value 6
See T-0-0059 Kinematic Value 10 for description.

T-0-0056 Kinematic Value 7
See T-0-0059 Kinematic Value 10 for description.

T-0-0057 Kinematic Value 8
See T-0-0059 Kinematic Value 10 for description.

T-0-0058 Kinematic Value 9
See T-0-0059 Kinematic Value 10 for description.

T-0-0059 Kinematic Value 10

Default: based on kinematic
Attributes: Floating point, Read/Write in Parameter Mode.

These parameters are coefficients used in the kinematic equations of coordinated motion. Kinematics are unique to each application.

3.5.4. Coordinated Motion Status

These read-only parameters provide status values for each task.

T-0-0100 Target Point Number

Attributes: Integer, Read-only

This parameter shows the current target point. For example, if 10 is displayed here, motion to point ABS[10] or REL[10] is taking place, or the machine is currently at this point.
T-0-0101 Segment Status

Attributes: Integer, Read-only

This parameter shows the status of the current segment. The segment status codes are shown in the table below. The codes are valid for the current segment, excepting codes 0, 1 and 7 which are transitional or do not apply to the current segment. Use code 8 to check if the target position has been reached. Use code 6 to check if motion is halted due to a path/stop.

<table>
<thead>
<tr>
<th>Code</th>
<th>Segment Status</th>
<th>Activity</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Segment ready</td>
<td>Segment in queue for path planner</td>
</tr>
<tr>
<td>1</td>
<td>Acceleration</td>
<td>Acceleration in progress</td>
</tr>
<tr>
<td>2</td>
<td>Slew (constant speed)</td>
<td>Slew in progress (at target velocity)</td>
</tr>
<tr>
<td>3</td>
<td>Blending</td>
<td>Blending in progress</td>
</tr>
<tr>
<td>4</td>
<td>Target deceleration</td>
<td>Deceleration to target position in progress</td>
</tr>
<tr>
<td>5</td>
<td>Controlled stop</td>
<td>Controlled stop taking place (error, jog or path/stop)</td>
</tr>
<tr>
<td>6</td>
<td>Stopped</td>
<td>Motion has stopped (error, jog or path/stop)</td>
</tr>
<tr>
<td>7</td>
<td>At Target</td>
<td>Segment at target position</td>
</tr>
<tr>
<td>8</td>
<td>Done</td>
<td>Motion on the segment is complete</td>
</tr>
</tbody>
</table>

T-0-0102 Rate Limit Status

Codes: 0 = No rate limiting

1 = Rate limiting taking place on this segment

Attributes: Integer, Read-only

This parameter indicates if the coordinated motion path planner is performing rate limiting. If any of the axis speeds exceed the maximum values when a motion profile is calculated, the path planner calculates a new profile. If this value is set to "1", the CLC is spending more time in the motion instructions and the target speed and acceleration are being rate limited.

T-0-0111 Current X Position

Units: World Position

Motion Type: Coordinated

Attributes: Floating point, Read-only

This parameter returns the current commanded position of the X axis in world coordinates.
T-0-0112 Current Y Position

Units: World Position
Motion Type: Coordinated
Attributes: Floating point, Read-only

This parameter returns the current commanded position of the Y axis in world coordinates.

T-0-0113 Current Z Position

Units: World Position
Motion Type: Coordinated
Attributes: Floating point, Read-only

This parameter returns the current commanded position of the Z axis in world coordinates.

3.5.5. Task Status

T-0-0120 Task Operating Mode

Codes: 0 = initialization
1 = parameter
2 = manual
3 = automatic
Attributes: Integer, Read-only

This parameter returns the current operating mode of the task as a code number. Information about the task’s state, etc., is available in the CLC’s Control and Status I/O registers. See Chapter 2. CLC I/O Systems for further information on the Control and Status I/O registers.
T-0-0122 Task Diagnostic Message

Codes: 200-399: Warning error
400-599: Shutdown error
Attributes: ASCII string, Read-only

This parameter returns the current diagnostic message and/or code. During normal operation, a Message/Diag statement in the user program sets this message. If an error occurs during task execution, this diagnostic message is overwritten with an error message.

Examples:

<table>
<thead>
<tr>
<th>Picking up the part</th>
</tr>
</thead>
<tbody>
<tr>
<td>_ Message for Operator from message/diag</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>421 User Program Stack Overflow</th>
</tr>
</thead>
<tbody>
<tr>
<td>_ Task Error Message</td>
</tr>
<tr>
<td>_ Error code generated by CLC</td>
</tr>
</tbody>
</table>

T-0-0123 Task Status Message

Attributes: ASCII string, Read-only

This parameter returns the current status message for this task. A Message/Status command in the user program sets this message as an aid to the operator or for debugging purposes. This message is not overwritten with an error message, allowing debugging of an error condition set in the Task Diagnostic Message.

Examples:

<table>
<thead>
<tr>
<th>Executing Gripper Routine</th>
</tr>
</thead>
<tbody>
<tr>
<td>_ Debugging Message from message/status</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>No active message</th>
</tr>
</thead>
<tbody>
<tr>
<td>_ Default when no message is set in the program</td>
</tr>
</tbody>
</table>

T-0-0130 Current Instruction Pointer

Attributes: Hex Long, Read-only

This parameter returns a hexadecimal value equal to the current task’s execution address (i.e. the instruction pointer). The hex value is an offset from the start of the program. For example, "0x000000F0" indicates that the program counter is at 0xF0, or 240 bytes from the start of the program.
T-0-0131 Current Instruction

Attributes: ASCII string, Read-only

This parameter returns the mnemonic for the current instruction and the first 2 arguments of the instruction. The mnemonic is in the base code format generated by the CLC compiler. This parameter is primarily used for debugging and troubleshooting programs.

Example:

003C AXIS_WAIT

T-0-0132 Instruction Pointer at Error

Attributes: Hex Long, Read-only

As a future enhancement, the CLC can possibly execute an error routine if an error in a task occurs. This status parameter returns the instruction pointer where the error occurred.

T-0-0133 Composite Instruction Pointer

Attribute: ASCII string, Read-only

This parameter returns a string of flags and pointers to determine program flow in the task. The first number is a flag denoting the instruction is of the task (1), a subroutine (2 to 11), or an event (-2 to -12).

For instructions in the main task, the next number is the instruction pointer.

Example:

>0 TP 1.133
>0 TP 1.133 1 009C $24

For instructions in a subroutine of the task, the flag contains the count of pointers. Except for the last number, all pointers are to subroutine calls, starting in the main task. The last number is the instruction pointer in the subroutine.

Example 1:

>0 TP 1.133
>0 TP 1.133 2 0034 009C $24
Main task subroutine call and pointer in subroutine.
Example 2:
>0 TP 1.133
>0 TP 1.133 4 0034 009C 1205 1344 $24
Main task subroutine call, second call, third call, and pointer in subroutine.

For instructions in an event of the task, the flag contains the count of pointers. Except for the last two numbers, all pointers are to subroutine calls, starting in the main task. The second last number is the instruction pointer when interrupted. The last number is the instruction pointer in the event.

Example 1:
>0 TP 1.133
>0 TP 1.133 -2 0034 009C $24
Main task instruction pointer and pointer in event.

Example 2:
>0 TP 1.133
>0 TP 1.133 -4 0034 009C 1205 1344 $24
Main task subroutine call, second call, pointer in subroutine, and pointer in event.

T-0-0135 Current Subroutine

Attribute: ASCII String, Read-only

This is a string which indicates the current subroutine being executed with the function number and name. This includes tasks, non-accessible functions, functions, and subroutines. If function number and name information is not included in the user program file, the string “NONE” is printed.

To read data:
>0 TP x.135 ;Host asks for current subroutine for task ‘x’. (x=A-D or 1-4)
>0 TP x.135 2 test_sub ;The CLC responds with function number 2, subroutine ‘test_sub’

T-0-0136 Stack Variable Data

Attribute: Data string, Read-only

This is a string of data of current stack variables data. Stack variables are valid only while the program flow is within a task or subroutine. Maximum number of stack variables is 16. If there are no arguments or local variables in a task or function, the string “NONE” is returned.
To read data:

-0 TP x.136 ;Host ask for stack variables for task ‘x’.
-0 TP x.136 1.22 5 7.998 1 0 $24
  ;The CLC responds with values of all stack variables.

To get the types of these variables, use the “SR” command in the serial protocol with the current function number from Tx.135. This command returns a string for each of a function argument’s attributes.

**T-0-0137 Task Subroutine Breakpoint**

**Attribute:** integer, Read/Write

This task parameter specifies the index of the subroutine to halt program execution at when task breakpoint is enabled. Task breakpoint enable is bit 11 of the “Task_Control” register. Task program flow continues with a 0->1 transition on “Cycle_Start” bit of the “Task_Control” register. To get the count and names of subroutines of a program use:

-0 ST 0.0 ;Host ask for count of functions of currently active program
-0 ST 0.0 23 $24 ;CLC return count of 23

-0 ST 0.1 ;Host ask for function at index 1.
-0 ST 0.1 Task_A $24 ;CLC return “Task_A”

The list of all function of a program can thus be uploaded. To set a breakpoint put the function index in the “Task Subroutine Breakpoint” parameter.

-0 TP x.137 5 $24 ;Host put function index 5 in task x breakpoint parameter.
-0 TP x.137 $24
T-0-0138 Sequencer Information

Attribute: String, Read Only

This task parameter displays information about the currently running sequencer in both index and name format. The index indicates the current row in the sequence list or step list. If no sequencer is running, the string “NONE” is printed.

Otherwise, the CLC responds in the following format:

sequence_number sequence_row step_row sequence_name step_name function_name

Example:
> 0 TP x.138 ;Host asks for sequencer information
> 0 TP x.138 1 3 2 sequence_one Open_Mold move_fwd ;CLC responds with row numbers and names

T-0-0200 Last Active Event Number

Attributes: Integer, Read-only

This parameter returns the index of the current or last active event in the event (EVT) table. The value can be used to access other information (message, status, function, etc.) contained in the event table.

For example, if "99" is returned, EVT[99] is either executing or was the last event completed for this task.

3.5.6. Task Parameter Lists

T-0-2000 List of All Parameters

This is a multiple-step List Parameter that requests a list of all the parameter numbers in this parameter set. The 'D' subclass of the serial port protocol is used to list these parameters.

T-0-2001 List of Required Parameters

This is a multiple-step List Parameter that requests a list of all the parameters stored in nonvolatile CLC RAM that must be set for proper CLC operation. The 'D' subclass of the serial port protocol is used to list these parameters.
3.6. Axis Parameters

Class A

The CLC Axis Parameters are used to configure the axis and provide limits for coordinated motion. Some parameters only apply to a specific axis mode (coordinated or single-axis). This is noted by the Motion Type in each parameter description.

3.6.1. Axis Setup

A-0-0001 Task Assignment

<table>
<thead>
<tr>
<th>Type:</th>
<th>Integer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Codes:</td>
<td>0 = No task selected</td>
</tr>
<tr>
<td></td>
<td>1 = Task A</td>
</tr>
<tr>
<td></td>
<td>2 = Task B</td>
</tr>
<tr>
<td></td>
<td>3 = Task C</td>
</tr>
<tr>
<td></td>
<td>4 = Task D</td>
</tr>
<tr>
<td>Motion Type:</td>
<td>Coordinated and Single Axis</td>
</tr>
<tr>
<td>Default:</td>
<td>0, or axis selected in program</td>
</tr>
<tr>
<td>Attributes:</td>
<td>Write in Parameter mode only</td>
</tr>
<tr>
<td></td>
<td>(Automatically set when program activated.)</td>
</tr>
</tbody>
</table>

This parameter associates an axis with a user task. The TASK/AXES command and the Axis Setup Icon automatically set the parameter when a program is activated.

A-0-0002 Type of Positioning

| Codes:   | 0 = Normal Positioning |
|          | 1 = Lagless Positioning |
| Default: | 1 (lagless) |
| Attributes: | Integer, Write in Parameter Mode Only |

This selects the SERCOS drive control mode. For coordinated, ELS phase, Cam, and single-axis motion, it switches between normal and lagless positioning modes.
A-0-0003 Axis Motion Type

<table>
<thead>
<tr>
<th>Selections</th>
<th>Description</th>
<th>Default</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Disabled</td>
<td>0, or axis selected in program</td>
<td>Integer, Write only when the CLC is in Parameter Mode</td>
</tr>
<tr>
<td>1</td>
<td>Single Axis</td>
<td></td>
<td>(Automatically set when the user program is compiled)</td>
</tr>
<tr>
<td>2</td>
<td>Coordinated Axis</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Velocity Mode</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Ratio Slave</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>ELS Slave</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Torque Mode</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>CLC Cam Axis</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This parameter selects the type of motion for an axis and enables the axis/drive address on the SERCOS ring. The applicable SERCOS parameters are automatically sent to the drive. If a user program is present on the card, this parameter is automatically set at exit from parameter mode based on the commands that set up coordinated or single-axis motion.

**Single Axis** runs the axis as a single, non-coordinated axis. The CLC sends only the position to the intelligent DDS drive. The DDS drive generates and controls the position motion profile.

**Coordinated Axis** is used for coordinated circular or linear motion profiles generated by the CLC path planner. The axes used for each path are determined by user program statements.

**Velocity Mode** operates the axis at constant velocity without a position control loop. The CLC sends only velocity commands to the DDS drive. The intelligent DDS maintains the velocity profile.

**Ratio Slave** designates the axis as a slave to the master axis designated by Parameter As.30. The axis' drive operates in position loop mode, with its commanded velocity equal to the product of the master axis velocity times a ratio. The drive remains in position loop mode and at the commanded velocity at all times, even when the user task is not running.

**ELS Slave** designates the axis as an Electronic Line Shaft (ELS) slave.

**Torque Mode Axis:** The axis runs in torque modes, with torque commands sent from the CLC and no velocity or position loop.

**CLC Cam Axis:** The axis runs from a cam table stored on the CLC and uses the ELS virtual or real master.
A-0-0004 Axis Options

Defaults: 000000000000001 (for single-axis, velocity, and coordinated axes)
0000000000000011 (for ELS axes)
Motion Type: All types
Attributes: Binary Word, Non-volatile, Write in Parameter Mode only or at compile time

Bits 1-16:

<table>
<thead>
<tr>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
</table>

Bit 1: Position Initialization
0 = Reset feedback in Phase 2
1 = Keep feedback value in Phase 2
Default: (1)

When a single-turn absolute or incremental encoder is used with the Indramat DDS drive, the drive defaults to resetting the encoder to a one-turn measurement each time it exits SERCOS Phase 2. The DDS includes a parameter "Starting Position Value" to override the default and initialize the feedback to the value specified.

When the Position Initialization bit is set to 1, the CLC sets the drive’s Starting Position Value to the Position Feedback, and the drive’s position remains the same after a switch out of parameter mode. When the Position Initialization bit is set to 0, a different Starting Position Value can be entered through the SERCOS service channel, or the feedback value will be reset to the default one-revolution measurement.

This parameter does not affect a multi-turn absolute encoder feedback. To set the multi-turn encoder’s position, follow the procedure described in the DDS manual, using the "Set Absolute Position Measurement" and homing commands.

Bit 2: Positioning Mode
0 = Linear Positioning Mode
1 = Rotary Positioning Mode
Default: (0)

The positioning mode is valid for all axis types. All relevant scaling parameters are automatically set in the drives. If an axis is an ELS slave, its mode is automatically set to rotary positioning.
**Linear Positioning Mode**
When this bit is set to 0, linear positioning is selected. The units and scaling are specified with the Axis Position Units parameter. Absolute positioning is enabled in the drive.

**Rotary Positioning Mode**
When this bit is set to 1, rotary positioning is enabled. The position, velocity, and acceleration units and scaling are fixed at the drive. Modulo positioning is enabled by default, with a rollover value specified in Drive parameter S-0-0103.

When rotary positioning mode is enabled in the drive, position is in degrees, velocity in RPM, and acceleration in radians/sec. Single-axis and velocity mode values are entered in these units. Events for single-axis and velocity mode work with both rotary and linear positioning.

Coordinated motion and events are compatible with rotary positioning only if motion takes place within the modulo value and does not roll over. If an axis is coordinated, velocity is in linear units/sec and acceleration in units/sec.

**Bits 3 and 4: ELS Synchronization Mode**
This option sets the type of synchronization for an electronic line shaft axis. These bits are set automatically at program activation by the ELS/INIT command.

<table>
<thead>
<tr>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Velocity Synchronization</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The axis runs in velocity mode, with its velocity equal to the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>master velocity times the translation ratio.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The ratio can be fine-adjusted at run time.</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Phase Synchronization</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The axis runs in the operating mode selected in parameter As.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(lagless positioning or positioning with lag), and</td>
</tr>
<tr>
<td></td>
<td></td>
<td>maintains a phase relationship according to the phase offset</td>
</tr>
<tr>
<td></td>
<td></td>
<td>and translation ratio parameters. The phase offset can be</td>
</tr>
<tr>
<td></td>
<td></td>
<td>adjusted at run time.</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Cam Table</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The axis is linked to a cam and synchronized to a master.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Position relationship is maintained according to cam table,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ratio and stretch factors. Position offset can be adjusted at</td>
</tr>
<tr>
<td></td>
<td></td>
<td>run time.</td>
</tr>
</tbody>
</table>
**Bit 5: DIAX01 Drive**
0 = Indramat DDS or other SERCOS standard drive
1 = Indramat DIAX01 Glossary_DIAX01
Default: (0)
Only on versions GPS-01.1x. This bit is no longer used on the CLC-D/P or the CLC-V 2.3, because the drive type is automatically detected and configured. On the CLC-V 2.1, it is necessary to set this bit to a 1 if a DIAX01 drive is used.

**Bit 6: Optional Cyclic Data**
0 = Use smallest cyclic data configuration
1 = Include optional cyclic data (velocity feedback, programmed acceleration)
Default: (0)

The drives are set up for the smallest amount of cyclic data to increase the number of drives on the ring and increase cycle time. Velocity feedback is normally read through the service channel, which can take up to 50ms using a program command. To include velocity feedback in the cyclic data for all axis modes, set this bit to 1. In single-axis mode, programmed acceleration is also sent cyclically instead of through the service channel.

In CLC versions that use the SERCOS ASIC (software versions 01.20 and greater), up to five optional IDN’s can be specified with parameters 180-196 in addition to or exclusive of this option bit.

**Bit 7: Velocity Mode Ramp**
0 = Velocity mode step command
1 = Velocity mode uses programmed acceleration
Default: (0)

When velocity mode is selected as an axis type, the drive commands the velocity that is sent to the CLC by default (bit=0). If bit 7 is set to 1, the CLC will accelerate and decelerate the axis as if it were in single-axis mode.

**Bit 8: Reserved for Future Use**

**Bit 9: ELS Secondary Mode**
0 = secondary mode is single-axis mode
1 = secondary mode is velocity mode
Default: (0)

For an ELS axis, this bit selects the secondary, non-synchronized mode for a DDS drive. This is the default mode until the ELS/MODE command switches the axis into ELS velocity or phase synchronous mode.
When this bit is set to (0), the default secondary mode is single-axis positioning mode. Velocity mode may also be switched to using the ELS/MODE command, if it is configured in the cyclic data using parameters A-0-0180 Optional Command ID #1 to A-0-0196.

When it is set to (1), the secondary mode is velocity mode. On drives that don’t support single-axis mode, the secondary mode is always velocity mode, regardless of this bit.

**Bit 10: Disable ELS Shortest Path**

0 = Shortest path positioning for ELS phase adjust
1 = Shortest path disabled
default: (0)

This bit selects the positioning method used for the ELS phase adjust move. When it is set to (0), the axis takes the shortest path within a revolution. If the difference in position is between 180 and 360 degrees, the axis travels counterclockwise.

**GPx-02.00 and later:** When this bit is set to (1), shortest path is disabled. A move within one revolution travels in the positive direction if the programmed position is positive, and in the negative if it is negative.

**GPx-01.1x and GPS-01.2x:** When this bit is set to (1), shortest path is disabled. A move within one revolution always travels in the positive direction.

**Bit 11: Positioning Using Secondary Encoder**

0 = Use primary feedback (encoder 1)
1 = Use secondary feedback (encoder 2)
Default: (0)

This bit configures the drive to use Encoder 2 to close the position loop and provide cyclic feedback from drive parameter S-0-0053. This option must be set if the CLC is used with DDS-2.1 linear motor firmware.

**Bit 12: Drive Disable Method**

0 = Stop axis immediately
1 = Coast to a stop
Default: (0)

This bit changes the response of the drive after a fatal error or when the disable bit in the axis control register is set. It configures the way the CLC sets the enable bits (14 and 15) in the SERCOS control word (D-1.00134). When it is set to (0) (default), the drive immediately commands zero velocity before disabling torque and applying the brake. This option should be
used for coordinated motion or linear motion, where coasting can cause damage or injury.

When it is set to (1), the drive immediately disables torque, causing the drive to coast, stopping with its own inertia. This should be used in some types of lineshafting applications, where immediate disabling of the drive could cause damage.

**Bit 13: Linear Axis Modulo Positioning**

0 = Linear axis uses absolute positioning  
1 = Linear axis uses modulo positioning  
Default: (0)

When it is set to (0), absolute motion is enabled, with signed positions and no modulo. This option should be used for coordinated motion and most absolute positioning applications.

When it is set to (1), the modulo value in S-0-0103 is used. When the axis position reaches the modulo value, it resets to 0. There are no negative position values. Unlike rotary mode, the scaling of position, velocity, and acceleration is linear (inches or mm). This option should be used for continuous indexing operations. It works only on DDS versions DSM2.1-S01.10 and greater and CLC versions 01.20 and greater.

**Bit 14: Configure Minimum Cyclic Data**

0 = Default or maximum cyclic data  
1 = Minimum cyclic data  
Default: (0)

**ELS or Cam Axes**

When set to (0), the velocity command is included in the cyclic data, which allows real-time update of velocity when the axis is in its secondary mode.

When set to (1), the velocity command is removed from the cyclic data. Because the velocity command is through the service channel only, no ramping or real-time control can be done on the axis. Therefore, the axis should only be run in ELS synchronization mode.

**Single-axis mode**

When set to (0), the programmed velocity is included in the cyclic data for applications where the velocity is changed often in the user program.

When set to (1), the programmed velocity is removed from the cyclic telegram to allow the maximum number of drives and options and the minimum SERCOS cycle time. The velocity is then sent through the service channel. This option is not recommended in applications where the velocity is changed often in the user program.

**Bit 15: Disable Modulo Positioning for Rotary Axis**

0 = Modulo positioning
1 = Absolute positioning
Default: (0)

When this bit is (1) modulo positioning is disabled if an axis is rotary. Positions less than zero are negative, and the modulo parameter is not used. This should be used only with positioning applications, not with indexing, ELS, or continuous motion. If the bit is set to (0) (default), the rotary axis position resets to the modulo value every revolution.

**Bit 16: Disable Automatic Scaling**

0 = CLC sets SERCOS scaling parameters (RECOMMENDED)
1 = CLC does not set SERCOS scaling parameters
Default: (0)

For most drives, the CLC sets SERCOS scaling parameters such as S-0-0044 and S-0-0076. If a drive does not accept the CLC’s parameter settings, these parameters can be set manually. Note: This bit should only be set at the recommendation of the Indramat applications department.

**A-0-0005 Linear Position Units**

<table>
<thead>
<tr>
<th>Selections</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>inches</td>
</tr>
<tr>
<td>1</td>
<td>millimeters</td>
</tr>
<tr>
<td>2</td>
<td>radians</td>
</tr>
</tbody>
</table>

Default: set automatically if coordinated motion, else 0 (inches)

Attributes: Integer, Write in Parameter Mode only

This parameter selects the display and scaling units for linear axis position, speed, and acceleration data. No unit conversions are performed when changing this parameter.

The drive’s display and scaling parameters are automatically set by the CLC. All data is sent in floating point format with the same resolution as the data sent to the drive. Velocity, acceleration, and jerk data is always sent with three decimal place accuracy. The following decimal places are used for position data:

<table>
<thead>
<tr>
<th>Units</th>
<th>Decimals</th>
<th>Max Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>inch</td>
<td>5</td>
<td>21474.83648</td>
</tr>
<tr>
<td>mm</td>
<td>4</td>
<td>214748.3648</td>
</tr>
<tr>
<td>rad</td>
<td>6</td>
<td>2147.483648</td>
</tr>
</tbody>
</table>

If an axis is in rotary mode, the drive automatically sets the position to degrees, the velocity to RPM, and the acceleration to radians/second/second. In rotary mode, the scaling and display units in this parameter do not apply.
If an axis is selected for coordinated motion, this parameter is automatically set on exiting parameter mode. If kinematic 1 is selected, it is set to the value stored in the Task Position Units parameter (3.5.2. Coordinated Motion)

T-0-0005 World Position Units). If any other kinematic is selected, it is set to radians (2) for use as a kinematic joint.

**NOTE:** Changing this parameter affects the scaling of drive parameters (i.e., the feed constant). If you need to retain such constants, you must explicitly save each required drive mechanical parameter before changing the Linear Position Units parameter.

**A-0-0006 Reference Options**

- **Attributes:** Write in parameter mode only, Binary
- **Default:** all bits = 0

This parameter selects options for reference position monitoring and homing.

**Bit 1: Issue error when drive is enabled**

When this bit is set to (1), the CLC immediately issues the error “500 Axis D is not referenced” if the drive is enabled while the position is not referenced. The CLC reads drive parameter S-0-0403, Position Feedback Status to determine if the position is referenced. While the drive is disabled, the Set Absolute Encoder procedure can be used to set the reference position.

This option should be set only when an absolute encoder is used, since it will prevent the incremental homing procedure from being used.

**Bits 2-16: Reserved for Future Use**

Other options will be added in the future.
A-0-0007 Configuration Mode

Default: 0  
Attributes: Integer, Write in Parameter Mode only

This parameter allows drives to be excluded from the user program and initialized to single-axis or velocity mode. An error will not be issued if the drive is not found on the SERCOS ring. When this parameter is set to (0), the axis can be used in the program with its defined axis type, and its presence on the ring will be verified. When it is set to (1), the drive is excluded from the program, but can be jogged using default values. When it is set to (2), the drive is not configured, and is put into a torque-free mode after initialization. The default value of this parameter is (0). It can be written in parameter mode only.

<table>
<thead>
<tr>
<th>A-0-0007</th>
<th>Selection in user program</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Drive can jog in default mode</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Drive can jog in default mode</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>Drive not configured, motion cannot be performed</td>
</tr>
<tr>
<td>0</td>
<td>nonzero</td>
<td>Drive is configured according to selection in A-0-0003, presence on ring is checked.</td>
</tr>
<tr>
<td>1</td>
<td>nonzero</td>
<td>user program selection is bypassed, drive can jog in default mode</td>
</tr>
<tr>
<td>2</td>
<td>nonzero</td>
<td>user program selection is bypassed, motion cannot be performed</td>
</tr>
</tbody>
</table>

A-0-0009 Drive PLS Register

Default: 0 (disabled)  
Motion Type: All types  
Attributes: Integer, non-volatile, Read/write at any time

This parameter selects the register to which the drive-based PLS will be associated. The register selected here can be read by the user program, the I-O mapper, or the user interface as a status of the current PLS outputs. If this parameter is set to 0, the drive-based PLS for this axis is disabled.

If the register is associated with a DEA card, the drive internally updates the outputs every 2ms. If the register is not assigned to a DEA, this register will updated with the PLS status bits read from the drive every SERCOS cycle.

SERCOS config. note: The PLS Status IDN P-0-0135 will be placed in the AT in the cyclic data, with 2 bytes allocated. Parameter P-0-0124 (Assign IDN->DEA) is automatically set by the CLC if this register is a DEA output register.
A-0-0010 Start of Drive I-O

- Default: for Axes 1-8, automatically set to defaults (see I-O description in manual) for Axes 9-40, set to 0 (no register allocated)
- Motion Type: All types
- Attributes: Integer, non-volatile, Read/write in Parameter Mode only

This parameter selects the starting register number that correspond to the I-O cards that reside in the drive. The configuration for an Indramat DIAx02 or DIAx03 drive is shown below:

<table>
<thead>
<tr>
<th>Register Number</th>
<th>Parameter used to Enable</th>
<th>Description</th>
<th>I-O Card</th>
</tr>
</thead>
<tbody>
<tr>
<td>A-0-0010</td>
<td>A-0-0011</td>
<td>Inputs, Card 1</td>
<td>DEA4.1</td>
</tr>
<tr>
<td>(A-0-0010) + 1</td>
<td>A-0-0011</td>
<td>Outputs, Card 1</td>
<td>DEA4.1</td>
</tr>
<tr>
<td>(A-0-0010)+ 2</td>
<td>A-0-0014</td>
<td>Inputs, Card 2</td>
<td>DEA5.1</td>
</tr>
<tr>
<td>(A-0-0010)+ 3</td>
<td>A-0-0014</td>
<td>Outputs, Card 2</td>
<td>DEA5.1</td>
</tr>
<tr>
<td>(A-0-0010)+ 4</td>
<td>A-0-0017</td>
<td>Inputs, Card 3 or Analog Input 1</td>
<td>DEA6.1 or DRF</td>
</tr>
<tr>
<td>(A-0-0010)+ 5</td>
<td>A-0-0017</td>
<td>Outputs, Card 3 or Analog Input 2</td>
<td>DEA6.1 or DRF</td>
</tr>
</tbody>
</table>

A-0-0011 Drive I/O Card 1 Setup

- Type: Integer
- Codes: 0 = disabled
1 = enable DEA 4.1 parallel I-O
- Default: 0
- Attributes: Read/Write in Parameter Mode Only

The Indramat DDS digital drive can include up to three optional parallel I/O cards that provide 16 outputs and 15 inputs. If this parameter is set to '1', the first I/O card is enabled and its contents are transmitted in the cyclic telegram to and from the corresponding I/O registers.

A-0-0014 Drive I/O Card 2 Setup

- Type: Integer
- Codes: 0 = disabled
1 = enable DEA 5.1 parallel I-O
- Default: 0
- Attributes: Read/Write in Parameter Mode Only

The Indramat DDS digital drive can include up to three optional parallel I/O cards that provide 16 outputs and 15 inputs. If this parameter is set to '1', the second I/O card is enabled and its contents are transmitted in the cyclic telegram to and from the corresponding I/O registers.
A-0-0017 Drive I/O Card 3 Setup

| Type:  | Integer |
| Codes: | 0 = disabled |
|       | 1 = enable DEA 6.1 parallel I-O |
|       | 2 = enable analog input 1 on DRF card |
|       | 3 = enable both analog inputs on DRF |
| Default: | 0 |
| Attributes: | Read/Write in Parameter Mode Only |

This parameter is used to select DDS analog or parallel inputs. Options allow selection of a DEA6.1 parallel I-O card (1), analog input 1 on the DRF card (2), or analog inputs 1 and 2 (3). Cyclic data can be optimized if only one analog input is needed by selecting (2) instead of (3).

**Analog Inputs:**
The analog inputs on the DRF card are 16-bit values which can be read as parameters or from I-O registers. The inputs are converted and scaled by the DDS according to the amplification selected on the connector. The analog input 1 value is stored in the register for parallel input 3 and the analog input 2 in the parallel output 3 register. For example, on the CLC 1.1, if A-0-0017 Drive I/O Card 3 Setup is set to 2, analog input 1 value is in register 44. If parameter A2.17 is set to 3, analog input 1 value is in register 50, and analog input 2 value in register 51.

A-0-0020 Maximum Velocity

| Units:  | Axis Velocity (units/min) |
| Default: | 1000.0 |
| Motion Type: | All types |
| Attributes: | Floating point, Read/write at any time |

The axis velocity is limited to this value during a coordinated move. Note that the Bipolar Velocity limit on the drive (S-0-0091) must also be set, and it is independent of this parameter. For single-axis or velocity mode motion, this is the maximum velocity that can be programmed.

A-0-0021 Maximum Acceleration

| Units:  | Axis Acceleration (units/sec^2) |
| Default: | 200.0 |
| Motion Type: | All types |
| Attributes: | Floating point, Read/write at any time |

The axis acceleration is limited to this value during a coordinated move. On the drive level, the acceleration capability is limited by the amount of peak torque that the drive and motor are rated for, and the Bipolar Acceleration limit parameter if it is set. For single-axis or velocity mode motion, this is the maximum acceleration that can be programmed.
A-0-0022 Maximum Deceleration

Units: Axis Acceleration (units/sec^2)
Default: 200.0
Motion Type: All types
Attributes: Floating point, Read/write at any time

The deceleration for this axis is limited to this value during a coordinated move. At the DDS drive level, the acceleration capability is limited to the rated drive or motor peak torque; and, if set, the drives’ Bipolar Acceleration limit parameter.

A-0-0023 Jog Acceleration

Type: Integer
Units: Percent of Maximum Acceleration
Limits: 20% to 100%
Default: 100%
Attributes: Read/write at any time

This parameter sets the acceleration and deceleration rate for axis jogging in manual mode. It is entered as a percentage of the maximum acceleration (A-0-0021 Maximum Acceleration).

A-0-0025 Maximum Jog Increment

Units: linear: position units
        rotary: revolutions per minute
Default: 1.0
Attributes: Floating point, Read/write at any time

This parameter defines the maximum distance used for incremental single-axis jogging. The Large Increment and Small Increment percent parameters are based on this value.

A-0-0026 Maximum Jog Velocity

Units: linear: position units per minute
        rotary: revolutions per minute
Default: 100.0
Attributes: Floating point, Read/write at any time

This parameter defines the maximum velocity used for jogging an axis in single-axis mode, velocity mode, or as a joint. The system fast and slow axis percent parameters are multiplied by this value to calculate jog velocity.
A-0-0030 Ratio Mode Master Axis

Selections: 1 to maximum number of axes
Motion Type: Ratio Slave
Attributes: Integer, Write in Parameter Mode only
   (Automatically set when the user program is compiled.)

This parameter displays the axis that is used as the master for ratio, ELS, or Cam functions. If an axis is slaved to the ELS virtual master, it is set to 0. If a real master is used, it is set to the axis on which the encoder resides. For the ratio function, the master axis is set independently for each slave at program compile time. The velocity feedback of the master is multiplied by the ratio established by parameters A-0-0031 CLC Cam/Ratio Master Factor (N) and A-0-0032 CLC Cam/Ratio Slave Factor (M).

A-0-0031 CLC Cam/Ratio Master Factor (N)

Limits: -100000 to 100000
Motion Type: Ratio Slave or Cam
Attributes: Floating point, Read/write at any time
   (This parameter is also set by the AXIS/RATIO instruction.)

The slave-to-master ratio is defined using two parameters, A-0-0031 and A-0-0032 CLC Cam/Ratio Slave Factor (M). This allows the CLC to normalize the ratio calculation, preserving full system accuracy for repeating-decimal ratios such as 2/3. Both parameters must be set or a run-time error may occur.

In Ratio mode, the velocity of the slave is determined by:
    \[ V_{\text{slave}} = V_{\text{master}} \times \left( \frac{K_{\text{slave}}}{K_{\text{master}}} \right) \]

where:
    \[ V_{\text{slave}} \] = Velocity of the slave axis
    \[ V_{\text{master}} \] = of the master axis
    \[ K_{\text{slave}} \] = Slave factor set by Parameter A-0-0032
    \[ K_{\text{master}} \] = Master factor set by Parameter A-0-0031

A-0-0032 CLC Cam/Ratio Slave Factor (M)

Same format as parameter A-0-0031 CLC Cam/Ratio Master Factor (N)
A-0-0033 CLC Cam Stretch Factor (H)

Default: 1.0
Limits: -100000.0 to 100000.0
Motion Type: CLC Cam
Attributes: Float, Read/write at any time.

This is the stretch factor (H) for Cam motion on the CLC. Every position at the output of the cam is multiplied by this value.

A-0-0034 CLC Cam Currently Active

Default: 0
Limits: 0- max. number of cams
Motion Type: CLC Cam
Attributes: Integer, Read/write at any time

This is the cam number that is currently active for this axis. If the cam is set to 0, the axis directly follows the master axis. The cam activation only takes affect after the master has passed zero degrees or when the master is stopped.

A-0-0035 CLC Cam Linear Scaling (L)

Default: 0.0
Limits: -100000.0 to 100000.0
Motion Type: CLC Cam
Attributes: Float, Read/write at any time.

This factor for Cam motion on the CLC allows the master position to be added to the cam position, according to the equation:

\[ \phi_s = L \cdot \phi_m + H \cdot \text{CamTable}(z) \]
A-0-0036 Ratio Mode Encoder Type

Attributes: Read/write in parameter mode only, integer
Units: None
Selections: 0 if not used or primary feedback is used
1 if primary feedback is used
2 if secondary feedback is used.

This option selects the type of master used for the CLC Ratio Mode. The primary encoder or secondary encoder can be used. Options (0) and (1) select the master drive’s primary feedback, which is the value read from drive parameter S-0-0051. Option (2) selects the secondary feedback, read cyclically from drive parameter S-0-0053 (feedback 2). This parameter is set automatically from the axisinitialize instruction.

A-0-0037 Ratio Mode Step Rate

Format: Floating Point
Storage: volatile, read/write at any time
Motion Types: Ratio Mode
Default: 0.0 (instantaneous change)
Units: units/ second

This parameter sets the rate used in CLC Ratio Mode when the Ratio parameters A-0-0032 CLC Cam/Ratio Slave Factor (M) and A-0-0033 CLC Cam Stretch Factor (H) are changed, either directly or through the axis/ratio command. If the step rate is set to 0, the ratio will be changed immediately without a ramp.

Example: The current ratio is 0, the programmed ratio is 10:1, and the step rate parameter is set to 10 units/sec. The ratio will be ramped for one second until it reaches the target value.
3.6.2. Axis Status

These read-only parameters provide status values for the specified axis.

Note: The feedback values are obtained from the drive through the cyclic SERCOS telegram rather than through the service channel. They should be read from the user program or user interface, since they will be updated much faster than using the service channel.

A-0-0100 Target Position

Units: Axis Position  
Motion Type: Single-axis  
Attributes: Floating point, Read/write at any time

DIAx02 Drives
This is the programmed position used by the drive in Single-axis mode, equivalent to parameter P-0-0049

DIAx03, ECODRIVE, SERCOS Standard
This is the programmed position used by the drive in Single-axis mode, equivalent to parameter S-0-0258.

A-0-0101 Commanded Position

Units: Axis Position  
Motion Type: Coordinated  
Attributes: Floating point, Read-only

This is the commanded position used for coordinated moves, the cyclic equivalent of SERCOS parameter S-0-0047.

A-0-0102 Feedback Position

Units: Axis Position  
Motion Type: All types  
Attributes: Floating point, Read-only

This is the position feedback from the drive, equivalent to SERCOS parameter S-0-0051 in Phase 4. It is not updated in phases 0-3.
A-0-0110 Programmed Velocity

Units: Axis Velocity (units/min)
Motion Type: All types except coordinated
Attributes: Floating point, Read/write at any time

DIAX02 Drives
This is the programmed velocity used by the drive in Single-axis mode, equivalent to parameter S-0-0091.

DIAX03, ECODRIVE, SERCOS Standard
This is the programmed velocity used by the drive in Single-axis mode, equivalent to parameter S-0-0259.

A-0-0111 Commanded Velocity

Units: Axis Velocity (units/min)
Motion Type: Coordinated and Velocity
Attributes: Floating point, Read/write at any time
(Automatically set when the user program is compiled.)

This is the commanded velocity, which the drive generates in coordinated mode, and the CLC generates in Velocity mode. It is equivalent to SERCOS parameter S-0-0036.

A-0-0112 Feedback Velocity

Units: Axis Velocity (units/min)
Motion Type: All types
Attributes: Floating point, Read-only

This is the velocity feedback from the drive, equivalent to SERCOS parameter S-0-0040 in Phase 4.

NOTE: If velocity isn’t configured in the cyclic telegram, the CLC reads this value from the service channel.
A-0-0120 Programmed Acceleration

Units: Axis Acceleration (units/sec^2)
Motion Type: All types except coordinated
Attributes: Floating point, Read/write at any time

DIAx02 Drives
This is the programmed acceleration used by the drive in Single-axis mode, equivalent to parameter S-0-0138.

DIAx03, ECODRIVE, SERCOS Standard
This is the programmed acceleration used by the drive in Single-axis mode, equivalent to parameter S-0-0260.

A-0-0131 SERCOS Control Word

Motion Type: All
Attributes: Binary word, Read-only

This is the current value of the SERCOS master control word (S-0-0134).

A-0-0132 SERCOS Status Word

Motion Type: All
Attributes: Binary word, Read-only

This is the current value of the SERCOS drive status word (S-0-0135).

A-0-0133 AT Error Count

Attributes: Integer, Read-only

The AT error counter can be used for troubleshooting of SERCOS connections. If the value of this parameter is increasing while it is being displayed, there may be a noisy SERCOS connection or a faulty communication card on the drive associated with this axis.

If two consecutive AT’s are invalid, the CLC will issue a SERCOS disconnect error. This counter counts up to a maximum of 65535. If a large value is set in the counter, there may have been a noisy transmission over a long period of time.
A-0-0140 Mfg. Class 3 Status Word

Type: Binary Word
Motion Type: Single-axis, ELS
Attributes: Read-only

This is the cyclic equivalent of parameter S-0-0182 on Indramat drives. It is always included in the cyclic data in single-axis mode. In ELS mode, it is included only if the phase adjust is executed on the drive. The CLC sets the axis status register based on some of these bits.

Bits (1=LS bit)
1   AS input is active (DIAx03, ECODRIVE)
5   Set if target position exceeds position limit value
6   Set if programmed velocity exceeds max motor speed
7   Set if axis is in-position and target position reached
9   Set if ELS phase adjust or velocity adjust is complete (DIAx03)
12  Set if drive halt was completed

A-0-0141 Torque Mode Commanded Torque

Units: Percent of motor standstill (stall) torque
Attributes: Floating point, Read/write at any time

This is the cyclic equivalent of drive parameter S-0-0080, Torque Command. It is set with a torque command from the CLC user program when an axis is in Torque Mode.

A-0-0142 Torque Feedback (cyclic)

Units: Percent of motor standstill (stall) torque
Attributes: Floating point, Read-only

This is the cyclic equivalent of drive parameter S-0-0084, Torque Feedback. It is updated only in Torque Mode.

A-0-0145 Current Motion Type

Codes: See Parameter A-0-0003 Axis Motion Type
Attributes: Integer, Read-only

The CLC switches ELS, Cam, Ratio, and Torque Mode axes to different operating modes for jogging and setup. This parameter displays the current motion type corresponding to the drive's operating mode. For an ELS axis in synchronization mode, Ratioed Axis (4) is printed. When it is not in sync. mode, it is either in single-axis (1) or velocity (3) mode.
3.6.3. Electronic Line Shaft

A-0-0150 Programmed Ratio Adjust

Units: Percent
Default: read from drive at switch to phase 4
Limits: -100.0 (stopped) to 300.0 (4 times master speed)
Attributes: Floating point, read/write at any time

This is the ratio fine adjust parameter, corresponding to drive parameter P-0-0083. It is adjusted from -100 to 300 percent using the ELS/ADJUST command. If fine adult is included in the cyclic data, it is updated every SERCOS cycle and may be adjusted using a ramp (A-0-0159 Ratio Adjust Step Rate). Otherwise, the drive’s service channel is used.

A-0-0151 Programmed Phase Offset

Units: degrees (modulo 360)
Attributes: Floating point, Read/write at any time

This is the programmed phase offset value used in position synchronization and drive cam modes. The drive executes a position profile with acceleration (P-0-0142) and velocity limit (P-0-0143) when this parameter is changed. This value corresponds to drive parameter S-0-0048, in the cyclic data. Set through the ELS_Adjust or Cam_Adjust command.

Note: When dynamic synchronization is enabled on the drive, parameters A-0-0152 through A-0-0163 are not used.

A-0-0153 CLC Cam Adjust Average Velocity

Units: RPM
Default: 10.0
Attributes: Floating point, Read/write at any time

When the phase offset is changed, the CLC performs an absolute positioning move in addition to the ELS master command. This is the target velocity used for the phase offset adjustment move.

In versions GPS-01.30 and above, the time constant parameter A-0-0155 CLC Cam Adjust Time Constant can be used to automatically set the velocity for the phase offset based on the time of the move. When A-0-0153 is used, it sets the average velocity of the phase offset move. Note that the peak velocity will be up to twice as large as this value.

Versions GPS-02.00 and greater: This parameter is overwritten with the expected peak velocity that will be added to the axis velocity when A-0-0155 is set to a value other than 0.
A-0-0155 CLC Cam Adjust Time Constant

Units: seconds
Default: 0.0 (disabled)
Limits: 0 to 60
Attributes: Floating point, read/write at any time

The CLC uses a filter to implement a jerk limited profile for the phase adjust move.

This parameter sets the amount of time that the move will require, regardless of the position. This can be calculated in the user program so that the phase adjust is always distributed for the length of one part, for example.

If parameter A-0-0155 is set to 0, the CLC uses parameter A-0-0153 CLC Cam Adjust Average Velocity, Phase Adjust Velocity, to set the average velocity of the phase offset move. Note that the peak velocity will be up to twice as large as this value.

A-0-0157 CLC Cam Current Master Adjust

Units: degrees
Attributes: Floating point, Read-only

This parameter shows the current cyclic phase offset command sent from the CLC to the drive.

A-0-0159 Ratio Adjust Step Rate

Units: percent per second
Default: 0.0 (instantaneous change)
Limits: 0.0 to 10000.0
Attributes: Floating point, read/write at any time

This parameter sets the rate used when the ELS Programmed Ratio Adjust, A-0-0150 Programmed Ratio Adjust, is changed.

If the step rate is set to 0, the ratio adjust will be changed immediately without a ramp. If ratio adjust is not included in the cyclic data for the drive, the value is not ramped. See A-0-0004 Axis Options and A-0-0180 Optional Command ID #1 for data configuration.

Example: The current ratio adjust is 0%, the programmed adjust is 10%, and the step rate parameter is set to 10 %/sec. The ratio adjust will be ramped for one second until it reaches the target value.
A-0-0160 Commanded Ratio Adjust

Units: percent
Attributes: Floating point, read only

This is a status of the currently commanded value of ratio adjust, which is updated either gradually or immediately to the Programmed Ratio Adjust value, A-0-0150 Programmed Ratio Adjust.

A-0-0161 CLC Cam Programmed Slave Adjust

Attributes: Read/write, floating point

This is the target value for the slave phase adjust (Sph in cam equation), which is set using the CAM/ADJUST instruction.

A-0-0162 CLC Cam Current Slave Adjust (Sph)

Attributes: Read only, floating point

This is the currently commanded value of the slave phase adjust (Sph in cam equation).

A-0-0163 CLC Cam Output Position

Attributes: Read only, floating point

This is the slave position, which is the output of the cam equation for this axis. It provides a status of the cam position even when the axis is not synchronized to the cam. A initial move or phase offset can be performed with the value read from this parameter before switching into cam synchronization mode.
A-0-0164 ELS Options

Format: Binary Word
Storage: Non-volatile, Write in Parameter Mode only or at program activation
MotionTypes: ELS and cam
Note: Bit 1 is the rightmost bit.

This parameter sets several options for ELS and cam motion. It is present only in CLC versions GPx-02.10 and greater. For dynamic synchronization and ramp up - lock on bits 1 and 6 should be set to 1.

<table>
<thead>
<tr>
<th>Bit #</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Use drive internal phase offset</td>
</tr>
<tr>
<td>2</td>
<td>Remove phase offset from cyclic data</td>
</tr>
<tr>
<td>3</td>
<td>Remove ‘mfg class 3 diag’ from cyclic data</td>
</tr>
<tr>
<td>4</td>
<td>Remove ‘target position’ from cyclic data</td>
</tr>
<tr>
<td>5</td>
<td>Use service channel for cam control/status</td>
</tr>
<tr>
<td>6</td>
<td>Do not automatically set phase offset when switching into synchronization</td>
</tr>
</tbody>
</table>

**Bit 1: Use drive internal phase offset**

(0) = Phase offset profile is generated by CLC
(1) = Phase offset profile is generated by the drive.
Default: (0)

On Indramat DIAx03 drives, the phase offset for ELS and drive-based cams can be performed on the drive.
Enabling the phase offset on the drive frees CLC resources.

**Notes:**

1. Parameter P-0-0060 is a time constant for this filter. Note that the CLC filter and other related parameters and bits are disabled with this option.

2. Enabling the phase offset filter on the drive automatically places parameter S-0-0182 into the cyclic data so that the ‘phase adjusted’ bit in the axis status register can be updated with bit 8 of this parameter.

3. When the phase offset is performed on the drive, the ‘phase adjusted’ status in the axis status register depends on parameter S-0-0228 (position synchronization window).

4. The current drive version allows only shortest path phase offset positioning. This will be changed in later versions of the drive.
Bit 2: Remove phase offset from cyclic data
(0) = Phase offset is in cyclic data
(1) = Phase offset is through service channel
Default: (0)

If the phase offset is never changed while the user program is running, cycle time in the SERCOS ring can be conserved by eliminating it from the cyclic data. When the phase offset is sent through the service channel, the user program command will take up to 50ms to execute. If the CLC is generating the phase offset profile, the value will change instantly.

Bit 5: Use service channel for cam control/status
(0) = Cam control and status uses drive real time bit
(1) = Cam control and status is through service channel
Default: (0)

In most applications, this bit should be set to (0). The cam can then be changed at the following SERCOS cycle through the drive real time bits. The drive has two real time bits that are used for cams and probe functions. If the cam does not need to be changed on the fly and more than one probe is needed for registration, this parameter bit can be set to (1).

Bit 6: Do not automatically set phase offset
(0) = Set phase offset automatically at ELS synchronization
(1) = Phase offset can be initialized with programmed offset (A-0-0151 Programmed Phase Offset) before synchronization.
Default: (0)

If this bit is set to (1), the phase offset can be initialized to any value before the drive is switched into ELS mode. If the bit is set to (0), the CLC automatically establishes relative phase synchronization, as it did in versions before GPS-3.0.
3.6.4. Axis Feedback Capture (Registration)

A-0-0170 Feedback Capture Status

Attributes: Binary Word, read only

This parameter shows the status of feedback capture set up by program activation or execution. Refer to the Feedback Capture Events in the Events section of the Programming Elements chapter. For a description of the SERCOS probe functions available with DDS-2 drives, refer to the DDS-2 manual.

Bits 1-4: Probes Enabled

- Bit 1: 0 = probe 1 positive edge disabled
- Bit 2: 0 = probe 1 negative edge disabled
- Bit 3: 0 = probe 2 positive edge disabled
- Bit 4: 0 = probe 2 negative edge disabled

1 = probe 1/2 positive edge enabled from capture/setup command

A-0-0171 Probe 1 Positive Captured Position

Units: Axis position (linear), or degrees (rotary)
Attributes: Floating point, Read-only, cyclic data

The parameter shows the last captured position for each edge of the probe inputs on the DDS drive. Refer to Feedback Capture Events in the Events section of the Programming Elements chapter.

A-0-0172 Probe 1 Negative Captured Position
See the description for Parameter A-0-0171 Probe 1 Positive Captured Position

A-0-0173 Probe 2 Positive Captured Position
See the description for Parameter A-0-0171 Probe 1 Positive Captured Position

A-0-0174 Probe 2 Negative Captured Position
See the description for Parameter A-0-0171 Probe 1 Positive Captured Position
3.6.5. Optional SERCOS Data

A-0-0180  Optional Command ID #1
A-0-0181  Optional Command ID #2
A-0-0182  Optional Command ID #3

Default: 0 (disabled)
Attributes: Integer, read/write in parameter mode only

The CLC automatically configures SERCOS with cyclic data according to the axis type. In addition to the default cyclic data, up to two optional drive parameters can be added for update every SERCOS cycle.

These parameters select the IDN of the drive parameter to be transmitted cyclically. Any parameter supported by the drive can be written. Parameters A-0-0190 and A-0-0191, and A-0-0192 Command Data #3 are the real-time values corresponding to these parameters.

Example: To update the torque limit in real-time, set A-0-0180 to 92 (Torque Limit). While the drive is in Phase 4, the value in Parameter A-0-0190 is sent cyclically to the drive, and can be written using the parameter transfer instruction in the user program.

A-0-0185 Optional Feedback ID #1

See A-0-0186 Optional Feedback ID #2

A-0-0186 Optional Feedback ID #2

Default: 0 (disabled)
Attributes: Integer, read/write in parameter mode only

The CLC automatically configures SERCOS with cyclic data according to the axis type. In addition to the default cyclic data, up to two optional drive parameters can be added for update every SERCOS cycle.

These parameters select the IDN of the drive parameter to be received cyclically. Any parameter supported by the drive can be read. Parameters A-0-0195 Feedback Data #1 and A-0-0196 Feedback Data #2 are the real-time values corresponding to these parameters.

Example: By default, feedback velocity is received through the service channel. To obtain the feedback velocity in real-time, set A-0-0185 to 40 (Feedback Velocity). While the drive is in Phase 4, the value in Parameter A-0-0195 is updated, and can be read using the parameter transfer instruction in the user program.
A-0-0190 Command Data #1  
A-0-0191 Command Data #2  
A-0-0192 Command Data #3  

Attributes: Depends on A-0-0180 through A-0-0182 Optional Command ID #3 respectively, read/write at any time

These real-time values correspond to the selected optional cyclic parameter, as described in parameters A-0-0180, A-0-0181 and A-0-0182.

A-0-0195 Feedback Data #1

See A-0-0196 Feedback Data #2

A-0-0196 Feedback Data #2

Attributes: Depends on respectively A-0-0185 & A-0-0186, read only

These real-time values correspond to the selected optional cyclic parameter, as described in parameters A-0-0186 Optional Feedback ID #2.

3.6.6. Axis Parameter Lists

A-0-2000 List of All Parameters

Type: Parameter List  
Attributes: Read-only

This is a multiple-step List Parameter that requests a list of all the parameter numbers in this parameter set. The 'D' subclass of the serial port protocol is used to list these parameters.

A-0-2001 List of Required Parameters

Type: Parameter List  
Attributes: Read-only

This is a multiple-step List Parameter that requests a list of all the parameters stored in nonvolatile CLC RAM that must be set for proper CLC operation. The 'D' subclass of the serial port protocol is used to list these parameters.
3.7. Drive Parameters

Class D, Sets 1-8

The CLC SERCOS master conforms to SERCOS Version V 01.02, as does the Indramat DDS and RAC drives. Any drive that conforms to this specification should run in the CLC’s Coordinated motion and Velocity modes. The CLC single-axis mode uses the internal intelligence of the Indramat DDS 2.1 drive and is product-specific. Consequently, DIAx01 drives will not run in single-axis mode. DIAx01 drives include: RAC and KDA drives.

A SERCOS-compatible drive contains many parameters needed to configure a motion system. Several of these parameters are used only by the controller to configure the communications, timing, and scaling. The CLC automatically sets the SERCOS timing and scaling parameters when it switches into Communication Phase 2.

Most servo and position loop parameters are contained on the SERCOS-compatible digital drive. These parameters include: Feed Constant, Kv Factor, In-Position Window, Monitor Window and all homing parameters. Acceleration, Deceleration and Jerk are defined in the user program and limits for these are set in the drive. A list of most often changed drive parameters is included in this manual. For a completed description of the parameters, refer to the DDS-2.1 drive manual. The SERCOS and DDS-2.1 manuals name parameters according to the following format:

|     |     | Parameter number
|____ |____ | Parameter set 1-8
|___  |___  | Parameter type

The CLC parameters are classified according to type or location. Translating the above parameter, as it is referred to in the SERCOS manual, to a CLC parameter:

SERCOS user parameter base 32768
SERCOS parameter set 1 * 4096 4096
SERCOS parameter number 100
36964

CLC parameter D 1.36964

The communication protocol allows the SERCOS "P" to be specified, but some decoding may have to be performed to translate between the two formats if there is more than one parameter set.
3.7.1. Drive Status

The most useful SERCOS status parameter is the drive diagnostic message, SERCOS Parameter 95 (S-0-0095). When a drive error occurs, the DDS drive sends a descriptive text message of up to 80 characters showing the status or error condition for the drive. Other drive status parameters are better accessed as axis parameters or should be used only when more specific diagnostics are needed. A complete list of the DDS drive diagnostic messages is included in the DDS 2.1/3.1 or DIAx03 manuals.

3.7.2. Required DDS Setup

Indramat’s Visual Motion program provides a utility (select Drives under the setup menu) for setting up many of the parameters needed for system start-up and operation. Many of these parameters are essential for drive operation. The number of necessary parameters depends on the application. If homing is used, many more parameters need to be entered. See Appendix D. Drive Parameter Editor.

SERCOS drives can provide lists of which parameters are required in each SERCOS phase, and which ones are invalid. A command is also included in the DDS to load default tuning values based on the connected motor and feedback method.

Default settings for new parameters for DIAx03 drives with ELS-04Vrs firmware:
The drive will not run at all or will issue errors if the following parameters aren’t set to the correct default values below. The drive does not set valid defaults at initial start-up. These parameters should be set in addition to the ones needed for ELS-03.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Set to</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>P-0-0109 Torque/force peak limitation</td>
<td>S-0-0092</td>
<td>The drive sets this parameter to 0% by default, and the motor will not move at all.</td>
</tr>
<tr>
<td>S-0-0138 Bipolar acceleration limit</td>
<td>A-0-0021</td>
<td>This is the rate used for halting the drive. Its default value of 0 means that the drive will never stop.</td>
</tr>
<tr>
<td>P-0-0127 Overload warning</td>
<td>100%</td>
<td>This is initially set to 0 by the drive, and the warning E2-61 is continuously issued.</td>
</tr>
<tr>
<td>S-0-0393 Modulo Format</td>
<td>0</td>
<td>If this is not initially set to 0 certain CAM functions will not work properly.</td>
</tr>
</tbody>
</table>
The following is a list of DDS parameters that typically need to be changed when configuring a CLC system. The list also includes ELS-04VRS parameters that have new identification numbers for compliance with the latest SERCOS specification. User interfaces and user programs should be changed if necessary to handle the new parameter numbers.

<table>
<thead>
<tr>
<th>Parameter (ELS-03VRS)</th>
<th>ELS-04VRS</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S-0-0049</td>
<td>D 1.00049</td>
<td>Positive Position Limit Value</td>
</tr>
<tr>
<td>S-0-0050</td>
<td>D 1.00050</td>
<td>Negative Position Limit Value</td>
</tr>
<tr>
<td>S-0-0055</td>
<td>D 1.00055</td>
<td>Position Polarity Parameter</td>
</tr>
<tr>
<td>S-0-0057</td>
<td>D 1.00057</td>
<td>Position Window</td>
</tr>
<tr>
<td>S-0-0091</td>
<td>D 1.00091</td>
<td>Bipolar Velocity Limit Value</td>
</tr>
<tr>
<td>S-0-0092</td>
<td>D 1.00092</td>
<td>Bipolar Torque Limit Value</td>
</tr>
<tr>
<td>S-0-0100</td>
<td>D 1.00100</td>
<td>Velocity Loop Proportional Gain</td>
</tr>
<tr>
<td>S-0-0101</td>
<td>D 1.00101</td>
<td>Velocity Loop Integral Action Time</td>
</tr>
<tr>
<td>S-0-0104</td>
<td>D 1.00104</td>
<td>Position Loop KV-Factor (closed-loop control)</td>
</tr>
<tr>
<td>S-0-0121</td>
<td>D 1.00121</td>
<td>Input Revolutions of Load Gear</td>
</tr>
<tr>
<td>S-0-0122</td>
<td>D 1.00122</td>
<td>Output Revolutions of Load Gear</td>
</tr>
<tr>
<td>S-0-0123</td>
<td>D 1.00123</td>
<td>Feed Constant</td>
</tr>
<tr>
<td>S-0-0124</td>
<td>D 1.00124</td>
<td>Zero Velocity Window</td>
</tr>
<tr>
<td>S-0-0138</td>
<td>D 1.00138</td>
<td>Bipolar Acceleration</td>
</tr>
<tr>
<td>S-0-0159</td>
<td>D 1.00159</td>
<td>Monitoring Window</td>
</tr>
<tr>
<td>P-0-0004</td>
<td>D 1.32772</td>
<td>Smoothing Time Constant</td>
</tr>
<tr>
<td>P-0-0005</td>
<td>D 1.32773</td>
<td>S-0-0265</td>
</tr>
<tr>
<td>P-0-0019</td>
<td>D 1.32787</td>
<td>Starting Position Value</td>
</tr>
<tr>
<td>P-0-0020</td>
<td>D 1.32788</td>
<td>S-0-0298</td>
</tr>
<tr>
<td>P-0-0050</td>
<td>D 1.32818</td>
<td>S-0-0348</td>
</tr>
<tr>
<td>P-0-0120</td>
<td>D 1.32888</td>
<td>S-0-0391</td>
</tr>
</tbody>
</table>
CHAPTER 4. VISUALMOTION MENU COMMANDS

4.1. Introduction

This chapter is a reference for VisualMotion’s menu commands:

<table>
<thead>
<tr>
<th>4.2. The File Menu</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.3. The Edit Menu</td>
</tr>
<tr>
<td>4.4. The View Menu</td>
</tr>
<tr>
<td>4.5. The Setup Menu</td>
</tr>
<tr>
<td>4.6. The Tools Menu</td>
</tr>
<tr>
<td>4.7. The Data Menu</td>
</tr>
<tr>
<td>4.8. The Status Menu</td>
</tr>
<tr>
<td>4.9. The Options Menu</td>
</tr>
<tr>
<td>4.10. Help</td>
</tr>
</tbody>
</table>

Some of the frequently used menu commands also have short cut buttons represented on the screen. Some of the commands described in this chapter are discussed in further detail in *Chapter 5 - Programming Concepts*.

4.2. The File Menu

Standard Windows file commands, compiling CLC program, archiving CLC program and variables, and printing CLC programs.

**New** - Opens a new icon or textual language program.
**Open** - Opens a existing icon or textual language program.
**Save** - Saves a named icon program to a file.
**Save As** - Saves a icon program to a file, filename is prompted for.
**Compile** - Checks and converts icon or textual language program to code CLC can interpret.
**Display Code** - Displays the CLC "Base Code", intermediate code generated by the first pass compiler and used by second pass compiler. This is only for reference to errors generated by the second pass compiler.
4.2.1. Program Management

The CLC Program Management dialog box provides the facilities for downloading compiled programs to the CLC card, uploading executable programs for archiving on the Host systems hard drive, activating and deleting CLC resident programs, and transferring data sets between the active program and another resident program.

![Program Management - Card 0](image)

To download a program, simply click the **Download** button and select the desired file using the in the dialog box. Executable programs are stored with the ".exe" file extension together with other project files in the Project Directory, which is defined using the Setup/Configuration command. Once the file is selected, another dialog box is displayed for specifying a program handle (number). You may choose any handle (1 - 10) not presently used in the CLC card. The default handle is the next unused handle. After clicking OK, the download is executed; a bar graph displays the progress of the download.

A program cannot be run until it is activated. To activate a program, highlight it in the file list and click the **Activate** button. The active program can only be changed when the CLC is halted. The CLC is halted when in parameter mode, an error condition exist, no program is active, or after power-up before any cycle start.

To delete a program, highlight it in the file list, click the **Delete** button, and press <ENTER> or click the Yes button in the confirmation window. If a program is currently active, it cannot be deleted from the CLC card.
Selecting **Clear All** erases all programs and data from the CLC cards memory. *See 4.2.2. Archive* before clearing. This process can be used to unfragment the CLC cards memory.

To upload a program from the CLC card to the host, highlight the file in the file list, click the **Upload** button and select the file destination and filename. A bar graph displays the progress of the upload.

Clicking on the **Data Transfer** button opens the Data Transfer dialog box. Clicking on OK copies the absolute and relative points, the events, and the integer and floating point variables from the CLC's active program to the target program selected from the CLC Program Management resident program list.

Using Data Transfer overwrites the destination program's data sets and the values are lost. If the source program's "Data Size" allocation (i.e., the size of the data sets) is larger than the target's, only the data elements within the "Data Size" allocated by the target are transferred. For example, if the active program has 75 events and the selected target program has a Data Size allocation specifying 50 events, only the first 50 events of the active program will be transferred. Data transfer is a useful tool for developing programs by incrementally testing and modifying sequential copies of a working program without the need to continuously re-input data sets for the new program.
4.2.2. Archive

Use of the Archive System dialog box is similar to the Transfer Point Tables dialog box. This menu item provides backup and restore functions for a CLC card. Backup programs, I/O Mapper strings, and parameters of the CLC card and active drives are saved by default to the \CLC\SAVESET directory. The **Browse** button allows the user to select another directory location. Programs, I/O Mapper equations, CLC card parameters, and drive parameters can also be selectively restored.

![Archive System Dialog Box](image)

Select **Backup** or **Restore**, then press the **Start** button to transfer. If restoring, the check boxes are enabled to allow selection of kind of data to be restored. During data transfer, a bar graph will advise you of progress.

Only parameter sets of active drives (those defined in the active program) will be backup. VisualMotion does not have knowledge of other drives. Backup files are named as follows:

- **Drives** "drivexx.prm" where xx is drive number
- **System** "system.prm"
- **I/O Mapper** "mapper.iom"
- **Programs** "mmmm.exn" where mmm is program name, n is program handle
- **Cams** "cam.csv"
4.2.3. Transfer Cams

This menu it is used to upload a cam table to a file for archiving, download a cam table file to the CLC card, or delete a cam table on the card. Cam table files are stored with the ".csv" file extension together with other project files in the Project Directory defined using the Setup/Configuration command. The "csv" format is a standard used by EXCEL and other spreadsheets. Each line of the file consist of two ASCII floating numbers separated by a comma, the line is finished with a "<CRLF>".

![CAM Transfer Dialog Box]

The number of lines in a cam file to be downloaded can be from 10 to 1024. At the end of the download, the CLC expands the table using a 5th order polynomial to 1024 lines. When a table is uploaded from the CLC, the expanded table( 1024 lines ) as save to a file.

There are 8 cams tables possible on CLC cards, these tables are dynamically allocated to conserve memory when not used. There can be two cam tables on each drive. When a table is present a "*" follows the number in the list box.

Cam tables may be downloaded or deleted at any time as long as the cam table number is not active. If the cam is already active, the CLC responds with a communication error “Cam is already active for axis ‘x’”. To download a new table, either switch into parameter mode or deactivate the cam for all axes using it. A cam is active when it is assigned to an axis by the Cam icon.

Select the operation mode and cam number, press the OK button. Another dialog box will prompt for the filename on non-delete operations. During cam table transfer, a bar graph will advise you of progress.
4.2.4. Transfer Events

This menu item uploads and downloads event data between a program on the CLC card and a file. By default, event filenames have a ".evt" extension and are saved to the \clc\events directory. Because event data is part of the program, it is also saved when the program is saved.

Select the operation type. The “To” or “From” Program selection (depending on the operation type) uses a combo box which contains a list of programs on the CLC card plus "Currently active". Press the Start button and then another dialog box will prompt the user for a filename. During the transfer, a bar graph will advise you of progress.

4.2.5. Transfer I/O Mapper

Use of the Transfer I/O Mapper dialog box is similar to the Transfer Point Tables dialog box with one exception. Since the CLC uses only one set of I/O mappings, a selection for a source or target is not required. I/O Mapper string files are stored with the ".iom" file extension together with other project files in the Project Directory defined using the Setup/Configuration command.
4.2.6. Transfer Parameters

The Parameters transfer dialog box is similar to the Transfer Points dialog box with a few additional capabilities. The dialog box adds radio buttons for selecting the type of parameter set to be transferred. All CLC parameters may be transferred at one time, or parameter sets for the CLC card, Tasks or Axes may be individually transferred. In addition, the parameter set for a selected DDS-2 drive connected to the CLC may be transferred through the SERCOS communication system.

![Parameters Dialog]

Parameters may be uploaded in one of two formats. Uploading for archiving saves the file with the ".prm" file extension in the :\CLC\SAVESET directory, with the data in the proper format for downloading to the CLC. Uploading for viewing saves the file to the same subdirectory as a text file with a ".txt" extension and may be viewed using Notepad or another ASCII text editor or file viewer. A *.txt parameter set uploaded for viewing cannot be downloaded to the CLC.
4.2.7. Transfer Points

The Transfer Point Tables dialog box is used to transfer the absolute and relative point tables between a resident CLC program and a point file on the Host system hard drive. Both the absolute and relative point tables are transferred at the same time.

Selecting Upload Tables and a CLC program opens a Save As dialog box permitting entry of a filename. Point files are stored with the “.pnt” file extension together with other project files in the Project Directory defined using the Setup/Configuration command.

Selecting Download Tables and a CLC program opens a File Open dialog box with :\CLC\*.PNT selected by default. Clicking on OK downloads the selected file to the specified CLC program’s absolute and relative point tables. A user may specify a different sub-directory and file extension as long as the file contents are in a proper format.

Attempting to download a data set larger than that specified by the Size in CLC resident program will cause an error. Acknowledging the error terminates the operation with the remaining points not transferred.

4.2.8. Transfer Variables

This menu item uploads and downloads program variable data between a program on the CLC card and a file. All float( Fx ) and integer( Ix ) program variables are saved together in one file. By default, variables filenames have a "var" extension and are saved to the \clc\var directory. Because variable data is part of the program, it is also saved when the program is saved.

The “To” or “From” Program selection is through a combo box containing a list of programs on the CLC card plus the "Currently active" one. Select the operation type, press the Start button. Another dialog box will prompt for the filename. During transfer, a bar graph will advise you of progress.
4.2.9. Transfer Zones

Use of the Transfer Zones dialog box is similar to the Transfer Point Tables dialog box. System Zone files are stored with the "zon" file extension together with other project files in the Project Directory defined using the Setup/Configuration command.

4.2.10. Print

Choosing "Print ..." from the File menu displays the Documentation Selection dialog box. All program data is selected by default. Checking Icon Data Flow provides a graphic printout of a VisualMotion Icon Language window. To print a graphic window you must have a graphics-capable printer. Text-based files, such as Text Language user programs or parameter files uploaded for viewing, may also be loaded and printed from Window’s Notepad or another editing program.

Setup Information can be printed for “All” or a selected task, subroutine, and event. Selecting User Labels, Register Labels, and/or Bit Labels prints a list of the respective labels for the task, subroutine, or event window currently displayed by VisualMotion.

Fields for Project Name and Programmer permit adding identification to printed pages. A header on each page contains: filename, task name, date, time, and page number. Date and time are relative to the time of printout and is based on time kept in the PC. The header can be customized by filling in the edit fields of project name and programmer.
4.3. The Edit Menu

The Edit menu contains Windows editing features for icon based programs, add event functions, and editing of user, register, and bit labels.

- **Undo** - undoes last icon edit.
- **Cut** - cuts selected program flow to paste buffer.
- **Copy** - copy selected program flow to paste buffer.
- **Paste** - enables paste operation into selected program space.
- **Delete** - deletes selected program flow from program space.
- **Select All** - selects entire program flow of current screen.

4.3.1. Clear Current Task

This deletes all contents of the current VisualMotion task, subroutine, or event workspace.

4.3.2. Find, Find Next

This menu item is used to locate labels or subroutine/event functions in the opened program.

**To find the first occurrence of a label or subroutine:**
Click label or subroutine in type grouping.
Enter the name in the Find What box. The Browse button will provide a listing of all labels or subroutines found in the opened program, one may be selected from the list.
Click the Match Case checkbox to limit search to exact match.

![Find Dialog Box](image)

**To find next occurrence of a label or subroutine:**
Select Find Next in the Edit menu or press the F3 key.
4.3.3. Add Subroutine

Selecting Add Subroutine from the Edit menu opens a Subroutine Control Block dialog box and allows the user to write a subroutine that may be used in conjunction with a Sequencer.

Enter the name of the desired subroutine in the edit field, name must begin with an alpha character. Pressing OK button changes to the program workspace with the name entered. This screen is not saved unless one or more icons are placed on the program workspace before changing to another. A maximum of 200 screens, consisting of tasks, subroutines, and event functions are allowed.

![Subroutine Control Block](image)

4.3.4. Add Event Function

Selecting Add Event Function from the Edit menu opens an Event Function Control Block dialog box and is the only way to write a new event function.

Event functions are similar to subroutines. However, event functions are not "called" from a program. Instead, an event function is "triggered" by the conditions (distance, time, etc.) that are specified in an event setup (refer to Calc icon in the chapter, “Icon Programming” and the description of the Event Table in the chapter, “Programming Concepts”). Note that an event function must be written before it can be assigned to an axis.

Entering a name for an event function and clicking OK replaces the current VisualMotion workspace with an event function workspace. You may then write an event function using icons and connecting lines in the same manner as when writing a task or subroutine.

4.3.5. Labels - User Labels

Assigning symbolic names to system resources, such as axes, drives, etc., is a very useful programming technique. Labels may also be used for absolute or relative point names, or in place of "literal" constant or variable values in expressions. For example, once assigned, the label "PI" can be used throughout a program, instead of repeatedly entering the literal value 3.14159.
Symbolic names, or Labels, may use up to twenty ASCII characters and are case-sensitive. Blank spaces are not allowed within a symbol; use a printable character as a separator if it is required for clarity. For example, "next_move", rather than "next move".

The CLC compiler, used for both Icon and Text Language programming, allows the use of a literal integer value (i.e., a number such as "1" or "5") providing it is within the range of integers that are valid for the specified argument. Integers used to specify system devices, such as an axis or drive, must be within the range permitted by the complete CLC system and software installed. For example: a CLC system with eight DDS-2 drives installed can specify an axis or drive using an integer from 1 to 8. The compiler must be able to resolve a symbol used as a table index argument to an integer index within the range, or size, of the table.

User Defined Labels Dialog Box
The User Defined Labels dialog box allows you to assign an ASCII name to a value or system component, as previously described. This dialog box can be accessed from most VisualMotion dialog box data entry fields. Simply click once in the data entry field, then double-click anywhere in a non-active area of the dialog box. A User Defined Labels dialog box appears in an adjacent area of the screen. Thereafter, instead of explicitly entering a value or redefining a system component, the label can be entered by accessing the User Defined Labels dialog box and selecting the appropriate label. An optional eighty character comment field is also available for each user defined label.
4.3.6. Labels - Register Labels

Selecting the Edit/Labels/Register Labels displays the Register Labels dialog box, used to provide symbolic ASCII names for CLC control status and I/O registers. The first 87 registers are reserved for CLC system use and are assigned names for system, task, DDS-2 drive, and axis use. These registers may not be modified. Registers from register 88 and above may be assigned names. If drivers for a CLC supported I/O system are installed, the associated I/O register names will be listed, typically beginning at register 100. The register list may be sorted alphabetically by name by checking the Sort by Label checkbox.

Clicking on Add or Edit (with a register selected) opens an Add/Edit Register Labels dialog box for adding or modifying a specified register name. Here the Label Name must be explicitly entered; the pop-up User Defined Labels dialog box is not available. Once modified, register labels are embedded in the motion program (the .str file) and will not be lost if the program is later transferred to a different CLC system.

New programs are loaded with the default register names. Refer to the CLC Input/Output Systems chapter for the default register names.

![Register Labels dialog box](image)
4.3.7. Labels - Bit Labels

Selecting the Edit/Labels/Bit Labels displays the Bit Labels dialog box, used to provide symbolic ASCII names for individual bits within CLC control and I/O registers. The bit label list may be sorted alphabetically by name by clicking on the Sort by Label checkbox.

The bit labels for the first 39 registers are may not be modified. Clicking on Add or Edit (with a register-bit selected) opens an Add/Edit Bit Labels dialog box for adding or modifying a specified bit label. Here the Label Name must be explicitly entered; the pop-up User Defined Labels dialog box is not applicable. Once modified, bit labels are embedded in the motion program (the .str file) and will not be lost if the program is later transferred to a different CLC system.

![Bit Labels dialog box]

New programs are loaded with the default register names. See Chapter 2. CLC Input/Output Systems for the default bit names.

4.3.8. Import User Label File

Choosing Import User Label File from the Edit/Labels menu opens an Import Label File dialog box which permits you to import a previously saved label file into the current program. Importing a label file overwrites all existing labels in the program. User defined label files are stored by default in the CLC directory with a .lbl file extension.

4.3.9. Export User Label File

Choosing Export User Label File from the Edit/Labels menu opens a file Save As dialog box. Saved label files may be subsequently imported into another program allowing you to maintain standardized user labels for various categories of programs. User defined label files are saved by default in the clc directory with a .lbl file extension.
4.4. The View Menu

The View menu allows you to select a task, subroutine, or event function for display or editing. Choosing one of the menu selections loads the selected portion of the program into the VisualMotion workspace, replacing the current workspace contents.

4.4.1. Subroutines

The workspace may be loaded with a subroutine by using the View menu’s Subroutines command. The Subroutines dialog box lists the programmed subroutines by name and displays the total number of subroutines in its title bar. Selecting an existing subroutine from the displayed list permits deleting the subroutine or replacing the current VisualMotion workspace with the subroutine for viewing/editing.

A subroutine also may be loaded to the workspace by double-clicking an existing subroutine icon to obtain the icon Subroutines dialog box. The subroutine icon's name is displayed as the default. Clicking on enter function loads the workspace with the subroutine.

4.4.2. Event Functions

Choosing the "Event Functions..." from the View menu displays the Event Functions selection dialog box and is used in the same manner as the View menu's Subroutines command. The View menu Event Functions is the only way to access an event function for editing or deletion. The event function must have been previously created using the Edit menu's "Add Event Function."

4.4.3. Zoom Out

The Zoom Out menu item is used to load the VisualMotion workspace with the parent of the presently loaded subroutine. Zoom Out, or the <F6> key, can be used to move from the back to the front of a nested subroutine queue one subroutine at a time. The queue of nested subroutines is displayed in the title bar of the VisualMotion workspace.

The Zoom Out command is unavailable when any of the four main tasks are loaded in the workspace or when the presently loaded subroutine was opened via the Subroutines command instead of the Enter Subroutine button of a subroutine icon. Since events functions are independent functions invoked by a specified axis motion or time-based event, they have no "parent". Therefore, Zoom Out cannot be used to return to a higher level.
4.5. The Setup Menu

4.5.1. Card Selection

*Refer to the CLC Start Up Guide for instructions on how to use the Card Selection Setup dialog box to configure Host-to-CLC communication for your system.*

After saving and compiling an icon program file, the user can now setup variables, events, point tables and zones without being connected to a CLC. To setup the off-line programming, choose “Card Selection” from the “Setup” menu. Select “CLC File” from the connection method drop down box. This tells VisualMotion® to get setup information from a file on your computer instead of the serial port, AT bus, etc. You must then select a file to use. Click on the “File” button and select the .exc file that you just compiled. Now, the programmer can enter values for variables and point tables and configure events that will be downloaded to the CLC with the .exc file.
4.5.2. Configuration

This menu item allows the user to personalize Visual Motion.

Windows Editor - Select the Windows compatible text editor to be used for non-graphical displays. The default is Notepad.exe, Write.exe is also supplied with Windows.

Project Directory - Select the directory for file storage. The default is \clc\project. The Browse button simplifies the selection of the project directory.

Generate Map File - When icon based programs are compiled a map file can be generated to be used to show program flow. On larger programs or older PCs, the generation may take several minutes. The default is on.

Compiler Output File Version - The second pass compiler can produce a file for earlier versions of the CLC card (pre GPS 2.0). Refer to 4.8.4. System under the Status menu to see if the GPS version number is less then 2. If so, choose Pre GPS 2.0. Otherwise choose Other. Features such as cams, PLS, Sequencer, local variables are not supported in the earlier version.

Language Selection - On this release only English is supported

![Visual Motion Configuration]

4.5.3. Drives

Refer to Appendix D. CLC Drive Parameter Editor.
4.5.4. Coordinated Motion

Jogging
Coordinated Jogging Percents set the increments and velocities used for fast and slow jogging. The Increment data area is used to set the Large and Small percentage of the maximum distance for a single-step jog operation. The maximum is defined by the axis parameter Maximum Jog Increment (A-0-0025). Similarly, the Speed data area is used to set the Fast and Slow jog speeds as a percentage of the maximum velocity, which is defined by the axis parameter Maximum Jog Velocity (A-0-0026). These values are stored in the following parameters: Large Increment - C-0-0052, Small Increment - C-0-0053, Fast Speed - C-0-0055, Slow Speed - C-0-0056.

Task Limits
The Task Maximum Path Limits dialog box allows speed, acceleration, and deceleration limits for coordinated motion to be individually set for each task. This dialog box permits setting acceleration and deceleration independently. The DDS-2’s internal function uses the same accel/decel value for single axis non-coordinated motion.

Clicking Save downloads the changed values to the CLC card, without requiring the CLC to be in Parameter mode.
4.5.5. I/O Setup

The I/O Setup dialog box allows viewing and editing of the I/O system configuration for the CLC. There are three general classes of I/O, DDS resident DEA cards, SERCOS based I/O, and VME cards. A CLC system can have one or the other classes, all, or none if controlled serially or from the VME bus by another VME device (PC, PLC). Selecting the I/O type installs the specific device drivers for that I/O. An error will result if a I/O card is configured but is not present.

NOTE: I/O Setup may only be altered in Parameter Mode. Refer to System Control Register in the chapter “CLC input/output Systems.”

Each DDS can hold up to three DEA cards, each DEA card has 15 inputs and 16 outputs. DEA cards are hardware configured to be card 1, 2, or 3. Their order in the drive rack does not matter.

To configure a DEA card in a DDS, select the drive and press the Configure DDS I/O… button. A dialog box will appear with three combo boxes. DEA cards in drives 1-8 have predefined base registers, for cards in drives 9-99 a base register must be entered. When I/O setup is complete, press the SAVE button to send all setup information to the CLC.

SERCOS based I/O (Lutze, etc.) is automatically configured by the CLC when they are seen on the ring.

VME I/O cards supported are: Pentland MPV922, Xycom XVM-201, Xycom XVM-202, Xycom XVM-244. Select the external I/O type from the combo box. Xycom I/O cards require additional setup, press Configure VME I/O… button to select options. When I/O setup is complete, press the SAVE button to send all setup information to the CLC. The Overview… button brings up a list of currently configured I/O. This list includes the register number, drive number, module ID, I/O type and the number of registers.
4.5.6. Overview

Selecting Overview from the Setup menu opens a View CLC card Parameters dialog box; the initial list display is blank. This dialog box may be used to view and setup all CLC and DDS-2 user accessible parameters.

Parameters may be uploaded for display or editing by selecting one of the “Param Source” radio buttons, then clicking the OK button. Choosing CLC card allows uploading the CLC card system parameters. Choosing Drives or Axes requires the entry of a Drive or Axes Number in the enabled data entry box. Selecting Task requires choosing one of the four CLC tasks from the enabled pull-down menu.

After the selected list has been loaded, you may scroll the list by clicking on the list scroll bar’s up and down arrowheads, or clicking and dragging the scroll button. Clicking and holding (or dragging) the scroll bar button will display the number of the first viewable parameter in the display list in the upper left of the title bar.

Selecting a list entry and clicking the Edit button opens a parameter editing dialog box that allows you to change the parameter value and Update the parameter by downloading it to the CLC or DDS-2 drive.

In addition to uploading all the parameters of a set (i.e., all drive or task parameters, etc.) you may build a custom set of parameters by clicking on the Build Custom… button and opening a Build Custom Display dialog box.

When the Build Custom Display box opens, the parent View … Parameters dialog box remains active. Selecting a list item from the parent box’s list, then clicking Add in the build box adds the selected parameter to the custom parameter list. Only one parameter may be added at one time. Selecting one of the parameters in the custom list, then clicking Delete, removes the item from the custom list.
Since the parent dialog box remains active, the parameter list in the parent dialog box may be changed by re-selecting a Param Source and type. Parameters from the newly uploaded list may then be added to the custom list in the build dialog box. In this manner a custom list may be built by selecting any available parameter from any parameter list.

A previously saved custom list may be loaded and used as a source for building another custom list, however, it must be the first list loaded. You cannot load a custom display list once another list has been loaded without closing the Build Custom Display dialog box.

Clicking on Save pops-up a Name Custom Display dialog box permitting entry of an eight character maximum filename. Custom parameter lists are stored in the CLC.INI file in the CLC sub-directory.

Clicking on the Display Custom button opens a Display Custom List dialog box with a list of the currently saved custom parameter lists by name. Selecting a list and clicking on OK or double clicking the list item uploads and displays the parameters in the selected list.

**4.5.7. Pendant Security**

The Teach Pendant Security menu item selects the level of user accessibility for variables and registers. *See the CLC Start Up Guide for more information on the Teach Pendant.*
4.5.8. CLC Serial Ports

The “Comms Setup” buttons permit setup of the CLC’s two serial ports. Port 1 is typically used for communication with the Host system and defaults to 9600 baud. Port 2 is used to communicate with a teach pendant (default 9600 baud), if one is installed. If an ASCII "dumb" terminal is used to communicate with a CLC, the checksum should be disabled. Before saving any serial port modifications, VisualMotion requests the user to confirm the changes via a dialog box with the message, “Caution! Changing port configuration may stop communications. Continue?”

Clicking Save downloads the changed values to the CLC card.

4.5.9. VME Configure

Selecting VME Configure from the Setup menu opens the VME Configuration dialog box. CLC VME configuration beyond the standard default configuration is only required if one of the following conditions is valid for the VME system:

- There will be more than one CLC VME motion control card in the VME card cage.
- There will be more than one VME master (e.g., a CLC motion control with a VME PLC).
- There are VME I/O cards that must occupy the default CLC memory space.
- The VME system designer requires use of the CLC default memory space.

Refer to the CLC Start Up Guide Hardware Appendix for more detailed CLC-V configuration information.

![VME Configuration Dialog Box](image)
4.6. The Tools Menu

4.6.1. Breakpoint Control

When enabled, this utility stops program flow, of a selected task, at the start of a selected functions. It is intended to be a debugging tool. Toggling the Cycle_Start_Resume(bit 6) of the task control register(2-5) of the selected task resumes program flow.

The Window is divided into four vertical grouping, one for each task. Each grouping contains a list of all functions of the active program, an enable checkbox, and an indicator of when the breakpoint was hit.

To stop at a function:

- Select the function from the list-box of the task to be stopped. The list-box contains a listing of all functions, some functions are never ran from a given task (i.e. Task B is never run from Task A).
- Arm it by clicking on the enable checkbox.
- Wait for dot in the indicator.

Also see Register 2-5 Task Control - Single Step, Breakpoint Enable, Step Sequence Step, Step Sequence Function.
4.6.2. CAM Builder

The CAM Building Utility is a Windows based application used to build cam tables for Indramat’s CLC motion control cards. See Appendix C. ELS Configuration for more information.

**Controls:**

*Target* - selects the format of the output data.
- CLC card - the output file will be in Degrees - Units.
- Drive - the output file will be in Degrees - Percent.
*(Units and Percent depend on the axis modulo)*

*Type* - selects the algorithm used to build output data from input data (ACAM, PCAM, SCAM or VCAM).

*Output Steps* - select the number of lines of output data to be built.

*S Shaping* (Jerk limiting) - select the amount of s shaping to be used on the velocity curve to prevent a jump in acceleration.

*Output Modulo* - for scaling the range of ACAM and VCAM output calculations. By default, this scaling is 360.

*Input Data* - list box containing input data. Double click left mouse button to edit.

*Output Data* - list box used to display output data.
Buttons:

*Graph* - creates/refreshes a graph from data in the output data listbox.
*Add* - add an entry to the input data listbox.
*Edit* - edits selected entry in input data listbox.
*Delete* - deletes selected entry in input data listbox.
*Build* - builds position, velocity, and acceleration profiles and displays position data in the output data listbox. Uses the algorithm selected in **Build Types** to build the number of data sets chosen in **Output Steps** using the **Input Data** as input and **S Shaping** to modify.

Menu:

The following description outlines the Menu Items and the overall functionality of this utility.

- **File** - getting file data, saving data to a file, printing, and exit.
  - **New** - clear input data list-box.
  - **Open Input File** - data from user selected input file is loaded into input data list-box.
  - **Open Output File** - data from user selected output is loaded into output data list-box.
  - **Save Input** - save values from input data list-box to a user selected input file
  - **Save Output** - saves values from output data list-box to a user selected output file
  - **Chart Wizard** - a series of setup screens to help setup choices on the main screen.
  - **Cam Transfer** - cam transfer function to move cam file to CLC or drive cam table.
  - **Transfer Input to Point Table** - This screen is used to transfers the contents of the input data list-box to points in a CLC program.
**Input Predefined Cam** - input data from a predefined cam file.

- Cycloidal
- Harmonic
- Constant Acceleration
- Modified Sine
- Modified Trapezoid

**Print Output** - prints graph and output data to Windows printer.
**Exit** - closes cam building API.

**Options** - graphing and printout choices.
- *Acceleration Graph* - when checked, the acceleration graph is visible on the graph and print out.
- *Position Graph* - when checked, the position graph is visible on the graph and print out.
- *Velocity Graph* - when checked, the velocity graph is visible on the graph and print out.
- *Simulation Speed for Graph* - selects the speed used in graphing the velocity and acceleration.

**Help** - accessing help system and identifying product.
- *Topic* - accesses this help system.
- *About* - identifies product and date of build.
User selected input file
Input files are assumed to have the following format.

File identifier string. ;first line only
master value     slave value ;second line
master value     slave value ;third line
                       ---     ---
master value     slave value ;last line

Minimum and maximum number of data sets:
Build Types       ACAM      PCAM SCAM VCAM
Minimum Sets       2         2        5        2
Maximum Sets       1024      1024      500      1024

User selected output file
Output files are assumed to have the following format. This is commonly referred to as CSV format and is supported by Excel. Cam output files are sent to the CLC card using Visual Motion.

master value,     slave value ;first line
master value,     slave value ;second line
                        ---     ---
master value,     slave value ;last line

Transfer Input to Point Table
This screen is used to transfers the contents of the ‘input data list-box’ to points in a CLC program. A set of input values used for the CAM Build icon or instruction can be evaluated with the BUILD and GRAPH buttons of the previous screen. When the resultant cam is acceptable, the set of input values are transferred to the point table of the selected program on the CLC card.

Programs on Card: This combo-box list the programs on the CLC card. The list is uploaded when the screen is opened. Select the program containing the point table to be written to.

Starting Point Number for Transfer: Enter the number of the beginning point of the destination. The values of the first line of the “input data list-box” will be copied to this point. The values from each additional line of the “input data list-box” will be moved to successive point locations.

The “OK” button starts the transfer. Before the transfer is started, checks are made for valid point and valid range. A dialog screen will appear asking you to confirm your action. The “Cancel” button exits this screen.
4.6.3. Jogging

This menu item spawns an API for jogging DDS2.1, DDS2.2, DKC, and DKS drives attached to a CLC card with GPS 5.0 or later. When the CLC card is in manual mode the Jog Control tool allows operators to jog an axis that is in Velocity or Single Axis mode. For Coordinated motion all jogging functions are controlled by the Axis(n)_Control and Task(A-D)_Jog Registers:

Registers 11-18, 209-240: Axis Control
Registers 7-10: Task Jog Control

See Chapter 2. I/O Systems for more information

When system parameter C-0-0010, bit 11 is set to 1 (0000010000000000), jogging can also be performed in Auto Mode or when a task is running.

The dialog box displays the current Task assignment, Mode and Position for the active axis along with status messages and extended diagnostics. Selecting either the jog reverse or jog forward buttons respectively changes the state of bit 3 and 4 in the Axis_Control Register from 0 to 1. Deselecting these buttons will change the state of these bits from 1 back to 0.

A low-to-high (0-1) transition on these bits causes motion to start in the positive (bit 2) or negative (bit 3) direction. A high-to-low (1-0) transition immediately stops the motion. Motion is also stopped when the task mode selection changes, or when a travel limit or incremental distance has been reached.
Axis Options

Selecting **Axis** from the **Options** menu will open a dialog box which allows the user to input the maximum jog distance and speed. The maximum distance value is stored in parameter \(A-0-0025\). The maximum speed is stored in parameter \(A-0-0026\). The large/small increment values and the fast/slow velocities are calculated according to the values entered in the System Jogging Options selection described below.

![Axis Jogging Options](image)

Task Options

Selecting **Task** from the Options menu will open a dialog box which allows the user to select the jog type and rate. The **jog type** can be distance (incremental) or speed (continuous). The type of jog takes effect when the next jog is started with a transition on the jog forward or reverse buttons. When **distance** is selected the jogging motion will stop after the large or small travel limit is reached. When **speed** is selected the jogging motion will continue until the jog button is deselected or the travel limit is reached. See **Appendix D. Drive Parameter Editor-Drive Limits**.

![Task Options](image)
When distance is selected, the jog rate selection is **Large** or **Small**. When speed is selected, the jog rate selection is **Fast** or **Slow**. Refer to Bit #6 in the **Task Jog Control Register**.

The **System Jogging Options** dialog box is used for setting the increments and velocities used for fast and slow jogging. The **Increment** data area is used to set the **Large** and **Small** percentage of the maximum distance for a single-step jog operation. The maximum is defined by the axis parameter Maximum Jog Increment **(A-0-0025)**. Similarly, the **Speed** data area is used to set the Fast and Slow jog speeds as a percentage of the maximum velocity, which is defined by the axis parameter Maximum Jog Velocity **(A-0-0026)**.

These values are stored in the following parameter locations:

- Large Increment - **C-0-0052**
- Small Increment - **C-0-0053**
- Fast Speed - **C-0-0055**
- Slow Speed - **C-0-0056**

**Single-Axis Mode Jogging**

**Position Limits Enabled**
When a jog forward is started, the CLC sets the target position to the positive travel limit. When a reverse jog is started, the target position is set to the negative travel limit. When the jog is stopped, the target velocity is set to zero, but the target position remains at the travel limit.

**Position Limits Not Enabled - DIAx03 Drives version 04 and greater**
On DIAx03 drives version 4 and later, the drive is switched to velocity mode. The ramps are generated internally by the drive using the axis jog acceleration parameter **A-0-0023**.

Before performing single-axis positioning using the `axis.move` command, it is necessary to execute the `els.mode` command to switch the drive back into single-axis mode. A cycle stop followed by a cycle start will also reset the drives to single-axis mode.

**Position Limits Not Enabled - Other Drives**
The drive remains in single-axis mode. The CLC will continually increase or decrease the target position by a small amount to keep the drive moving, until the jog is stopped. It sets the acceleration to a value high enough so that the drive does not decelerate. The jog acceleration parameter will be used only if it is lower than that value.

**Velocity Mode Jogging**
DIAx03 Drives version 04 and greater
The drive generates the ramps internally based on the programmed acceleration, as described above in 'position limits not enabled'.

**Other Drives**
The ramp selection in parameter **A-0-0004** bit 9 determines if the programmed acceleration is used. The CLC generates a ramp if ramping is enabled, otherwise the velocity is immediately stepped to the programmed value.

**Other Changes Since pre-GPS 5.0**
When a continuous jog is stopped on a DIAx03 drive, the drive remains in an AF state, and the CLC commands it to zero velocity using the programmed acceleration.

If DIAx03 ELS-04VRS is used, there is always a ramp on the velocity as long as its programmed acceleration is nonzero. If a step is needed, the jog acceleration should be set a large value.

Refer to the following parameter and register descriptions for more jogging information:

- C-0-0042 World Large Increment
- C-0-0043 World Small Increment
- C-0-0045 World Fast Jog Speed
- C-0-0046 World Slow Jog Speed
- C-0-0160 Virtual Master Maximum Jog Velocity
- T-0-0025 Maximum Jog Increment
- T-0-0026 Maximum Jog Velocity

**Registers 31-38:** Axis(n) Status - bit 2 jogging fwd, bit 3 jogging rev.

**4.6.4. Oscilloscope**

For information on the Oscilloscope function *See Appendix D. CLC Drive Parameter Editor.*
4.6.5. Show Program Flow

This menu item highlights the currently executing icon. Many icons operate quickly and may appear to be skipped over. Program flow for subroutines require the operator to identify the task controlling the subroutines.

Show Program Flow utilizes a map file generated at compile time to tag the screen location of an instruction. If this map file is not found, or if the icons have been moved, added or deleted since the time the program was compile and downloaded, erroneous program flow may appear.

During Show Program Flow other menu items are disabled, selecting Show Program Flow a second time enables the other menu items and removes the highlighting.

4.7. The Data Menu

4.7.1. Events

Events are used to execute an event function on time, position, angle, or I/O state conditions. Each event has: {status, type, direction, distance or time, event function, and message} fields. The View Events Table dialog box permits viewing and editing of the event table of a program that has already been downloaded to the CLC card. The dialog box automatically uploads and displays the contents of the event table for the currently active program on the CLC. See Chapter 5. Programming Concepts for more information.

To edit an event, select it in the list-box, and press the EDIT button, or double-click on the list item. In either case, another dialog box will appear with buttons and edit fields. Change one or more fields and press SAVE button to send to CLC. Editing a selected CLC resident
program's event table is one method for associating an event table entry with an event function, and specifying how the function is invoked by the program.

Events for coordinated motion are related to the beginning or end of a path segment as a percentage of the total path distance or time on the path. Events for single axis motion may be set to take place at an absolute distance from the beginning or end of the axis move. A cyclic event may be setup to be triggered by a continuously repeating timer or as an angular position on an axis using rotary motion.

Task Input Transition triggers an event on a low to high transition of a Task Control Register interrupt event bit. The I/O mapper is used to map a specified I/O register condition from the I/O register to bit 9 in the appropriate Task Control Register. The approximate latency is \( \leq 2 \) milliseconds.

VME-based CLC systems may trigger events using the VME Broadcast Event or VME Short Address (mailbox) Event. These triggers are typically used to trigger or synchronize events on other CLC cards in a multi-card CLC system.

Feedback Capture uses the Probe capability of DDS-2 drives to trigger a CLC event based on a positive or negative transition of the drive's Probe 1 or Probe 2 input.

To obtain a list of a program's CLC resident events, you can pop-up an Event Functions dialog box with a list of the events. Place the focus within the "Optional 'Event Function' to run" data entry box, then double-click on a non-active area of the Edit Event Values dialog box.

After setting up an event, download the event configuration to the CLC by clicking on the Save button. The Previous and Next buttons permit editing and downloading a series of events without having to leave the Edit Event Values dialog box. Once an event function is associated with an event ID in a program's event table, an event icon is used to "arm" or "disarm" events, or to suspend program execution until the event is done.

### 4.7.2. I/O Mapper

#### Overview
The CLC I-O mapper allows manipulation of I-O registers using Boolean strings and an optional ladder logic interface. This can be used to map physical I-O bits to the control and status registers. Simple PLC functionality is included, with AND, OR, and NOT functions. The I-O mapping program interpreter executes all equations every scan time (4ms or 8ms) in all modes, independent of the User Tasks. This allows a deterministic response to critical I-O.
Storage and Compilation
An I-O mapper string file (.IOM) can be generated from the ladder logic interface or entered using a text editor, and saved on the PC. This file is then sent to the CLC, where it is compiled to an executable format.

The mapper is stored as Boolean equations. On the CLC, comments can take a large amount of SRAM storage. The ladder interface shows the total space consumed before the file is sent to the CLC, and the user can optionally disable storing comments on the CLC.

NOTE: The I/O Mapper may only be altered in Parameter Mode. Refer to Chapter 2. CLC Input/Output Systems.

File Menu
New - Opens a new Ladder Editor - I/O Mapper File
Open - Opens an existing I/O Mapper file (*.iom)
Save - Saves a named I/O Mapper to a file.
Save as - Saves an I/O Mapper to a file, filename is prompted for.

Upload Strings - When Uploading an I/O Mapper from the CLC card, the Ladder Editor will prompt the user before overwriting any existing rungs.

Download Strings - When Downloading an I/O Mapper to the CLC card, the Ladder Editor will prompt the user before overwriting any existing rungs. The CLC card must be in parameter mode.

Print Output - Prints all or individual components of the I/O Mapper file which include rungs, comments, strings and cross references. A project and programmer name can also be sent with the print job.

Edit Menu
Add rung - Adds a rung to the end of the I/O Mapper.
Delete rung - Deletes the currently displayed rung from the I/O Mapper.
Insert rung - Adds a rung in front of the currently displayed rung in the I/O Mapper.
Copy rung - Copies the currently displayed rung to a clipboard.
Paste rung - Pastes (overwrites) the contents of the clipboard in the currently displayed rung.
Check - Checks the rung for missing contacts, coils, shorts and non-contiguous lines.
Find... - Finds the rung which contains a specific Bit within a Register.
Find Next - Finds the next occurrence of the above within the I/O Mapper.

Options Menu
Display Strings - Displays each rung as a Boolean argument represented by an ASCII text string.

Double clicking a string in the displayed list or selecting one and clicking the Edit button opens an Edit Mapper String dialog box. The Edit Mapper String dialog box allows editing of I/O Mapper strings resident on the CLC.
For proper use and limitations of I/O mapping see Chapter 2. CLC Input/Output Systems - I/O Mapper.

A semi-colon at the end of an I/O Mapper string will allow comments to be entered. After adding or editing one or more strings, the Save button may be used to download changes to the CLC; a bar graph displays the progress of the download.

Settings Menu
Card Selection - launches the same dialog box accessed under the VisualMotion Setup Menu. Forcing - Sets the forcing options for the I/O Mapper. Choose Registers under the Data menu to change or force bits.

Scan Time - The I/O Mapper equations can be executed every 4 or 8 milliseconds, and therefore must be kept to a minimum to allow sufficient time for motion tasks.

Online Menu
Status - Contains Current and Maximum values for File size, Executable size, and total I/O mapper operations.
The following **Function Keys** are used with the Ladder Editor:

- **F1** - Help
- **F4** - Find Next
- **F5** - Previous Rung
- **F6** - Next Rung
- **F7** - Show Status

### Right Mouse Button

- **Delete** - Deletes the contents of the square pointed to.
- **Edit** - Opens a dialog box to change register, bit, or contact type.
- **Copy** - Copies the contact or coil pointed to into memory.
- **Cut** - Copies the contact or coil pointed to into memory and clears it.
- **Paste** - Pastes a contact or coil from memory into current rung.
- **Undo** - Undoes the last edit operation.
- **Insert column** - Shifts contents of column pointed to, to the right. Existing lines are extended. Error if right most contact column in use.
- **Insert row** - Shifts contents of row pointed to down one row. Existing lines are extended. Error if bottom row is not empty.
- **Force term** - Opens dialog box which allows the user to force "on" or "off" the selected register bit. Clear forcing clears the selection. "Clear All Forcing" clears all within I/O Map.

![Force Control](image)

**Double click Left Mouse Button** - Edits contact or coil pointed to.
4.7.3. PID Control Loops

PID Overview  (See also PID Icon)
CLC programs, GPS 5.00 and later, can have up to 10 PID loops. The PID loops are added to the program with the PID icon or PID/INIT textual command. PID loops can be monitored and tuned using the PID Monitor.

The PID instruction is activated at program activation with SERCOS ring in phase 2 or greater and it's control register “PID Enable” (bit 5) set. The tasks do not need to be running.

The CLC can include up to 10 PID (Proportional, Integral, Derivative) control loops with each program. These PID’s are parameterized with program variables or registers, and have a minimum update rate of 8ms. A choice of optional filters (Low pass or Butterworth) may be applied to the feedback signal.

PID Instruction.
The PID instruction configures a PI or PID control loop. The set point, feedback, and output variables can be registers, integers, floats, or parameters; appropriate conversions are supplied. Control factors (Ki, Kp, Kd, Last_I_Result_Preset) and limits (min., max.) can be constants or variables. Minimum loop update time is 8 milliseconds. In operation, the PID instruction only needs to be executed once in the program flow. The label for the PID loop is it's control registers label.

---

**Diagram of PID and Limiter**

- **PID**
  - **Tc** = Tc_Scaler(Command - Tc_Bias)
  - Pf = Kp * Error
  - Df = Kd * (Tf - Tf_old)
  - If = Last_I_Result + (Ki * Error)
  - O = Max

- **Limiter**
  - Output > Max?
    - Y: Output = Max
    - N: Output = Min
  - To = To_Scaler(Output - To_Bias)
  - Output = Min
  - Tension Out

---

**VisualMotion Menu Commands**

- **PID Limiter**
  - O = Min
  - Tension Out
Block of 20 program float variables per PID loop (Fx). (Type 1 PID usage)

F20
Command scaler value, default 1.0.  4
1  Command bias value, default 0.0.  4
2  Feedback scaler value, default 1.0.  4
3  Feedback bias value, default 0.0.  4
4  Kp value, default 1.0.  4
5  Ki value, default 0.0.  4
6  Kd value, default 0.0.  4
7  Ki limit value, default 0.0.  4
8  Minimum output value, default -10.0.  4
9  Maximum output value, default 10.0.  4
F30
Preset value, default 0.0.  4
1  Output scaler value, default 1.0.  4
2  Output bias value, default 0.0.  4
3  Feedback cutoff frequency (Hz), default 0  4
4  Feedback filter type, default 0  4
    0 = None
    1 = First order low-pass
    2 = Second order low-pass
    3 = Third order Butterworth
    4 = Second order Butterworth
    5 = Third order Butterworth
    6 = Velocity tracking 2nd order
    7 = Accel ramp tracking 3rd order
5  reserved  4
6  reserved  4
7  reserved  4
8  reserved  4
9  reserved  4
**PID Monitor - Overview**

This menu item spawns a API for monitoring and tuning the PID of the active program on the CLC card. CLC programs, GPS 5.00 and later, can have up to 10 PID loops. The PID loops are added to the program with the PID icon or PID/INIT textual command.

The initial screen displays a list of PID loops of the active program, and the set point, feedback, output, loop time, and status of the selected loop. All fields, except loop time are continuous updated. PID loops are activated when a program is selected, loops of other programs are not viewable. Pressing the right mouse button displays a menu list of: About, Update, and Properties.

A Tuning Button displays a screen of parameters for editing.
The tuning screens is used to adjust the selected PID loop while monitoring its values on the main screen. This screen has several grouping for scaling and adding offset to the process variables. A grouping adjusting the Kd, Kp, Ki, integral preset, and integral limit. The output grouping also has min, max limits for it. The feedback has an optional digital filter to condition the signal. The feedback filter choices are:

No filter.
First order low-pass, 
\[ G(s) = \frac{1}{s+1} \]
Second order low-pass, 
\[ G(s) = \frac{1}{s^2 + 2s + 1} \]
Third order low-pass, 
\[ G(s) = \frac{1}{s^3 + 3s^2 + 3s + 1} \]
Second order Butterworth, 
\[ G(s) = \frac{1}{s^2 + 2^{1/2}s + 1} \]
Third order Butterworth, 
\[ G(s) = \frac{1}{s^3 + 2s^2 + 2s + 1} \]
Modified 2nd order low-pass with velocity ramp tracking, 
\[ G(s) = \frac{2s+1}{s^2 + 2s + 1} \]
Modified 3rd order low-pass with accel ramp tracking, 
\[ G(s) = \frac{3s^2 + 3s + 1}{s^3 + 3s^2 + 3s + 1} \]

When a filter is chosen the cutoff frequency for the filter must be entered. The cutoff frequency is the frequency where the signal is reduced 3db.

The Properties screen shows the variable assignment used for this loop, these are read only.

**Control Register**
Bit 4 Enable Preset - If set, loads integral preset, on program activation or when loop enabled by bit 5.
Bit 5 Enable PID - If set, enables loop, clearing it, disables it. This bit is checked every SERCOS update.

**Status Register**
Bit 4 Enable Preset - Acknowledgment of control register preset bit(4).
Bit 5 Enable PID - Acknowledgment of control register enable bit(5).

4.7.4. PLS

Overview
This menu item spawns a API for viewing and editing the PLS tables on the CLC card and DDS2.2 version 4 drives. Each CLC program has a 16-position programmable limit switch(PLS), each drive a 8 position PLS. The card PLS can be driven by an external feedback or an internally generated position command(Virtual Master). The drive PLS is driven by it’s primary or secondary axis feedback.

Menu Items:

File  Programs: Selects a program on the CLC card. Each program has a unique PLS. “Currently active” choses the active program.
      Exit: Exits this utility.
Edit  Display the edit dialog for the PLS chosen.
Selection  Choose a card or drive based PLS for display or edit.
Settings  Selects the DDE topic to communicate with a CLC card.
Refresh All  Warm boot of this utility, all program name, labels and current card PLS values are uploaded.

The initial screen is only for viewing, it graphically displays the “ON” period of each switch. On the bottom is the assigned register and current offset. To the right is the status of the assigned register bits and bit labels. For display purposes the default range of the graph is 0 to 360, other ranges can be entered on the edit screen.

To edit the PLS, select the EDIT menu item, or click on the graph on the switch to be set. In either case, another dialog box will appear, change the fields and press the SAVE button to send to CLC. Based on the type of PLS( card or drive), some of the the following PLS variables can edited on this screens:

Assigned Register
Master Type
Master Axis( card )
Current Offset( card )
Switches On Position
Switches Off Position
Lead Time( drives)
Graph Limit Minimum Value
Graph Limit Maximum Value
Card PLS Overview
The CLC provides a 16-position programmable limit switch per program. This limit switch can be based on either an external feedback or an internally generated position command. The PLS can be modified and set in a program utilizing the “Calc” icon or interactively through the PLS display. In the “Calc” icon, individual elements of the PLS can be changed (i.e. PLS[1].{a, o, r, t, on1-on16, off1-off16}). The PLS can be initialized, at compile time, using the PLS icon.

Limit Switch Operation
The programmable limit switch (PLS) is defined as 16 sets of position ranges for outputs. When the master position is in the On range, the outputs are on. When it is in the Off range, the outputs are off. The position ranges can overlap. The positions must be greater than or equal to the positions in the table to be in a position range. The position ranges in the table are defined clockwise, but the switch operates in either direction. If both the On range starting position and the Off range starting position are 0, the output is always off.

Example Programmable Limit Switch

<table>
<thead>
<tr>
<th>Output</th>
<th>On Position</th>
<th>Off Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100</td>
<td>160</td>
</tr>
<tr>
<td>2</td>
<td>30</td>
<td>220</td>
</tr>
<tr>
<td>3</td>
<td>90</td>
<td>80</td>
</tr>
<tr>
<td>4</td>
<td>200</td>
<td>290</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>350</td>
</tr>
<tr>
<td>6</td>
<td>270</td>
<td>100</td>
</tr>
<tr>
<td>7</td>
<td>180</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>95</td>
<td>85</td>
</tr>
<tr>
<td>9</td>
<td>130</td>
<td>185</td>
</tr>
<tr>
<td>10</td>
<td>27</td>
<td>95</td>
</tr>
<tr>
<td>11</td>
<td>122</td>
<td>330</td>
</tr>
<tr>
<td>12</td>
<td>330</td>
<td>122</td>
</tr>
<tr>
<td>13</td>
<td>310</td>
<td>10</td>
</tr>
<tr>
<td>14</td>
<td>49</td>
<td>139</td>
</tr>
<tr>
<td>15</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>16</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Output States at 90 degrees
(phase offset = 0)

<table>
<thead>
<tr>
<th>Output</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>OFF</td>
</tr>
<tr>
<td>2</td>
<td>ON</td>
</tr>
<tr>
<td>3</td>
<td>ON</td>
</tr>
<tr>
<td>4</td>
<td>OFF</td>
</tr>
<tr>
<td>5</td>
<td>ON</td>
</tr>
<tr>
<td>6</td>
<td>ON</td>
</tr>
<tr>
<td>7</td>
<td>OFF</td>
</tr>
<tr>
<td>8</td>
<td>OFF</td>
</tr>
<tr>
<td>9</td>
<td>OFF</td>
</tr>
<tr>
<td>10</td>
<td>ON</td>
</tr>
<tr>
<td>11</td>
<td>OFF</td>
</tr>
<tr>
<td>12</td>
<td>ON</td>
</tr>
<tr>
<td>13</td>
<td>OFF</td>
</tr>
<tr>
<td>14</td>
<td>ON</td>
</tr>
<tr>
<td>15</td>
<td>OFF</td>
</tr>
<tr>
<td>16</td>
<td>OFF</td>
</tr>
</tbody>
</table>

The On range for Output 1 starts at 100 degrees, and the Off range at 160 degrees. The master is moving continuously. Output 1 will be on for 60 degrees between 100 and 160, and off for 300 degrees between 160 and 100.
The PLS menu item spawns an API for viewing and editing the PLS table on the CLC card.

![Active Program, PLS - Card 0](image)

A listing of programs on the CLC card is provided under the File menu item. “Currently Active” is also listed to access the PLS of the active program.

The initial screen is only for viewing, it graphically displays the “ON” period of each switch. On the bottom is the assigned register and current offset. To the right is the status of the assigned register bits and bit labels.

To edit the PLS, select the EDIT menu item, or click on the graph on the switch to be set. In either case, another dialog box will appear, change the fields and press the SAVE button to send to CLC.

The following PLS variables can be edited on this screen:

- Assigned Register
- Current Offset
- Switches On Position
- Switches Off Position
Phase Offset
The programmable limit switch has a phase advance which allows outputs to be turned on or off before or after their programmed position. The phase advance applies to all PLS outputs, and may be changed on the fly to correspond to different products or line speeds. The phase advance is specified as a position in degrees. This position is added to each position in the programmable limit switch table. For Output 1 in the example above with a phase advance of 5 degrees, the output will be on between 105 degrees and 165 degrees.

Master Type
The master axis is parameter-defined, with the following choices:
1. ELS master
2. Virtual master
3. Real master
4. On Drive

For many applications, the **ELS master** will be used. The ELS Master option allows synchronization of the limit switch outputs with the rest of the machine. The PLS will follow the same master that the ELS axes are following.

When the **Virtual Master** option is selected, the CLC always follows the ELS virtual master, even if the ELS slave axes are following a real master.
When the **Real Master** option is selected, the CLC follows one of the CLC’s real masters. The real master can be independent of the ELS master, and includes a set of parameters to define gear ratio, feedback type, and filtering. In the current CLC version, there can be up to two real masters in the system.

The **On Drive** option uses the PLS function built into Indramat DIAX03 type drives. As many PLS’s as there are drives can then be defined in the system. The accuracy of the PLS is much better, since it is updated directly at the drive. The PLS is stored on the drive in parameters P-0-0131 to P-0-0135. The PLS_init command automatically sets these parameters when the program is first activated.

**Register Designation**

For maximum performance, the programmable limit switch is assigned to a register on the CLC. The I-O Mapper and image table are bypassed for this register. This allows drive-based I-O to be updated as the position is updated, regardless of the I-O scan time. This register can be associated with DDS DEA outputs or any other I-O card or PLC register. The register designation is defined in a system parameter.

The PLS can be disabled initially or “on-the-fly”. If the assigned register number is set to 0, the PLS is disabled. The CLC leaves the outputs at the last state, and allows the user program and I-O Mapper to control them. If the application requires clearing the I-O after the PLS is disabled, the register must be cleared in the user program immediately after the PLS is disabled.
Drive Based PLS

Overview
The DIA03 drives, version 04, include an 8-position limit switch that can be driven from the primary or secondary axis feedback. The drive includes parameter lists for on positions, off positions, and lead times. The status of the position switch is evaluated by the drive every 2ms, and an IDN in the drive is set accordingly. The outputs can be updated immediately by the drive, or in the next I-O cycle by the CLC.

Differences Between Drive and CLC PLS
The CLC-based PLS is stored with the user program as a separate table, and can be tied to any ELS master or real master. It has 16 switches, with a single position-based phase offset for all switches. Its elements can be easily changed at any time using a CALC expression.

The drive-based PLS is stored on the drive in parameter lists. It has 8 switches, with a separate time-based phase offset for each switch. The elements of this list cannot be changed individually. Changing one element of the list requires sending the entire list through the service channel.

<table>
<thead>
<tr>
<th>PLS Type</th>
<th>Stored in</th>
<th>Outputs per PLS</th>
<th>Phase Offset</th>
<th>DEA outputs updated</th>
<th>Elements changed</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLC</td>
<td>user program on CLC</td>
<td>16</td>
<td>one position-based offset</td>
<td>by CLC 2ms after position</td>
<td>immediately</td>
</tr>
<tr>
<td>Drive</td>
<td>EEPROM on DIA03 drive</td>
<td>8</td>
<td>individual time-based offsets</td>
<td>by drive at same time as position</td>
<td>via parameter list (up to 200ms per list)</td>
</tr>
</tbody>
</table>

User Program
Unlike the CLC-based PLS, the limit switch cannot be compiled into the program or changed dynamically via the CALC command. Therefore, the existing PLS/INIT command and the PLS table operands will apply only to CLC-based PLS. Drive based PLS are setup interactively using the PLS utility.

Assigned Register
Changes to the assigned register are saved to the drive but do not take effect until the card makes the transition from “Parameter Mode”.

IAE 74792
CLC
4.7.5. Points

The View Point Table command permits the user to view and edit the absolute and relative points in a program that has already been downloaded to the CLC card. The dialog box automatically uploads and displays the contents of the point table for the currently active program on the CLC. To examine point tables of other CLC-resident programs, use the File/Program Select command.

Points are used in coordinated motion programs to describe a location in Cartesian coordinates, tool orientation, and associated events. Some programs may not have points allocated in the Size icon. Each point has: { x, y, z, blend, speed, acceleration, deceleration, jerk event 1, event 2, event 3, event 4, roll, pitch, yaw, and elbow } fields.

Programs on CLC card - program selection is through a combo box containing a list of programs on the CLC card plus "Currently active".
Point Table - These radio buttons selects the absolute or relative point tables for viewing and editing. Relative points are from the last location.

<table>
<thead>
<tr>
<th>File</th>
<th>Edit</th>
<th>View</th>
<th>Table</th>
<th>Update</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>y</td>
<td>z</td>
<td>Blend</td>
<td></td>
</tr>
<tr>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
<td>50</td>
</tr>
<tr>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
<td>50</td>
</tr>
<tr>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
<td>50</td>
</tr>
<tr>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
<td>50</td>
</tr>
</tbody>
</table>

To better navigate long tables, the user may click and drag the vertical scroll bar button while looking at the point number indicator that appears at the right of the dialog box title bar. The point number corresponds to the top of the list when the scroll bar button is finally released. The horizontal scroll bar permits scrolling the display to show the additional roll, pitch, yaw points and elbow bit used with a six axis CLC system.

Another viewing aid is the View menu, which presents a list of the point elements and allows the user to deselect any number of them in order to exclude them from the display.

The Table menu allows the user to switch between the absolute and relative point tables.

Points may be edited by double-clicking on a point in the list, or highlighting the point and then clicking Edit. Editing a point table entry opens an absolute or relative Edit Point Values dialog box, depending on which type was selected with the Table menu.
The Edit Point Values dialog boxes permit individually changing the values for each point table entry on the CLC card. Clicking on Update immediately downloads the new values to the program on the CLC card. The changed values are displayed at the next automatic update. For more information on the absolute or relative point tables and how they are used, refer to the appropriate section in the chapter, “Programming Concepts.”
4.7.6. Registers

This Windows utility allows viewing and editing of registers on the CLC card. There are 1024 registers on the CLC-V card, and 512 registers on the CLC-P and CLC-D. See the Control and Status Registers for a description of the registers reserved by the CLC for system control and status.

Registers can be changed by editing the register, editing the register’s bits, or forcing the register bits. Register priority is as follows from highest to lowest:

- VME/PC Access - registers written directly by an external device.
- I/O Drivers - registers control by I/O device drivers.
- Register Forcing - via serial port.
- I/O Mapper - via I/O mapper equations.
- Direct Register Access - via serial port or program instructions.

On the CLC/V these registers are accessible from the VME address space by other VME cards. On the CLC/P these registers can be accessed through the twoport RAM on the ISA bus by the PC.

The Physical Name of the registers in the listbox are taken from register labels, register labels can be assigned to non-system registers.

Data Display - These radio buttons selects the display format of the register data.

Choosing the Registers menu item from the Data menu opens a Registers dialog box listing the register ID numbers and the symbolic label names associated with the registers (from the register.lst file). When the dialog box opens, it automatically uploads and displays the current contents of the CLC registers. Note that the CLC has a maximum of 1024 registers.
The **Format** pull-down menu allows you to change the displayed format of the register data.

The **Options** pull-down menu shows the available F-key commands for editing, forcing, and clearing. Selecting a list entry and pressing F3 opens an Edit Reg dialog box allowing change of the value in the selected I/O register table. Clicking the Save button downloads the changed value to the CLC.

![Edit Reg (1 System_Control)](image)

Selecting a list entry and pressing F2, or simply double clicking on a list entry, opens a Bit Names dialog box for the selected register and displays the bit names and values associated with the register. Bit names are obtained from the "bit.lst" file in the Host system :\CLC subdirectory and are not uploaded from the CLC.

![Bits of Register( 1 - System_Control )](image)

Selecting a list item and clicking on the "0 <= > 1" button inverts the current state of the bit and downloads the change of state to the CLC through the serial communication link. The next periodic upload from the CLC reflects the change of state in the displayed list.

Bits controlled by the CLC I/O, user program, or I/O Mapper will not be altered using the "0
< = > 1" toggle button. However, their state may be "forced" using the Forcing option (F4). Items denoted with an "x" in the rightmost column have forcing in effect for the associated bit.

System installation and troubleshooting may require directly changing the state of register bits without depending on the I/O sub-system. Selecting a register and pressing F4 opens a Register Forcing dialog box that allows you to setup a forced bit during system setup and debugging.

Forced bits cannot be affected by a CLC program or the I/O sub-system. Forcing directly accesses the CLC system's I/O lines. Bits that are forced will remain in the forced state until the forcing is changed, the CLC card is reset, or power is cycled on-to-off.

Forcing employs forcing mask and forcing state 16-bit control words allowing you to change a single bit, or combination of bits, within the register without affecting the other bits.

The Forcing mask value enables which bits may be affected by forcing. The Forcing state value determines the actual state of the enabled bits. The following algorithm mathematically describes how the final register state is set:

New register state = (old state & inverted forcing mask) | (forcing state & forcing mask)

Example:

| forcing mask | 0000 0000 0000 1000 |
| forcing state | 0000 0000 0000 1000 |
| inverted forcing mask | 1111 1111 1111 0111 |
| old register state | 0000 1101 1010 0001 |
| new register state | 0000 1101 1010 1001 |

**WARNING:** Bit forcing directly changes the state of the CLC's inputs and outputs. Forcing I/O bits can result in harm to people and equipment. Make sure you fully understand all the effects on the system that could result from forcing an I/O line.
4.7.7. Sequencer

The CLC Sequencer provides the user with a facility for making machine operational changes without having to edit, recompile and download a program to the CLC Card. A Sequencer is a list of steps which contain one or more functions, with up to five arguments per function.

When the Sequencer is selected from the Data menu, Visual Motion reads the Sequencer information from the CLC card and displays it in the Sequencer editor. This allows the user to view and edit a Sequencer list on-line. An error will occur if there are no Sequencer lists in the active program. A Sequencer List needs to be initialized using the Sequencer icon before it can be edited on-line. See Chapter 5 - Programming Concepts for more information.

Sequencer Editor
The caption on the window identifies the active CLC card number and program. The initial screen provides a drop down combo-box which lists all the Sequencer list names found in the selected program. The box below it shows the order of all the Steps used with the selected Sequencer.

Select “Program” under the “File” menu to list programs currently on the CLC card. When a different program is selected, the associated Sequencer tables will be uploaded and the program name will appear in the window caption. The “File” menu also allows the user to save refresh or print the Sequencer table.
The “Edit” menu items allow the user to insert, append, delete, copy, paste and rename steps within a Sequencer list. This editing is also available through icons on the tool bar.

The steps can also be moved by selecting and dragging them with the mouse. Select Step List to show all available Steps.

The “Connection” item under the “Settings” menu allows the user to select one of the following connections: serial, XYCOM VME, PC ISA bus, AT Modem, Demo, GE Plug & Play or CLC File. The card number, 0 through 15, can also be selected. The DDE server name is also listed along with its topic string. This information is useful when creating client applications which can talk to the CLC communications server. See section 4.5.1. Card Selection.

Selecting "Configuration" allows the user to adjust the DDE Synchronous Time-out. This is the amount of time in seconds the Sequencer Editor will wait for a response from the DDE server before proceeding. See Appendix A - CLC DDE Server for more information. Sequencer Configuration also allows the user to monitor the Sequence System and/or Active Program for change.
Functions and arguments can be edited by double clicking on a selected step. A Step List window will open with the same edit and icon tools on top. Individual arguments can be edited by double clicking on them. After making any changes the Step List needs to be saved to retain the current edits in the Sequencer.

Argument value adjustments can also be sent directly to the CLC card while the program is running by selecting Send. This can only be done if the Step List order hasn’t changed since the last time it was saved. If a change has been made to the function order, this option will gray out. A Step List can not be saved to a Sequencer if it is running.
An entire function can be edited or replaced by double clicking on the function name. A list of available functions will open. These are all the predefined subroutines within the Visual Motion program. After selecting a function, the user will have to enter a value for each argument. This value will have to be within the programs predefined limits. Selecting a function this way changes the order of the Step List. The user will not be able to send the resulting argument values to the CLC card without first saving the Step List.

4.7.8. Variables

The CLC Variables dialog box permits viewing the integer and floating point variables of a CLC resident program. The dialog box automatically uploads and displays variables of the active program on the CLC. The list display and variable selection for editing is similar to the View Point Table dialog box. The program is selected with File/Program Select and the variable type is selected under the Types menu.

Selecting a list entry and clicking the Edit button, or double clicking the list entry, opens a Variable Edit dialog box allowing change of the value in the selected variable in the CLC resident program. Exiting the dialog box automatically downloads the changed values to the CLC.
4.7.9. Zones

This Windows utility allows viewing and editing of the zone table on the CLC card. Zones are used in coordinated motion programs to describe a volume of space where motion of any kind is prevented. Some programs may not have zones allocated in the Size icon.

Each zone has: \{ status, a, b, c, d, e, f \} fields. \{a, b, c\} and \{ d, e, f \} are the \{ x, y, z \} coordinates of two points in space that describe a rectangular box.

Status is if the zone is ACTIVE or INACTIVE.

Programs on CLC card - program selection is through a combo box containing a list of programs on the CLC card plus "Currently active".

To edit a event, select it in the list box, and press the EDIT button, or double-click on the list item. In either case, another dialog box will appear with a edit field. Change the fields and press the SAVE button to send to CLC.

![Active Program Zones - Card 0](image)
4.8. The Status Menu

4.8.1. Diagnostic Log

This window lists the last 100 errors the CLC card has encountered. Along with the error messages, the date, time and extended error codes are displayed.

<table>
<thead>
<tr>
<th>Date</th>
<th>Time</th>
<th>Error Message</th>
<th>Sec. Code</th>
<th>Ext. Sec. Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>01-02</td>
<td>04:50:41</td>
<td>409 SERCOS Disconnect Error</td>
<td>25</td>
<td>0</td>
</tr>
<tr>
<td>01-01</td>
<td>20:31:45</td>
<td>409 SERCOS Disconnect Error</td>
<td>25</td>
<td>0</td>
</tr>
<tr>
<td>01-01</td>
<td>20:31:44</td>
<td>420 Drive 1 Shutdown Error</td>
<td>F861</td>
<td>0</td>
</tr>
<tr>
<td>01-01</td>
<td>15:53:38</td>
<td>409 SERCOS Disconnect Error</td>
<td>25</td>
<td>0</td>
</tr>
<tr>
<td>01-01</td>
<td>15:10:31</td>
<td>409 SERCOS Disconnect Error</td>
<td>25</td>
<td>0</td>
</tr>
<tr>
<td>01-01</td>
<td>12:19:53</td>
<td>411 Drive 1 Phase 4 Switch Error</td>
<td>C202</td>
<td>0</td>
</tr>
</tbody>
</table>

Date and time are relative to the power on of the CLC D/P cards, they have no battery backed clock. The time can be set in card parameter C-0-0126.

The Log Options button opens an options window. It allows some common errors to be ignored and saving the diagnostic log to a file. Also see card parameter C-0-2020.

4.8.2. Drives

Refer to Appendix D. CLC Drive Parameter Editor.
4.8.3. Drives on Ring

This window displays the addresses, uses and identification of the drives and motors the CLC card sees on the SERCOS ring.

4.8.4. System

Choosing System from the Status menu opens the System Parameters dialog box. System Parameters displays information about the current CLC card hardware and software for the indicated unit number; and the total memory and free memory on the CLC card.
4.8.5. Tasks

Selecting Task from the Status menu opens the Task_A Parameters dialog box and uploads data regarding the current CLC task. The other tasks may be viewed by clicking on the Previous or Next buttons.

**Task_A Parameters**

- **Status**
  - Type: Non coordinated motion
  - Mode: Automatic

- **Current Instruction**
  - Pointer: 0x000000D8
  - Text: 0888 BRA
  - Error at: 0x00000000

- **Current Messages**
  - Status: Task Running
  - Diagnostic: Task Running

**Coordinated Axes**

- Assigned Drive:
  - X: 0
  - Y: 0
  - Z: 0

- Current Position:
  - X: 0
  - Y: 0
  - Z: 0

**Status** indicates the type of motion programmed in the selected task for the active program and the current CLC mode (Parameter, Initialization, Manual or Automatic).

**Current Instruction** displays the instruction executing and its pointer, and a pointer to a runtime error if one has occurred. This display is useful when debugging in single-step mode. If a program is running in automatic mode, the displayed instruction is the instruction that was executing at the time that the SERCOS cycle sampled instruction execution, which may appear to be random.

**Current Messages** displays the last messages encountered in the program.

**Coordinated Axes** displays the axes in the active task that are assigned to coordinated motion and their current position.
4.9. The Options Menu

The Options menu provides pre-configured palettes for single axis, coordinated, and Electronic Line Shaft (ELS). Choosing one of these menu items loads the selection onto the palette area on the left of the VisualMotion workspace.

**Icon Palette 'Single'** from the Options menu selects a set of icons used frequently in single axis control.

**Icon Palette 'Coord'** selects a set of icons used frequently in coordinated control.

**Icon Palette 'ELS'** selects a set of icons used frequently in electronic line shaft (ELS) control.

**Icon Palette 'Utility'** selects a set of general purpose icons used in parameter setup.

Choosing **Icon Comments** enables/disables identifying comments that appear when the mouse cursor is moved over the top of the icon. The icon comments in a program flow can be modified in the corresponding icon setup box. The setup box appears when the icon is first placed or by double clicking the left mouse button while the pointer is over the icon. Each box has a "Label" button to edit its comment text, by default there is no comment. The comment also appears on the printout of the setup information, if enabled or not.

Choosing **Icon Labels** from the Options menu alternately turns the icon labels in the VisualMotion workspace on or off. If there are no user entered labels, VisualMotion uses the default icon labels.
4.10. Help

VisualMotion's Help menu accesses the on-line help system.

Selecting Help/Getting Started or pressing <F1> opens the main help screen, from which you can browse through the system by pointing the cursor at the green keywords (help topics) and clicking.

Selecting Help/Search opens a dialog box into which you can type a keyword to go directly to a specific help topic.

Help/Change Log displays a list of the new features that appear in your version of VisualMotion.
CHAPTER 5. PROGRAMMING CONCEPTS

5.1. Overview

Indramat provides two methods for writing CLC motion control programs, the CLC Text Language and CLC Icon Language. Both methods of programming have many characteristics in common. This chapter describes these common elements and their implementation.

5.2. Program Tasks

The CLC provides up to four user-defined motion control tasks for control of up to forty axes. The CLC motion control tasks are named A, B, C and D. A single CLC task may support any number of non-coordinated axes. However, each task is limited to two or three coordinated axes of motion.

5.2.1. Command Execution

Normal multi-tasking is achieved by consecutively executing one instruction from each task. During normal operation, the CLC does not wait for an instruction to be completed before processing the next instruction. Also, the CLC’s multi-tasking path planner and the use of Indramat DDS intelligent digital drives permits simultaneous execution of multiple motion commands.

The CLC path planner also uses several levels of priority to control the execution of program statements. For example, coordinated multi-axis motion may require some instructions to wait for a specific type of motion to complete before proceeding to the next instruction. The CLC priorities associated with user programs are listed below. A higher numbered task has priority over any lower numbered task.

<table>
<thead>
<tr>
<th>Highest</th>
<th>Priority 7 - CLC Executive Path Planner and Single Axis Events</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Priority 6 - Event from Task A</td>
</tr>
<tr>
<td></td>
<td>Priority 5 - Event from Task B</td>
</tr>
<tr>
<td></td>
<td>Priority 4 - Event from Task C</td>
</tr>
<tr>
<td></td>
<td>Priority 3 - Event from Task D</td>
</tr>
<tr>
<td></td>
<td>Priority 2 - CLC Executive Timer Task (Repeating Events)</td>
</tr>
<tr>
<td>Lowest</td>
<td>Priority 1 - User Motion Tasks A, B, C, D, Teach Pendant</td>
</tr>
</tbody>
</table>
5.3. Events

CLC events are a privileged form of subroutines. During the execution of an event, all CLC program tasks are suspended until all pending events have completed. The CLC system executive program maintains an event queue or stack for events associated with each task. This permits each task to have up to four pending events active at any time. See Chapter 4. **Visual Motion Menu Commands** for more information.

Events are prioritized according to Task, from A to D. Events associated with Task A will interrupt execution of events associated with Tasks B, C, & D. Events associated with Task B will interrupt events associated with Tasks C & D, etc.

To add an event subroutine to a user program, select “Add Event Function” from the “Edit” menu in VisualMotion. After naming the event function (subroutine), a blank page will appear for programming the event. Event functions must begin with a “Start” icon and end with a “Finish” icon, just like a normal subroutine. Any VisualMotion icon that executes during run-time can be used in the event function subroutine. Events are armed in the user program with various icons (see following sections), depending upon how the event is to be called

**See 5.7.1. Event Table for information regarding Event Table Data.**

**NOTE:** Because an event preempts all user tasks, events can adversely effect task response time and execution. Events should be used only when real-time response is needed and should be kept as short as possible. For example, involved calculations or waiting for other I/O will block the execution of all tasks that are associated with that event. If possible, use the event routine to set a flag value in an integer variable and immediately return. Then test the flag during an idle period and perform the required operations at the later time (don’t forget to reset the flag).

5.3.1. Time-based Events

The CLC provides two types of time based events, repeating timer events and events that are time related to a coordinated motion path.

Once a repeating timer event is armed in the user program, it executes periodically at the programmed time interval until the event is disarmed. The timer period for the event is specified in the event table (argument) and has a resolution of 20 milliseconds. The event number to execute every timer period is specified in the “Event” icon. After the user program is compiled and downloaded, the event type – Repeating Timer --, event function, and timer period must be specified in the event table. The event table is loaded by selecting “Events” under the “Data” menu in VisualMotion.
Coordinated motion can provide time based events that are related to travel time along a specified geometry segment and are initiated by the path planner. These events execute at a fixed time period after motion starts or before motion ends on the specified segment. The time period is specified in milliseconds in the event table.

Figure 1: Repeating Timer Events

NOTE: CLC single axis non-coordinated motion instructions use the internal intelligence of the DDS drives for positioning. Because the rate profile for single axis motion is developed within the drive, the time method of triggering an event related to motion cannot be supported.

5.3.2. Distance-based Events

Distance based events are triggered after the distance specified has been traversed on the geometry segment. They can be triggered off both coordinated and non coordinated (single-axis) moves. The response time for coordinated motion events is dependent upon the number of coordinated axes that are programmed and the task that calls the event (task priority A - D). Each task using coordinated axes requires one additional SERCOS cycle. Therefore, the response time may vary from one to four SERCOS cycles.
**Single axis non-coordinated motion** requires specification of the trigger distance from the endpoint or beginning of motion using an absolute number of axis units. The event number to be executed is specified in the "Optional Events" section of the single axis move icon. After the user program is compiled and downloaded to the CLC the following information must be in the event table:

- Event type
- Single axis distance from the end or from the beginning
- Event function
- Trigger distance

The event table is loaded by selecting “Events” under the “Data” menu in VisualMotion. Single axis distance based events cannot be used on the same axis with repeating axis position events.

![Single Axis Move Setup](image)

![Edit Event 2 Values](image)

**Figure 2: Single Axis Event**

**Multi-axis coordinated motion** requires the trigger distance specified as a percentage of the total length of the segment. The event number to be executed is specified in the “Event” icon on the “Coord” icon palette. After the user program is compiled and downloaded to the CLC, the following information must be specified in the event table:

- Event type
- Percent of coordinated path from start or before end
- Event function
- Trigger distance
The event table is loaded by selecting “Events” under the “Data” menu in VisualMotion.

![Event Setup Box](image)

**Figure 3: Coordinated Distance Based Event**

When specifying a distance-based event trigger with coordinated motion, an event occurring within a blend segment may not trigger as anticipated. The range of potential paths that could be generated by the path planner through the blend segment must be considered. Consider when one segment is blended into another, with one event set to trigger near the end of the first segment and another event set to trigger near the beginning of the next segment. If the blend radius specified is sufficiently large, the second segment may blend into the first segment far enough that the second segment event triggers before the first segment event can occur. This situation can be controlled by ensuring that the programmed blend radius is smaller than the specified trigger distance of the second segment.

### 5.3.3. Repeating Axis Position (Rotary) Events

Repeating Rotary Events trigger each time an axis encounters an absolute position. The axis can be in single-axis, ELS, ratio, or velocity mode and can be configured for modulo or non-modulo positioning. See **AxisEvt Icon description**.

Each time the event position is passed in either direction, the function specified in the event table is executed. Since rotary motion uses the shortest path to reach the next specified position, you should take care that the axis will actually travel through the position specified to trigger the rotary event. Otherwise, the event will not occur.
Repeating rotary events only need to be armed once in a user program. After the user program is compiled and downloaded to the CLC, the event type – Repeating Axis Position, event function and trigger position must be specified in the event table for each rotary event. The event table is loaded by selecting “Events” under the “Data” menu in VisualMotion. If changes are made to the arguments of rotary events in the event table, then the “Rotary Events” icon must be executed again in the user program before the changes will take effect.

**NOTE:** Rotary events cannot be attached to an axis that has a CLC based PLS (Programmable Limit Switch) attached to it or with single axis distance based events.

Figure 4: Repeating Rotary Events

5.3.4. Interrupt Input Events

Bit 9 in each Task’s Control Register is reserved as an Event Interrupt Input for the task. Each low-to-high transition of this input can trigger an event in the corresponding task. An Interrupt Input type event permits triggering an event function from an external input. The event number to be executed is specified in the “Event” icon on the “Coord” icon palette. After the user program is compiled and downloaded to the CLC, the event type, task input transition, and event function must be specified in the event table. The event table can be loaded by selecting “Events” under the “Data” menu in VisualMotion.

The CLC scans the input every 2ms and queues an event upon a low-to-high transition. The event function will take priority over the user tasks, allowing quick response to an external input.

The I/O mapper can be used to invert the logic of the interrupt input, or to direct other external inputs to the Task Control Register’s Event Interrupt bit. Logic in the event function can then scan the multiple inputs to determine the source of the interrupt.
5.3.5. VME Events

The VME version of the CLC supports the ability to trigger an event on one or more cards on the VME bus.

Short Address Event
Using the VME short address space, a single event can be triggered on one other card.

A master device triggers the event by reading a byte from the location of mailbox 4 (A16:D8). The read generates a local interrupt on the CLC card, scheduling the proper task event handler to run. If the target CLC card acknowledges the trigger, the uppermost bit of the read operation will be 0. If a previous interrupt generated by reading the mailbox is still pending, the uppermost bit of the data will be 1. The CLC’s VME short address space mailbox 4 is located at an offset of 16 (0x00010) from the CLC card’s base address.

A local VME event can be initiated by using an Icon Language VMEEvt icon, or by a Text Language VME/EVENT instruction.

VME Broadcast Event
Later versions of CLC hardware, which include the connector P2 for accessing VME extended address space, can trigger an event simultaneously across multiple cards.

The broadcast message specifies the task to which the event is bound, and the event number to run. The broadcast message generates a local interrupt on the receiving card(s) which schedules the proper task event handler to run. If any CLC cards addressed by the broadcast message cannot accept it because a card has a previous message pending, all of the addressed cards will refuse the message and generate an error.
The unit's broadcast message address is a bit mask specifying the units to be affected. If the bit corresponding to a given unit number is high, then the unit will respond to the message.

The CLC message broadcast area always occupies VME address space FA000000-FA0FFFFFF (A32) even when broadcast messages are not used. No other VME devices on the bus may use this address space. The base address for calculating a broadcast message address is FA800000. Address lines A2-A17 are used to designate units 0 - 15.

The appropriate CLC card (unit) addresses are listed in the following table.

<table>
<thead>
<tr>
<th>VME Unit</th>
<th>32-bit Address</th>
<th>VME Unit</th>
<th>32-bit Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0xFA80 0004</td>
<td>8</td>
<td>0xFA80 0400</td>
</tr>
<tr>
<td>1</td>
<td>0xFA80 0008</td>
<td>9</td>
<td>0xFA80 0800</td>
</tr>
<tr>
<td>2</td>
<td>0xFA80 0010</td>
<td>10</td>
<td>0xFA80 1000</td>
</tr>
<tr>
<td>3</td>
<td>0xFA80 0020</td>
<td>11</td>
<td>0xFA80 2000</td>
</tr>
<tr>
<td>4</td>
<td>0xFA80 0040</td>
<td>12</td>
<td>0xFA80 4000</td>
</tr>
<tr>
<td>5</td>
<td>0xFA80 0080</td>
<td>13</td>
<td>0xFA80 8000</td>
</tr>
<tr>
<td>6</td>
<td>0xFA80 0100</td>
<td>14</td>
<td>0xFA81 0000</td>
</tr>
<tr>
<td>7</td>
<td>0xFA80 0200</td>
<td>15</td>
<td>0xFA82 0000</td>
</tr>
</tbody>
</table>

To select multiple units, OR the addresses. For example, to send a broadcast message to units 0 and 5, the master would write to VME address FA800084.

The data associated with a broadcast message contains the task ID and event index. Any VME data width formats can be used to write to the broadcast address, however, only the lower byte is used. The upper 2 bits of the broadcast message designate which task the event should be bound to. Tasks are assigned the following bit mask values: A = 00, B = 01, C = 10, and D = 11. The lower 6 bits of the message specify the event number (table index).

### 5.3.6. Feedback Capture (Probe)

The CLC uses the SERCOS Probe Functions and Real Time Bits along with the Event System to allow user programs to perform registration functions. The Indramat DDS drive provides two probe inputs that can be used for capturing the feedback position. D1ax02 drives capture the position with 250 µsec and D1ax03 drives capture the position within 5 µsec. Upon either a positive or negative transition of a probe input, the DDS captures the position into the cyclic data. Since the captured feedback positions must be included in the SERCOS cyclic data telegram, the probe setup icon must be included in the user program for each drive that will use the probe function.
The probe event must be armed each time it is to be executed. Typically, probes are used to detect registration marks on material. By controlling when the probe is armed, other printing on the material can be filtered out. When the position is captured, the drive signals the CLC with a real time bit in the SERCOS telegram. When the CLC detects a change in the real time bit, it can execute an optional event function. The event number to execute is specified in the “ProbeEvt” icon. After the user program is compiled and downloaded to the CLC, the event type, feedback capture, and event function to execute must be specified in the event table. The event table can be loaded by selecting “Events” under the “Data” menu in VisualMotion.

5.3.7. Event Tables

See 5.7.1. Event Table and Appendix B. Direct ASCI Communications
5.4. Subroutines

Subroutines can be used to pass function arguments and can include local variables. This eliminates the need the keep and maintain temporary variables and allows user programs to be structured more effectively. Subroutines can also be re-used by more than one task.

The Sequencer uses subroutines to perform data driven programming that allows function arguments to be adjusted without changing the program.

5.4.1. Function Arguments and Local Variables

Function arguments are stack based variables passed to a subroutine. The setup dialog box for the Sub icon has entries for the function arguments. Up to 5 arguments may be passed to a subroutine. The arguments can be constants or variables (float, integer, ABS index or REL index). The function arguments need to be declared in the subroutine Start icon dialog box. A minimum and maximum value must also be entered for each of these arguments. This limits the range of argument values used during Sequencer editing from the teach pendant or the Sequencer interface.

Function arguments are not allowed in tasks or events, however, an optional return argument may be passed back to a task from a subroutine. A Return Argument is an optional single argument that may be returned from a subroutine. The setup dialog of the Finish icon of the subroutine screen has an entry for the return argument, it may be a constant or variable. The setup dialog for the Sub Icon has an entry for a variable to receive the return argument, if no variable is entered the returned argument is discarded. A return argument is a convenient way to “get position”, utilizing a common subroutine from more than one task.

Local variables are stack based variables used in a function. The total number of local variables and function arguments is limited to 16. Local variables, floats and integers are initialized to zero on each entry into the function. This type of variable can be used to store temporary results in a re-entrant function called on by more than one task. Local variables can also be used to avoid possible conflicts with program or global variables between tasks.
5.5. Sequencer

The CLC Sequencer provides the user with a facility for making machine operational changes without having to edit, recompile and download a program to the CLC Card. Programs can be built or modified using predefined and tested functions or subroutines that are tailored to the machine. Each function can contain up to five argument values. Once these functions are in placed within a program a Sequencer can execute them in any order. To make a change, the user can simply add functions to a Sequencer Step List, then edit the function arguments as required. When the program runs, the CLC executes each function sequentially, in the order it is listed.

A Sequencer List is essentially a "to do" list. This list is broken down into individual steps. Each step contains a list of one or more functions with up to five arguments per function. The order of program flow sequentially follows this "to do" list through each of its steps and functions. The function or subroutine will then run with the values of the function arguments given in the Step List. Each Sequencer List can contain up to 30 steps.

The "Callable from Sequence" access flag must be set in the START icon of the subroutine before that function can be added to a Step List. Function arguments must also be defined before they can be used. See SUB icon.
The maximum number of different Sequencer lists and steps is configurable in the **SIZE icon**. If the same list or step is used more than once, it will only count as one towards the total. The number of functions refers to the total number of all functions executed and is independent of the number of Lists and Steps. Every function in a step counts, even if it is the same function used more than once. Sequencer lists, steps and functions are identified by a unique user determined name which can be up to twenty characters long.

Any Sequencer List or Step can be edited with the teach pendant and/or with VM. When using VM, the Sequencer editor under the Data menu can be used to edit or add to a Sequencer list. If the teach pendant is used the resulting software will be upwardly compatible with existing programs.

The Sequencer list can also be initially setup in the program within the **Sequencer icon**. If a Sequencer is initialized and setup in a program, any changes made to it with the Sequencer editor will be overwritten when the icon is executed.

### 5.5.1. Single Stepping a Sequencer Step

Bit 12 in the Task_Control_Registers( 2-5 ) is used to single step a Sequencer step. When set to 1, it instructs the CLC to stop program execution after each Sequencer step. Toggling the Cycle_Start (bit 6 ) will continue execution through the next step.

### 5.5.2. Single Stepping a Step Function

Bit 13 in the Task_Control_Registers( 2-5 ) is used to single step a function in a step. When set to 1, it instructs the CLC to stop program execution after each step function. Toggling the Cycle_Start (bit 6 ) will continue execution through the next function.
5.6. Data and Expressions

5.6.1. Integers and Floats

Integers are signed or unsigned whole numbers, such as 5 or -3. Integers cannot have a fractional component. CLC integers are stored as a 32-bit "two's complement" value using the high bit for the sign. Integers, therefore, have a range of -2147483648 decimal (80000000 hex) to +2147483647 (7FFFFFFF).

Floating point values possess three components. A signed integer component (the "characteristic"), an unsigned decimal ("mantissa") component, and an exponential component (the "exponent") using the base 10. Even if you do not specify a decimal or exponential component, floats are stored in full floating point format. Floating point numbers are typically specified by using a decimal point. (i.e., "5" is an integer, "5.0" is floating point.)

The CLC stores integer and floating point variables as 32 bit double words to preserve word addressing on even memory locations. Thirty-two bit addressing also simplifies operations on the CLC’s I/O registers. All variables are 32-bit values; CLC I/O registers are 16 bit values.

5.6.2. Constants and Variables

CLC programs permit both constants and variables in user programs. These are useful in that they allow storage of information that is application specific, such as point locations, but changing them does not require modifications to associated motion programs.

The tasks and subroutines within a CLC program share a common storage area. Therefore, all tasks and subroutines access the same data. In addition, the CLC provides integer and floating point global system variables that may be used to share values with other system devices that can access global memory. (i.e., multiple CLC cards on a VME bus.)

A constant may be any valid integer or floating point literal constant (e.g., a user specified number, such as 3.14159). Constants cannot be modified by the program during program execution.
Integer variables are identified by an upper case ASCII character I immediately followed by a string of decimal digits or a symbolic name enclosed in square brackets, such as I[01] or I[15]. Integers are stored in an integer variable table in the CLC memory. Floating Point variables are identified by an upper case ASCII character F immediately followed by a string of decimal digits or a symbolic name enclosed within square brackets, as in F[01] or F[23]. Floating point variables are stored in a floating point variable table in the CLC memory.

The CLC compiler also accepts a special case for constant indices to the integer or floating point tables. In this case the square brackets are not required. The table entry is addressed by adding an integer constant immediately after the I or F variable identifier. Each integer or floating point constant or variable requires 4 bytes (32 bits) of CLC memory.

**Examples:**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Variable Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>I[15]</td>
<td>;integer variable, formal form</td>
</tr>
<tr>
<td>I37</td>
<td>;integer variable, short form (integer constant)</td>
</tr>
<tr>
<td>I[Ipointer1]</td>
<td>;symbolic integer variable</td>
</tr>
<tr>
<td>GI[15]</td>
<td>;global integer variable, formal form</td>
</tr>
<tr>
<td>GI37</td>
<td>;global integer variable, short form (integer constant)</td>
</tr>
<tr>
<td>GI[Ipointer1]</td>
<td>;global symbolic integer variable (Ipointer1 must evaluate to an integer)</td>
</tr>
<tr>
<td>F[01]</td>
<td>;floating point variable</td>
</tr>
<tr>
<td>F99</td>
<td>;floating point variable, short form</td>
</tr>
<tr>
<td>F[Fpointer3]</td>
<td>;symbolic floating point variable</td>
</tr>
<tr>
<td>GF[01]</td>
<td>;global floating point variable</td>
</tr>
<tr>
<td>GF99</td>
<td>;global floating point variable, short form</td>
</tr>
<tr>
<td>GF[Fpointer3]</td>
<td>;global symbolic float (Fpointer3 must evaluate to an integer)</td>
</tr>
</tbody>
</table>

**Global Variables**

The global variables consist of a contiguous array of 256 floating point and 256 integer variables. The global variables are accessed as arrays of 4 byte values beginning at specified base addresses. The specific addresses depend on the type of CLC card. Global variables reside in CLC RAM and are not retained during power-off. Global variables in user programs are not automatically initialized by the CLC compiler. It is the programmer’s responsibility to explicitly initialize global variables in a program. If values need to be saved, they may be copied to standard variables which are battery-backed, provided sufficient space has been allocated.
Because the variable tables are common areas, they can be read or written at any time by any
of the user tasks, subroutines or event functions. Therefore, the common data areas are used
to pass values between tasks or subroutines. Global variables may also be used to exchange
values between external components of a CLC system that are capable of accessing the global
memory area. However, using variables to communicate program values directly can have
unexpected results. Because the CLC’s multitasking can be event driven, systems cannot
guarantee the order of event function execution. Use caution when directly exchanging
program data by writing to the same variable from more than one task or subroutine.

Depending on the occurrence and number of pending events, you cannot always be sure of the
exact time that variables will be read or written. User programs may assign bits within one or
more variables as "wait" flags or "semaphores" controlling program branching, thereby
synchronizing critical portions of the otherwise independent tasks and programs.

5.6.3. Expressions

An expression is used to calculate or modify variables. Standard algebraic syntax is used to
write a mathematical expression within a CLC program. An expression may use any
combination of constants, variables, operators, functions, and parenthesis. The value to the
left of the "=" is set to the calculated value of all elements of the expression on the right. The
right side of the expression may include up to 16 constants, variables, parentheses, operators,
and functions.

Evaluation and Precedence
The right side of an expression is evaluated sequentially from left to right. There is no
operator precedence. If necessary, balanced parentheses should be used to insure that your
expression is evaluated in the order that you intend. For example:  RATIO = 4 + 3 * 5 - 1
evaluates RATIO as 4+3=7, then 7*5=35, then 35-1=34, while RATIO = (4 + 3) * (5 - 1)
evaluates RATIO to 28.

Type Conversion
If an expression uses both integer and floating data types, the CLC compiler converts all
operands to the floating data type before evaluating the expression. After evaluation, the result
is converted to the data type of the destination (left side of the expression).
### 5.6.4. Mathematical and Logical Operators

The operators available for use in expressions are listed in the following table.

<table>
<thead>
<tr>
<th>Arithmetic operators</th>
<th>Logical operators Note: Logical operations are limited to unsigned integers.</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>&amp; logical bitwise AND of two integers</td>
</tr>
<tr>
<td>+</td>
<td></td>
</tr>
<tr>
<td>−</td>
<td>^ logical bitwise &quot;exclusive or&quot; of two integers (101 ^ 110 is 011)</td>
</tr>
<tr>
<td>*</td>
<td>not logical bitwise inversion of an int (changes &quot;0s&quot; to &quot;1s&quot; and &quot;1s&quot; to &quot;0s&quot;)</td>
</tr>
<tr>
<td>/</td>
<td>&lt;&lt;=n Int shift &quot;n&quot; bits left (1-16 bits), low bits padded with 0, high bits are lost</td>
</tr>
<tr>
<td>*</td>
<td>&gt;&gt;=n Int shift &quot;n&quot; bits right (1-16 bits), high bits padded with 0, low bits are lost</td>
</tr>
<tr>
<td>Logical operators Note: Logical operations are limited to unsigned integers.</td>
<td></td>
</tr>
<tr>
<td>sin(n)</td>
<td>returns the floating point sine of an integer or float</td>
</tr>
<tr>
<td>cos(n)</td>
<td>returns the floating point cosine of an integer or float</td>
</tr>
<tr>
<td>tan(n)</td>
<td>returns the floating point tangent of an integer or float</td>
</tr>
<tr>
<td>arcsin(n)</td>
<td>returns the floating point arcsine of an integer or float</td>
</tr>
<tr>
<td>arccos(n)</td>
<td>returns the floating point arccosine of an integer or float</td>
</tr>
<tr>
<td>arctan(n)</td>
<td>returns the floating point arctangent of an integer or float</td>
</tr>
<tr>
<td>ln(n)</td>
<td>returns the floating point natural logarithm (base e) of an integer or float</td>
</tr>
<tr>
<td>log(n)</td>
<td>returns the floating point logarithm (base 10) of an integer or float</td>
</tr>
<tr>
<td>Exponential functions:</td>
<td></td>
</tr>
<tr>
<td>n**p</td>
<td>returns the floating point value of &quot;n&quot; raised to the &quot;p&quot; power</td>
</tr>
<tr>
<td>sqrt(n)</td>
<td>returns the floating point square root of an integer or float</td>
</tr>
<tr>
<td>Conversion functions:</td>
<td></td>
</tr>
<tr>
<td>%</td>
<td>modulus (the remainder or fractional portion) of the result of the division of 2 integers or floating point numbers</td>
</tr>
<tr>
<td>int(n)</td>
<td>returns the integer portion of a floating point value as an integer</td>
</tr>
<tr>
<td>float(n)</td>
<td>returns a floating point value equal to an integer</td>
</tr>
<tr>
<td>frac(n)</td>
<td>returns a floating point value equal to an integer</td>
</tr>
<tr>
<td>absolute(n)</td>
<td>converts a positive or negative integer or float to a positive integer</td>
</tr>
<tr>
<td>bintoBCD(n)</td>
<td>converts a binary value to a packed BCD integer</td>
</tr>
<tr>
<td>BCDtobin(n)</td>
<td>converts a packed BCD to a binary value</td>
</tr>
</tbody>
</table>
Expression Examples:

I03 = I03 + 2
Adds 2 to the current value of the integer variable I03 and replaces the current value of I03.

F01 = 3.14
Replaces the current value of floating point variable F01 with the constant 3.14.

F33 = COS(F33)
Replaces F33 with the cosine of F33

F15 = (F10 + F11)*(F12 - F13)
Replaces the current value of F15 with the result of the multiplication of the value of F10 plus F11, times the value of F13 subtracted from F12.

5.7. Tables

The CLC stores program variable information and data in separate areas of memory called tables, also referred to as arrays.

CLC integer and floating point variables are maintained as one dimensional arrays. You can think of a one dimensional array as a simple list, or as a spreadsheet that contains series of rows with only one column. These arrays are referred to as the integer and floating point tables. Each entry requires 4 bytes of CLC memory.

As previously described, a variable table entry is addressed by using an "I" or "F" prefix before an integer value that is within the range of the respective table. For example, I[15] accesses the 15th entry of the integer table, or F[43] accesses floating point table entry 43. This is “direct” addressing of a variable. The integer may be thought of as an "index" into the table, with the base address of the table being the first entry, i.e. the integer "1".
Indirect Addressing
A calculation instruction may also access a variable or table entry indirectly by enclosing an integer variable within square brackets. This format allows you to use programmed calculations to dynamically control the array index. As a result, the program can use an I/O input value or calculation result to determine which variable or table entry is used. For example, F[I[3]] (or F[I3]) returns the value of the floating point variable table entry addressed (or "indexed") by the contents of the third integer table entry, I[3]. By writing a program that changes the contents of I[3] you can obtain a value from a table of constants for further calculation.

In addition to the single dimensioned arrays used for the variable tables, the CLC also uses several two dimensional arrays. The absolute and relative point tables are two dimensional arrays that contain information about a coordinated motion path. The event table which contains information and parameters for the event functions is also two dimensional.

A row of the two dimensional tables, absolute and relative point, and event, may be accessed using an index enclosed within square brackets "[ ]" in the same manner as the single dimension variables. Accessing a two dimensional table element uses the dot operator ("." ) to select a column element within the selected row.

An element of the absolute or relative point tables, or the event table is accessed using an ABS, REL, or EVT prefix to a row index enclosed within square brackets, immediately followed by the array dot operator ("." ) and a row element identifier.

For example: \texttt{ABS[n].x REL[n].x EVT[n].r}, where ABS or REL or EVT accesses the absolute point, or relative point, or event table, "n" is a valid integer constant or variable or equivalent label, and "x" is a table element identifier. Blank spaces are not allowed.

The integer value contained within the brackets may be thought of as an index to one of the rows of the table, while the letter following the dot operator indexes to the column position within that row. You may use a symbolic name or integer variable within the square brackets to access a table row, provided that the name evaluates to an integer value equal to a valid table entry.

For example: \texttt{ABS[I[n]].x EVT[event3].x}
5.7.1. Event Table

Events in a CLC program are stored in the CLC Event Table:

<table>
<thead>
<tr>
<th>Event #</th>
<th>s</th>
<th>t</th>
<th>d</th>
<th>a</th>
<th>f</th>
<th>m</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>Active</td>
<td>Time</td>
<td>Start</td>
<td>10</td>
<td>In_Zone</td>
<td>Message 00</td>
</tr>
<tr>
<td>02</td>
<td>Inactive</td>
<td>Distance</td>
<td>End</td>
<td>10</td>
<td>Gripper_On</td>
<td>Message 01</td>
</tr>
<tr>
<td>03</td>
<td>Active</td>
<td>Distance</td>
<td>Start</td>
<td>50</td>
<td>Gripper_On</td>
<td>Message 02</td>
</tr>
<tr>
<td>04</td>
<td>Inactive</td>
<td>Distance</td>
<td>End</td>
<td>25</td>
<td>Gripper_Off</td>
<td>Message 03</td>
</tr>
<tr>
<td>05</td>
<td>Undefined</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td></td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>nnn</td>
<td>Undefined</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The elements of the Event Table are defined in the following table:

<table>
<thead>
<tr>
<th>s</th>
<th>status</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>inactive</td>
</tr>
<tr>
<td>1</td>
<td>queued</td>
</tr>
<tr>
<td>2</td>
<td>pending</td>
</tr>
<tr>
<td>3</td>
<td>executing</td>
</tr>
<tr>
<td>4</td>
<td>done</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>t</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>event inactive</td>
</tr>
<tr>
<td>1</td>
<td>repeating timer</td>
</tr>
<tr>
<td>2</td>
<td>time on coordinated motion path</td>
</tr>
<tr>
<td>3</td>
<td>distance on coordinated motion path</td>
</tr>
<tr>
<td>4</td>
<td>single axis distance</td>
</tr>
<tr>
<td>5</td>
<td>repeating axis position</td>
</tr>
<tr>
<td>6</td>
<td>task external interrupt input</td>
</tr>
<tr>
<td>7</td>
<td>VME broadcast interrupt</td>
</tr>
<tr>
<td>8</td>
<td>VME short address (mailbox) interrupt</td>
</tr>
<tr>
<td>9</td>
<td>axis feedback capture</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>d</th>
<th>reference direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>start of segment</td>
</tr>
<tr>
<td>1</td>
<td>end of segment</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>a</th>
<th>argument</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a numeric value</td>
</tr>
<tr>
<td></td>
<td>(in milliseconds if timed event)</td>
</tr>
<tr>
<td></td>
<td>(a % of the segment distance if coordinated motion)</td>
</tr>
<tr>
<td></td>
<td>(degrees if repeating axis position event)</td>
</tr>
<tr>
<td></td>
<td>(contains the probe position read from the drive if feedback capture event is selected)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>f</th>
<th>event function mark</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Name of subroutine to be called when event is triggered</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>m</th>
<th>message</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ASCII text string for a descriptive message</td>
</tr>
</tbody>
</table>
5.7.2. CLC Path Planner

The CLC’s path planner interprets coordinated motion commands for all user program tasks and associated events. Using the program specified points in the absolute and relative point tables, the path planner continually calculates the position information that must be sent to each drive to maintain coordination between the multiple axes. The path planner is one of the highest priority CLC executive tasks.

The path planner continuously monitors the difference between the current system position and the planned or target position that will result from the user program. The distance that the path planner will look ahead of the current position is set by the CLC’s look ahead distance parameter, T’s .23.

Geometry Segments

The CLC defines multi-axis coordinated motion in terms of a path composed of standard straight line and circular geometry segments. Two points define a line, three points define a circle. The path combines these standard geometry segments so that the start of the next segment begins at the end of the previous segment.

The CLC always calculates a motion path using a series of points taken from the Absolute or Relative Point tables. The absolute or relative table entry that is used for the end point of a segment contains the rate profile parameters (speed, acceleration, deceleration and jerk) used by the path planner to construct the rate profile for that segment. Starting or middle points are not used to construct the profile, only the table point entry rate profile parameters for the end or target point of the segment are used.

Blend Segments

A blend segment is a special type of segment calculated by the CLC path planner and used to join two standard geometry segments. Blend segments are not directly accessible to the programmer. However, the programmer does specify the blend radii that the path planner uses to compute the blend segments.

Blend segments provide the capability of continuous smooth motion from one standard segment to another without stopping. They reduce the path planner’s cycle time as well as provide a means of optimal path shaping.

Speed

The point table entry for speed defines the velocity of motion through the segment as a percentage of the maximum velocity for the axis. If a blend radius is specified, the speed entry effects only the portion of the segment outside the blend areas.
Acceleration/Deceleration
CLC acceleration and deceleration for a single non-coordinated axis is set as an acceleration constant, in units per second per second, using the AXIS/INITIALIZE command. The deceleration for the axis is the inverse of the specified acceleration. Single axis accel/decel rates are developed within the intelligent DDS drive itself and require minimal processing by the CLC.

Acceleration and deceleration for multi-axis coordinated motion are set in corresponding entries in the absolute or relative point tables as a percentage of the task's maximum rates. The maximum rates are established through user set parameter values.

Jerk Limiting
The CLC provides two forms of jerk limiting. The simplest, used with a single axis, is 'S' curve limiting and is generated within the intelligent DDS drive itself. More sophisticated jerk limiting is provided by the CLC path planner for multiple coordinated axes.

Jerk limiting should only be used for start and stop movements. The jerk limit should be set to zero when blending one segment into another.

The jerk limit affects rate limiting. For example, with jerk limiting set to 100% (total s-shaping), acceleration and deceleration limiting occur at 50% of the point table entries. For optimum performance, the jerk limit should be set to a minimum so that there is no acceleration or deceleration limiting.
5.7.3. Absolute Point Table

Absolute points, used to define geometric path segments for coordinated motion, are stored in an absolute point table separate from the executable program code. The absolute point table is also saved in a point file on the Host PC. Points may be defined in motion programs, or directly entered in the point table if the program resides in the CLC.

Example Absolute Point Table:

<table>
<thead>
<tr>
<th>Point</th>
<th>X</th>
<th>Y</th>
<th>Z</th>
<th>Blend</th>
<th>Speed</th>
<th>Accel</th>
<th>Decel</th>
<th>Jerk</th>
<th>E1</th>
<th>E2</th>
<th>E3</th>
<th>E4</th>
<th>r</th>
<th>p</th>
<th>w</th>
<th>e</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>99.0</td>
<td>-333.2</td>
<td>95.5</td>
<td>10</td>
<td>20</td>
<td>100</td>
<td>10</td>
<td>90</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>30</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>002</td>
<td>2.0</td>
<td>33.2</td>
<td>-60.5</td>
<td>5</td>
<td>67</td>
<td>50</td>
<td>10</td>
<td>80</td>
<td>1</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>60</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>003</td>
<td>2.0</td>
<td>33.2</td>
<td>-60.5</td>
<td>1</td>
<td>8</td>
<td>60</td>
<td>95</td>
<td>80</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>90</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>nnn-2</td>
<td>0.0</td>
<td>5.9</td>
<td>95.5</td>
<td>5</td>
<td>10</td>
<td>20</td>
<td>30</td>
<td>70</td>
<td>5</td>
<td>6</td>
<td>8</td>
<td>9</td>
<td>30</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>nnn-1</td>
<td>00.0</td>
<td>-333.2</td>
<td>95.5</td>
<td>10</td>
<td>40</td>
<td>99</td>
<td>40</td>
<td>50</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>15</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>nnn</td>
<td>00.0</td>
<td>-333.2</td>
<td>95.5</td>
<td>10</td>
<td>9</td>
<td>99</td>
<td>30</td>
<td>55</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

The elements of the Absolute Point Table are defined in the following table:

<table>
<thead>
<tr>
<th>Subclass</th>
<th>Description</th>
<th>Type</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>x cartesian coordinate</td>
<td>Floating</td>
<td>32-bit</td>
</tr>
<tr>
<td>y</td>
<td>y cartesian coordinate</td>
<td>Floating</td>
<td>32-bit</td>
</tr>
<tr>
<td>z</td>
<td>z cartesian coordinate</td>
<td>Floating</td>
<td>32-bit</td>
</tr>
<tr>
<td>b</td>
<td>blend radius</td>
<td>Floating</td>
<td>32-bit</td>
</tr>
<tr>
<td>s</td>
<td>% of maximum speed</td>
<td>Integer</td>
<td>8-bit</td>
</tr>
<tr>
<td>a</td>
<td>% of maximum acceleration</td>
<td>Integer</td>
<td>8-bit</td>
</tr>
<tr>
<td>d</td>
<td>% of maximum deceleration</td>
<td>Integer</td>
<td>8-bit</td>
</tr>
<tr>
<td>j</td>
<td>jerk limiting %</td>
<td>Integer</td>
<td>8-bit</td>
</tr>
<tr>
<td>E1-E4</td>
<td>event ID number</td>
<td>Integer</td>
<td>16-bit</td>
</tr>
<tr>
<td>r</td>
<td>degrees of roll</td>
<td>Floating</td>
<td>32-bit</td>
</tr>
<tr>
<td>p</td>
<td>degrees of pitch</td>
<td>Floating</td>
<td>32-bit</td>
</tr>
<tr>
<td>w</td>
<td>degrees of yaw</td>
<td>Floating</td>
<td>32-bit</td>
</tr>
<tr>
<td>e</td>
<td>elbow state</td>
<td>Integer</td>
<td>8-bit</td>
</tr>
</tbody>
</table>
5.7.4. Relative Point Table

Relative points, used to define geometric path segments for coordinated motion, are stored in a relative point table separate from the CLC user program. The relative point table is also saved in its own file on the personal computer (PC) Host development system.

During the processing of relative moves, events and the velocity profile parameters associated with the relative point being processed are active. The events and the velocity profile parameters associated with the absolute or reference point or origin point are not used. The events from the relative table are triggered; events from the absolute point table are ignored.

**Example Relative Point Table:**

<table>
<thead>
<tr>
<th>Point</th>
<th>X</th>
<th>Y</th>
<th>Z</th>
<th>Blend</th>
<th>Speed</th>
<th>Accel</th>
<th>Decel</th>
<th>Jerk</th>
<th>E1</th>
<th>E2</th>
<th>E3</th>
<th>E4</th>
<th>r</th>
<th>p</th>
<th>w</th>
<th>e</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>99.0</td>
<td>-333.2</td>
<td>95.5</td>
<td>5</td>
<td>20</td>
<td>100</td>
<td>1</td>
<td>90</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>30</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>002</td>
<td>2.0</td>
<td>33.2</td>
<td>-60.5</td>
<td>20</td>
<td>70</td>
<td>50</td>
<td>10</td>
<td>80</td>
<td>1</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>60</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>003</td>
<td>2.0</td>
<td>33.2</td>
<td>-60.5</td>
<td>50</td>
<td>80</td>
<td>60</td>
<td>20</td>
<td>80</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>90</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>nnm-2</td>
<td>0.0</td>
<td>5.9</td>
<td>95.5</td>
<td>50</td>
<td>10</td>
<td>20</td>
<td>30</td>
<td>70</td>
<td>5</td>
<td>6</td>
<td>8</td>
<td>9</td>
<td>30</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>nnm-1</td>
<td>00.0</td>
<td>-333.2</td>
<td>95.5</td>
<td>30</td>
<td>40</td>
<td>10</td>
<td>40</td>
<td>50</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>15</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>nnn</td>
<td>00.0</td>
<td>-333.2</td>
<td>95.5</td>
<td>5</td>
<td>55</td>
<td>10</td>
<td>50</td>
<td>55</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

The elements of the relative point table are defined in the same way as those of the absolute point table.
5.7.5. Zone Protection Table

The CLC’s zone tables provide a method of detecting and preventing motion through one or more three-dimensional rectangular volumes. Protective zones are typically used to prevent motion that would otherwise be attempted through supporting columns of a robot, or would damage clamps or fixtures. Zones are also used to prevent motion through spaces near a machine that may be occupied by an operator or other personnel. Zones are active during jogging as well as coordinated motion.

A prohibited-zone volume is described by specifying pairs of three dimensional coordinates that locate diagonally opposite corners of the volume. Each table entry consists of a status bit and the two sets of x, y and z coordinates. The status bit permits independently activating/deactivating individual zones within the table. Each of the four tasks (A, B, C and D) within a user program using zones maintains a separate zone table for the task. Memory for the task’s zone table must be allocated by a Text Language DATA/SIZE instruction, or Visual Motion’s SIZE icon, within the task.

Zone Detection
Detection of an instruction that may violate a zone is accomplished in the path planner, permitting zone violations to be detected before motion starts. However, detection is related to the user specified look-ahead distance.

For circular motion, the radius is used to determine zone interference. The total length of the circle is computed and interference calculations are taken at exact intervals along the circumference of the circle.

Joint moves are tested using a spherical volume instead of the rectangular volume. The prohibited volume is a sphere enclosing the rectangular area specified by the zone table coordinate pairs. In other words, the zone volume is a sphere with a radius equal to the distance from the center of the rectangular volume to a corner of the rectangle.

Zone interference detected by the path planner stop motion while maintaining the system position on the path.
CHAPTER 6. ICON PROGRAMMING

6.1. Introduction

This chapter describes how to use icons and other VisualMotion commands to create motion programs, a process briefly illustrated in the flow diagram at right.

The first section describes how icons are placed and connected in motion control programs and the second section contains descriptions of each icon, presented in alphabetical order.

6.1.1. Working with VisualMotion’s Pointer and Icon Palettes

VisualMotion requires a mouse with right and left push-buttons. (Although some mice have three buttons, VisualMotion ignores the center button). The left button is used for selecting single icons or lines, selecting commands from the pull-down menus, and positioning the cursor in dialog boxes for keyboard entry. In VisualMotion, the right mouse button is used for improved cut and paste operations. An area larger than a single icon is selected by clicking and holding the right mouse button, then dragging the gray-line rectangle to enclose the desired area of the VisualMotion workspace.

The VisualMotion window displays a palette of programming icons. Four standard palettes are provided for single, coordinated, ELS and utility icons. You may build a custom palette of up to twenty icons. Palettes may be selected or built by selecting the appropriate item from the Options menu.

An icon in the palette is selected for placement by clicking it once with the left mouse button. The selected icon is placed on the VisualMotion workspace by positioning the cursor where you want the icon to appear and clicking once. A different icon may be selected for placement simply by clicking a new icon in the palette. The current icon may be de-selected, freeing the workspace cursor, by re-clicking on the selected palette icon.

Every icon used in a program may be given a descriptive label of up to 80 characters which then appears in the comment window below the VisualMotion workspace as the cursor passes over the icon. The label can be entered using the Label button of the icon’s setup screen.
6.1.2. Connecting Icons

After you have placed a number of icons on the workspace, they must be connected to indicate the program flow. Most icons have a maximum of three possible inputs and one output. The exception is the Branch icon which has two outputs.

To draw a line, select the Line icon from the palette. Position the cursor on the first icon that you wish to connect and click. A rectangle appears, surrounding the icon. Move the cursor to the destination icon where the line is to end and click again. Visual Motion automatically draws a line from the first to the second icon, using square corners where appropriate. Arrows on the line indicate the direction of program execution. You may continue this process, connecting icon to icon, without re-selecting the Line icon.

You may wish to manually route an interconnect to provide room for additional icons at a later time. And, under some circumstances, the Line icon's auto-routing may fail to route an interconnecting line, displaying the "Try connecting adjacent lines." dialog box. Lines may be drawn manually by sequentially clicking on adjacent squares on the invisible workspace grid. A manually placed line may not cross another line, attempting to do so displays an error box.

A line connecting two icons may be deleted by using the Scissor icon from the Utility palette. Simply select the Scissor icon, position the Scissor over the line to be deleted and press the left mouse button.
6.2. CLC Icons

The CLC icons are listed in alphabetical order on the following pages. Each icon is listed by name with an example of the icon, its associated initial dialog boxes, and a description of usage with required selections and/or data entry.

Accel ..............  Decel ..............  Join ..............
Axis ..............  ELS ..............  Joint ..............
AxisEvt .........  ELSAdj .............
Branch .............  ELSMode .....  Line ..............
Calc .............  Event .............
Cam .............  Finish .............
CamAdj .............
CamBuild .......
Circle .............  I/O .............
Go .............
Home .............  Path .............
Path .............  PLS .............
Position .............
6.2.1. Accel

The Accel icon is used to set or change the acceleration of a single non-coordinated axis.

**Axis** indicates the axis to accelerate. The axis may be entered as an integer constant, integer variable, global integer variable or an equivalent label.

**Rate** specifies the acceleration rate in units per second per second. The entry may be a floating point constant, variable, global variable or an equivalent label.

The User Defined Labels pop-up dialog box is accessible from both data entry boxes.
6.2.2. Axis

The Axis icon is used to specify drives that are assigned to the current task. The Axis icon can only be used to assign a drive within the four Tasks, it should not be used in a subroutine or event function.

Motion Type provides a pop-down menu of the axis modes available for this occurrence of the axis icon. Only one motion type is allowed for each occurrence of a single icon. Choosing coordinated motion also enables the Kinematic Number data entry box permitting entry of a custom kinematic library routine number.

The drive list displays setup information for each assigned drive. The type of information depends upon the selected Motion Type. The list must be horizontally scrolled in order to display all the available setup data.

The Add, Edit and Delete buttons are used respectively to add a drive, modify axis values, and delete an assigned drive. You may also edit a listed drive by double-clicking the list entry. After editing, the dialog box is closed by clicking the Cancel button; all list items are saved as displayed.
In Single Axis the drive list may be scrolled to display:

<table>
<thead>
<tr>
<th>Drive #</th>
<th>Trigger 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acceleration</td>
<td>Trigger 1 Event</td>
</tr>
<tr>
<td>Velocity</td>
<td>Trigger 2</td>
</tr>
<tr>
<td>Halted?</td>
<td>Trigger 2 Event</td>
</tr>
<tr>
<td>Mode</td>
<td></td>
</tr>
</tbody>
</table>

In Velocity Mode the drive list may be scrolled to display:

<table>
<thead>
<tr>
<th>Drive #</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acceleration</td>
<td>Trigger 1</td>
</tr>
<tr>
<td>Velocity</td>
<td>Trigger 1 Event</td>
</tr>
<tr>
<td>Halted?</td>
<td>Trigger 2</td>
</tr>
<tr>
<td>Mode</td>
<td>Trigger 2 Event</td>
</tr>
</tbody>
</table>

In Ratioed Axes the drive list may be scrolled to display:

<table>
<thead>
<tr>
<th>Drive #</th>
<th>Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Master #</td>
<td>Trigger 1</td>
</tr>
<tr>
<td>Slave Ratio</td>
<td>Trigger 1 Event</td>
</tr>
<tr>
<td>Master Ratio</td>
<td>Trigger 2</td>
</tr>
<tr>
<td>Halted</td>
<td>Trigger 2 Event</td>
</tr>
</tbody>
</table>

In Torque Mode, the list displays only the drive number. In Coordinated mode, the list displays the drive number and its designation as an x-, y-, z-, roll, pitch or yaw axis.

A pop-up User Defined Labels dialog box is available for the data entry boxes by placing the focus ("I-beam" cursor) in an entry box, then double clicking the left mouse button on an empty area outside of the field.
Selecting **Single Axis** or **Velocity Mode** motion type in the Task Axes Setup dialog box displays another dialog box requiring the entry of an axis to be assigned to the drive. The axis may be entered as an integer constant or equivalent label.

![Single Axis Setup](image)

You may enable and enter an **initial acceleration** and **velocity** as floating point constants or variables or an equivalent label. If the initial accel and velocity are not defined they must be subsequently specified in the program using the Accel and Veloc icons. By default, the Axis **Initially Halted** option is checked. This disables the axis at the start of the task; it can be enabled using a Go icon later in the task. Clicking to uncheck the Axis Initially Halted option enables the axis to go at the start of a task. Both Single and Velocity Mode axes may be assigned linear (units) or rotary (degrees) positioning, and may use the DDS-2 probe capability to trigger events.

Selecting Velocity Mode displays a similar dialog box, however, **Velocity Change** is enabled, permitting selection of Ramp instead of the default Step velocity changes. Ramp velocity changes are based on current acceleration rate, whereas Step changes are made at the maximum rate allowed by the drive and motor.
Selecting **Coordinated** motion type in the Task Axes Setup dialog box and clicking the Add button opens a Coordinated Setup dialog box. Axes are assigned by entering an integer or equivalent label for each axis and clicking the OK button. The Coordinated Setup dialog box remains open until it is explicitly closed using the Cancel button. Although you may assign up to 99 axes using one coordinated motion Axis icon, this dialog box is designed to prevent you from assigning multiple axis numbers or labels to a single axis. A pop-up list of labels is available from the data entry box.

![Coordinated Axes](image)

Selecting **Torque Mode** allows entering axes in the same manner as coordinated mode; however, any number of axes may be assigned, up to the maximum number of axes in the system.

![Torque Mode Setup](image)
**Ratioed Axes** mode is used to slave one axis to a master (or controlling) axis. The axes may be assigned an *initial ratio* that determines the number of revolutions of the slave to revolutions of the master. If an initial ratio is not defined, this ratio must be set using a Ratio icon. A Ratio icon may also be used in the program to change the ratio. More than one axis may be slaved to a single master.

A ratioed may be assigned linear or rotary positioning, and may use the DDS-2 probe capability to trigger events.

In CLC Ratio Mode, the slave axis now follows the command position of the master axis if the master is a coordinated motion axis. If the master is any other type of axis, the slave can follow an internal or external secondary encoder feedback position. Previously, the feedback velocity was followed if the axis primary feedback was used. The axis mode can be switched to single-axis or velocity mode from within the program or for manual mode jogging. A step rate was added for gradual adjustment of the ratio.

**NOTE:** To keep the slave axis synchronized when the master axis is jogging, set Axis Control register bit 4 (Synchronized Jog bit). If the slave axis will never be switched to single-axis mode, this bit should always be mapped high. Ratio Mode Enhancements
6.2.3. AxisEvt

The AxisEvt icon is used to enable up to four rotary axis events for a specified axis. A maximum of four events may be enabled by any combination of AxisEvt or Move icons.

The Axis and Event numbers are specified using integer constants, variables, or global variables or equivalent labels. A pop-up User Defined Labels dialog box is available for the data entry boxes by placing the focus ("I" beam cursor) in an entry box, then clicking the cursor on an empty area of the dialog box.

For an event enable to have an effect, the event function must already be entered and setup as a type 5 (Rotary - Repeating Axis Position) event.

**NOTE:** Rotary events cannot be attached to an axis that has a PLS (Programmable Limit Switch) attached to it.
6.2.4. Branch

A Branch icon re-directs the program flow depending upon a true/false logical value.

Selecting an I/O type branch permits testing a bit in a specified I/O register for a logical (true/false) condition. The I/O register for comparison is specified by an integer or an equivalent label. A bit mask permits testing of only selected bits within the specified register to be tested. The bit mask may be entered as an integer or equivalent label. I/O tests are true/false only, the equality or inequality relationships are not allowed.

Selecting Variable permits a comparison of the contents of an integer variable, floating point variable or an equivalent label with a compared value (integer or floating point constant, or an equivalent label). User defined labels are accessible through a pop-up dialog box.

Selecting CLC or Drive CAM permits a specified CAM check for one of three conditions. The branch will return a yes or no value if there is No CAM or if the Cam is Ready or Running.

Four radio buttons at the bottom of the dialog box permit selection of one of four Branch icon graphics, each with a different positioning of the branch test outputs.
### 6.2.5. Calc

The Calc icon can be used to initialize points and zones, and setup events. By using Calc icons in this way, points, zones and events reside within the programs, thereby eliminating the need to specify them after downloading the program to a CLC. The **Resultant** box specifies the element of the point, event or zone tables, or an equivalent label. The **Equation** box contains the expression to be evaluated.

**Example:** Absolute point definition \( \text{ABS}[1] = \{10,10,0,0,50,50,50,90,2,0,0,0,0,0,0,0\} \)

**Example:** Distance-based event setup \( \text{EVT}[1] = \{1,3,0,95,\text{vac\_on}, \text{executing event}\} \)

#### Calculations

<table>
<thead>
<tr>
<th>Resultant:</th>
<th>Equation:</th>
</tr>
</thead>
</table>

**Operators:**

+  
-  
*  
/  

NOTE: The Operators menu includes type conversion operators for floating and integer data types. These operators should be placed immediately to the left of the operand (i.e., as the equation is parsed from left to right, the compiler first obtains the value, then performs the conversion.).

For descriptions of point, zone and event definitions, refer to the information on point, zone and event tables in the chapter, “Programming Concepts.”

**Calculations**

The Calc icon also permits placing a calculation into the CLC program flow. The **Resultant** box specifies destination of the calculation and must be a variable, either integer or floating point, or an equivalent label. A pop-up User Defined Labels dialog box is available. The **Equation** box contains the expression to be evaluated. Operators may be selected from the Operators scrolling menu or entered using the keyboard. Symbolic labels that equate to a constant may be used in equations. The equation is evaluated from left to right. Parenthesis can be used to control the order of evaluation.

NOTE: Although an equation may use a combination of floating point, integer variables and constants, be sure that the data types used for a single operation match. Data type mismatches are especially easy to overlook when using symbolic label names that do not implicitly identify the data type. Use the "float()" and "int()" conversion operators to force the proper data types.
6.2.6. Cam

The Cam icon is used to associate a cam to a slave axis and to supply coefficients for using the cam. To work properly, the axis must be configured as an ELS slave and synchronized to the master. The Cam can be drive or Card resident. For more information see Appendix C. Cam Applications.

Axis: Specifies which axis is associated with the cam. The entry may be an integer, global integer variable (GIx), program integer variable (Ix) or an equivalent label.

Cam Number: Specifies which Cam table to use. The entry may be an integer, global integer variable (GIx), program integer variable (Ix) or an equivalent label (range 1-8).

Coefficients: M, N, H and L may be entered as floats, global float variables (GFx), program float variables (Fx), or equivalent labels.

Cams execute every SERCOS cycle and are governed by the following equation:

\[ \text{Scmd} = H \cdot \text{CAM}[(M/N) \cdot \text{Mcmd} + \text{Mph}] + L \cdot \text{Mcmd} + \text{Sph} \]

Where,

- \text{Mcmd} \quad \text{Position command of the master (degrees)}
- M \quad \text{Multiplying scaling term of the master position}
- N \quad \text{Divisional scaling term of the master position}
- \text{Mph} \quad \text{Master phase adjust (degrees)}
- \text{CAM}[\] \quad \text{Normalized 1024 point cam table}
- H \quad \text{Cam scaling term (stretch factor)}
- L \quad \text{Linear scaling term of master position}
- \text{Sph} \quad \text{Slave phase adjust (degrees)}
- \text{Scmd} \quad \text{Commanded position to slave}
6.2.7. Cam Build

The CLC can build a cam on-line based on a set of input positions. Program instructions use the CLC’s ABS point table and an internal utility to build an internal CLC Cam which is stored on the Card or Drive as the Cam number indicated. For more information see Appendix C. Cam Applications.

**Cam Number** - indicates the cam to use. The entry may be an integer, global integer variable (GIx), program integer variable (Ix) or an equivalent label (range 1 to 8). The cam cannot be in use when this icon is executed.

**Drive Number** - indicates which drive the Cam is stored in. The entry may be an integer, global integer variable (GIx), program integer variable (Ix) or an equivalent label.

**Cam build type (PCAM, VCAM, ACAM or Spline)** - The X elements of the point table are the master positions, and the Y elements are the corresponding slave positions. The PCAM build type accepts input in the form of a table of target positions. The VCAM build type accepts a velocity profile as input and outputs a normalized profile. ACAM build type accepts an acceleration profile as input and outputs a normalized profile. When Spline is selected the CLC builds the cam by connecting the points with third order splines. A minimum of 5 and maximum of 200 user defined points are allowed.

**ABS Starting Point** - is the starting ABS point number used for cam generation. The entry may be an integer, global integer variable (GIx), program integer variable (Ix), or an equivalent label. The number must be in the range of points defined in the Size icon.

**ABS Ending Point** - is the ending ABS point number used for cam generation. The entry may be an integer, global integer variable (GIx), program integer variable (Ix), or an equivalent label. The number must be in the range of points defined in the Size icon.
The ABS point elements may be changed from within the program using CALC statements. Changes in the point table do not affect the cam until the CAM BUILD icon is executed. It is necessary to size the point table at compile-time to allow enough points for the profiles that will be needed. The x value of the first point must be 0, and the x value of the last point must be 360.

The cam generation may take one second or longer. The ‘wait’ checkbox can be cleared to exit this icon immediately and keep executing instructions while the cam is being built. The branch icon can be used to check if a cam is ready for activation. If the ‘wait’ checkbox is checked, the program flow will be stopped in this icon until the cam is ready for activation.

The CAM BUILD icon can be used to store a cam to an inactive location on the CLC. After the cam has been built, the CAM icon can select it for an axis.

Because the cam is stored in nonvolatile memory on the CLC, it is not necessary to execute this command each time through the program. A flag variable can be set and checked the next time through the program to avoid long delays when starting the program. For on-line changes, a register bit or variable should be checked each time through the program loop to avoid continually generating the cam, which consumes CLC resources and can slow down the program.

Cams enter the CLC as text files in CSV format, the kind most spreadsheets generate.

If the CSV file contains less than 1024 points, an algorithm within the CLC fills in the missing points. As a rule, a CSV cam file should contain at least 200 points, anything less than that does not sufficiently define the cam and unexpected results may occur.

**Errors at runtime**

1. The selected cam is currently active for any axis.
2. The point range exceeds the bounds of the point table.
3. Less than two points are defined.
4. The cam number is not valid (out of range or drive is not configured).
5. An error occurred when sending the cam to the drive.
6. When using PCAM option, and the first x position isn’t 0 and the last x position isn’t 360.
7. The x position exceeds the modulo of the master.
6.2.8. Cam Phase Adjust

This icon selects which phase adjust to perform and starts the phase adjust. There are two phase offset values for a cam axis: a master phase adjust, and a slave phase adjust. The master phase adjust shifts the position in the cam table relative to the master position. The slave phase adjust shifts the position of the slave axis. Since it is not related to the shape of the cam, the slave phase adjust is not multiplied by any of the cam factors.

The Master Phase adjust functions the same as the Fine Adjust selection in the ELS Axis Configuration Edit dialog box.

**Type** selects the method of adjustment:
- If **absolute**, the *degrees or percent* edit field is the new offset.
- If **incremental**, the *degrees or percent* edit field is added to the current offset. If in phase mode and sum exceeds 360, it rolls over. If in velocity mode the sum is limited to -100 or +300 percent.
- If **continuous +** or **continuous -** (phase sync mode only), a velocity dependent amount is added to the degree offset.

The CLC can perform only one phase adjust at a time. If two different phase adjust icons are used the second one will have no effect until the previous phase adjust is complete. Bit 4 in the axis status register is set to (0) when a phase offset is in progress, and (1) if the phase offset is complete.

**Type:** Absolute
Incremental (relative)
Continuous plus
Continuous minus

**Mode:** Master phase adjust (Mph in equation)
Slave phase adjust (Sph in equation)

**Axis:** Axis number

**Deg/Per:** Phase adjust target value

---

**CAM Phase Adjust**

- **Type**: Absolute, Incremental, Continuous +, Continuous -
- **Mode**: Master Phase Adjust, Slave Phase Adjust
- **Axis**: integer, Glx, Ix, or label
- **Deg/Per**: float, BFx, or label

---

The CLC can perform only one phase adjust at a time. If two different phase adjust icons are used the second one will have no effect until the previous phase adjust is complete. Bit 4 in the axis status register is set to (0) when a phase offset is in progress, and (1) if the phase offset is complete.
### 6.2.9. Circle

The Circle icon is used for multi-axis coordinated circular interpolation, moving in a circular arc from one point in 3D space to a second point in 3D space. The plane of the arc is two dimensional and the plane may have any orientation in three dimensional space. Placing a Circle icon on a Task or Subroutine workspace opens a Coordinated Circle Setup dialog box.

A circular move may be Absolute or Relative and is defined by three points on the circle. These points must have been previously defined in the absolute and/or relative point tables. Points are identified by a point table ID number or equivalent label.

An absolute circular move begins from the endpoint of the previous segment (or current position if the system is halted), moves through an absolute point, and terminates at another absolute point.

A relative circular move is similar; however, the intermediate and endpoints are defined as relative offsets from the starting point. The relative circular move begins at the endpoint of the previous segment (or current position if the system is halted), moves through the first relative offset, and terminates at the second relative offset.
6.2.10. Decel

The Decel icon is used to specify or change the deceleration rate for the CLC's Electronic Line Shaft (ELS) mode virtual axis (Axis 0) or an axis in velocity mode. The **Rate** may be specified by a floating point constant, variable, global variable or an equivalent label. A pop-up User Defined Labels dialog box is available for the data entry boxes by placing the focus ("I" beam cursor) in an entry box, then clicking the cursor on an empty area of the dialog box.
6.2.11. ELS

The ELS icon is used in place of the Axis icon to associate and configure slave axes in a CLC system that uses the ELS (Electronic Line Shaft) capability.

Virtual Master uses the CLC's ELS virtual axis (axis 0) as the master axis to provide master position data. The Virtual Master sends this position to all drives; therefore all drives on the CLC SERCOS communication loop have equal synchronization delays. Drives are synchronized within the single SERCOS cycle to axis 0, the virtual axis.

Daisy-Chained Real Master configures DDS-2 drive systems that use DDF cards. The DDF cards are interconnected using an independent cabling system and are synchronized to the drive containing the master card. Although the DDF cards provide the synchronization between drives, DDF system drives must still be configured using CLC Axis Configuration.

Real Master through SERCOS reads position data from the master drive's external encoder, then sends the position data to the slaves through the SERCOS communication system. Slaves are synchronized, but delayed one SERCOS cycle to read the master, then one cycle to transmit to the slaves. The master axis must be defined and configured using an Axis icon.

Follow Axis Feedback reads position data from an encoder through SERCOS communication, then sends position information to the slave through SERCOS. Slaves are synchronized but may be up to two SERCOS cycles delayed from the master. Encoder Used permits selection of the
The drive list displays the slave axis:

- **Mode** - velocity, phase or cam
- **Direction** - positive or negative with respect to the master
- **Master Ratio** - an integer value for the master used to compute the master/slave ratio
- **Slave Ratio** - integer value for the slave used to compute the master/slave ratio
- **Adjust** - the percentage difference used to compute the slave velocity or position

Clicking Add, or selecting a listed drive then Edit (or double clicking the list item) opens an ELS Axis Configuration Edit dialog box.

A list entry can be deleted by clicking and highlighting the entry, then clicking the Delete button.

Clicking OK in the ELS Axis Configuration dialog box saves the changes, clicking Cancel abandons any changes and returns to the Visual Motion workspace.

![ELS Axis Configuration Edit](image)

The ratio of the slave to the master is set by entering integer **Turn** values for both slave and master. The CLC calculates the ratio insuring that the highest accuracy possible within the system is maintained.

Drives are synchronized in terms of velocity, phase or cam position. If enabled, an adjustment factor may be entered (multiplied constant for velocity, additive constant for phase: see the
ELSAdj icon or Cam icon for more information). The slave drive may have normal or reversed
direction with respect to the master.

Clicking the OK button places the new or edited configuration into the drive list. Multiple drives
may be added or edited from the same Configuration dialog box. Clicking Cancel returns to the
ELS Axis Configuration dialog box. Additions and edits entered using the OK button are
displayed in the drive list.

6.2.12. ELSAdj

The ELSAdj icon is used to adjust the velocity or phase of an ELS configured axis to compensate
for mechanical variations between master and slave axes. The axis may be specified by an
integer constant, variable, global variable or an equivalent label. A pop-up User Defined Labels
dialog box is available from both data entry boxes.

Type selects the method of adjustment:
If **absolute**, the *degrees or percent* edit field is the new offset.
If **incremental**, the *degrees or percent* edit field is added to the current offset. If in phase
mode and sum exceeds 360, it rolls over. If in velocity mode the sum is limited to -100 or
+300 percent.
If **continuous +** or **continuous -** (phase sync mode only), a velocity dependent amount is
added to the degree offset.

For phase synchronization the resulting slave phase is:

$$\varnothing_s = (\varnothing_m * (K_s/K_m)) + \text{adjust}$$
For velocity synchronization the resulting slave velocity is:

\[ V_s = V_m \times ((1 + \text{adjust}) \times (K_s/K_m)) \]

Where:
- \( \Omega_m \) = master axis phase
- \( V_m \) = master axis velocity
- \( K_s \) = slave axis master/slave ratio turns value
- \( K_m \) = master axis master/slave ratio turns value
- \( \text{adjust} \) = a value in the range of -100% to +300% for ratio, or 0° to +360° for phase.

The adjust value (velocity or phase angle) is a floating point constant, variable (Fx), global variable (GFx), or equivalent label indicating the percentage or angular adjustment.
6.2.13. ELSMode

The ELSMode icon is used to toggle an ELS axis between being synchronized to the master and operation as an independent single axis. Once switched into Single axis mode, the axis may be positioned independent of any ELS master/slave relationship (i.e., jogged into position), then returned to Sync to Master restoring the relation.

The ELSMode icon can also be used to switch an axis that is configured for Single Axis Mode into Velocity Mode. Axis parameter A-0-0180 must be set to 36 to put command velocity into the cyclic data. Axis parameter A-0-0004, bit 7 must be set to 1 to enable acceleration. The Sync to Master mode is then ignored.

![ELS Mode Change](image)

The axis is specified by an integer constant, variable, global variable or an equivalent label. Axes must be configured as ELS axes. A pop-up User Defined Labels dialog box is available from both data entry boxes.
6.2.14. Event

The Event icon is used to control the way that the system handles events; and may be used to control program flow by suspending a task's program execution pending the completion of an event. For the event icon to have an effect upon system operation, an event must have been previously entered in the CLC event table. Placing an Event icon on a task or subroutine workspace automatically opens an Event dialog box.

The event is specified by entering an event ID number indicating a valid event in CLC's event table. The event may be entered as an integer constant, variable (Ix), global variable (GIx), or an equivalent label.

An event icon may have one of three effects:

Disarm Event de-activates the specified event. If the event has already been made inactive, the icon has no effect.

Arm Event is valid only for timed events. An Arm Event icon starts the specified event's timer. If the event is currently active, the icon has no effect.

Wait for event done suspends the execution of program flow until the specified event has completed.
6.2.15. Finish

Each program task, subroutine and event function must end with a single Finish icon. Subroutines can return an optional single argument to the calling function. The return argument may be a constant or a variable. A return argument is a convenient way to get position when utilizing a common subroutine from more than one task.

![Finish Setup](image)

6.2.16. Go

The Go icon is used to enable one or all non-coordinated axes used in any task. It also enables the associated position, velocity or servo loops. It should be placed before an associated Move icon. The Go icon can also be used to resume multi-axis coordinated motion that has been stopped.

Placing a Go icon automatically opens the Go Setup dialog box. The Motion Type radio button specifies the type of motion to start. Choosing Single Axis Motion Type requires an entry in the Axis data entry box specifying the CLC axis to start. The axis is specified by a valid integer constant, variable, global variable or an equivalent label. Specifying "-1" as the single axis enables all the single axes assigned to the current task in the Axis icon. Specifying "0" starts the virtual master of an ELS system.

Choosing Coordinated Motion Type to resume motion requires selection of one of the four CLC tasks from the pull-down Task pick list. The coordinated motion must have been halted by a coordinated stop. Aborted coordinated motion should not be simply resumed.

A pop-up dialog box of User Defined Labels is available.
6.2.17. Home

The Home icon commands the CLC to send an axis home signal to the specified drive. This is a single-axis non-coordinated motion command. Placing a Home icon on a task or subroutine workspace automatically opens the Homing Setup dialog box.

![Homing Setup Dialog Box]

The axis to home may be specified by a valid integer constant, variable (Ix), global variable (GIx), or an equivalent label. A pop-up dialog box of User Defined Labels is available.

Once commanded to home, the drive homes the axis to the home position without further intervention from the CLC. Any errors in the drive's homing operation are reported to the CLC.

The Home icon uses the internal homing capability of the intelligent DDS drive to perform the homing operation. For homing to occur, the homing parameters in the specified drive must have been setup prior to executing the Home icon. The homing parameters may be set using the Homing Setup dialog box accessed through the Drives menu item under Setup. See the Drive Reference menu selection under Parameters in the CLC Drive Parameter Editor. Refer to the DDS-2 manual for more information about homing and the homing parameters.
6.2.18. I/O

The I/O icon is used to control the state of I/O register bits. Placing an I/O icon on a task or subroutine workspace automatically opens an I/O Setup dialog box. Entering a number as a register ID code, or an equivalent label, in the I/O Register data entry box selects the target I/O register.

The I/O Bit mask data entry box permits entry of an integer constant or equivalent label, as an "and" mask for the target register. Multiple bits can be changed with a single icon by entering an I/O bit mask that specifies more than one bit (e.g., 0x0605, or a label equivalent to the desired mask). A pop-up dialog box of Bit Labels is available for adding, editing or deleting register bit labels.

The radio buttons for I/O state determine whether the target register bits, enabled by the I/O Bit mask, are cleared (logic zero, or off) or set (logic one, or on).

A pop-up dialog box of Register Labels is accessible, and permits adding, editing or deleting register labels. The Register Labels dialog box provides a scrolling list of default labels for the standard CLC system, axis, task, and DDS-2 drive I/O card control and status registers.

---

**I/O Setup Box**

- **I/O Register or label:**
  - [OK]
  - [Cancel]
- **I/O Bit mask or label:**
  - [Label...]
- **I/O state:**
  - [Off]
  - [On]
6.2.19. Join

A Join icon makes it possible to connect one line to another. VisualMotion icons have a maximum number of inputs; "join" overcomes this limitation by permitting many program flow paths to combine into a single path.

The Join icon is often the only method of completing a program flow, since VisualMotion cannot cross interconnecting lines. In addition, your program may require branching to several different calculations depending upon a certain condition. After the Calc icons you can use join icons to return to the main program flow before it enters the next icon.
6.2.20. Joint

The Joint icon is used for point-to-point movement and joint (elbow) positioning, typical to robotic motion.

A Joint move is an absolute point-to-point move, with only the endpoint of the move specified. It is the most efficient type of move because the path calculated by the path planner is optimized to minimize time. A Joint move uses the axis’ maximum accel and decel rates, while line and circle coordinated motion commands use Path Maximum percentages (defined in Task parameters) and Maximum Acceleration and Deceleration rates (defined in Axis parameters). The actual path taken to the specified point is not defined and may assume whatever form the path planner requires; however, once programmed the path is repeatable.

The destination is specified as an entry in the absolute point table as an integer constant, variable, global variable or an equivalent label.
6.2.21. Line

All icons in each task, subroutine and event function must be connected. The line icon is used to draw a line indicating program flow from one icon to another. Clicking on the beginning icon surrounds the icon with a box. Clicking on the ending icon automatically draws a line from the first to the second icon, with an arrowhead indicating the direction of program flow.

Under some circumstances VisualMotion may be unable to route a line and displays a "Connection could not be made, try connecting adjacent blocks" dialog box. Lines may be manually routed by clicking adjacent empty squares on the invisible workspace grid from the first icon to the second. A manually placed line may not cross another line; attempting to do so displays an error box.

A line connecting two icons may be deleted by using the Scissor icon from the Utility palette. Simply select the Scissor icon, position the Scissor over the line to be deleted and press the left mouse button.
6.2.22. Move

The Move icon is used to program movement on any single non-coordinated axis from any task. The Move icon does not initiate any motion (see the Go icon). Placing a Move icon on a task or subroutine workspace automatically displays a Single Axis Move Setup dialog box.

The Type radio buttons permit either absolute or relative movement of a specified distance from the end of the current segment. A pop-up User Defined Labels dialog box is available for each data entry box in the Single Axis Move Setup dialog box.

The Axis to move data entry box requires entry of an integer constant, variable, global variable or equivalent label, specifying the axis to move. The Distance data entry box permits the entry of a floating point constant, variable, global variable or an equivalent label specifying the distance to move.

Up to four optional events (event functions) may be associated with the move. A data entry box for each of four possible events allows you to specify an event by a valid integer constant, variable (Ix), global variable (GIx), or an equivalent label, identifying the event. Any specified event functions must be programmed before running the program. The AxisEvt icon may also be used to enable events on an axis. The limit of four enabled events includes both Move and AxisEvt enabled events.
6.2.23. Msg

The Message icon is used to select a status or diagnostic message (up to eighty characters) at a specified point in the program flow. Placing a Msg icon on a task or subroutine workspace automatically displays a Message Setup dialog box.

Messages are used to inform a user about the current state of the program through Task Parameters T-0-0122 and T-0-0123, or through the teach pendant. Systems using Direct ASCII Communication may obtain messages through the RS-232 port. Messages may also be sent to the top line of the pendant and to the serial ports. Messages are available after the icon is executed in the program and remain in effect until another message of the same type is executed.

Status messages tell a user or machine operator something about the ongoing process. At run time, the current status and diagnostic messages can be viewed by selecting Tasks from VisualMotion's Status menu.

Diagnostic messages are typically used to provide information about the current state of the system, e.g., "412 No drives were found on ring." If an error occurs during task execution, this diagnostic message is overwritten with an error message.

An optional variable may also be displayed. This is useful for operator interface or debugging. Messages may have one formatted variable in its string by using “%s”.

The displayed variable may also have a corresponding label. The label dialog box will be displayed when focus is on the variable field and cursor is doubled-clicked within the Message Setup Box.
6.2.24. Param

The Param icon is used to transfer specified CLC or DDS drive-related parameters or variables between the PC Host system and an array of CLC variables. Transferring parameters to CLC-stored variables is the only way that programs can perform calculations and logical operations on parameter values.

Subsequently downloading the modified CLC variables permits dynamic modification of parameters during system operation. Placing a Param icon on a task or subroutine workspace automatically displays a Parameter Transfer dialog box.

The radio buttons for Transfer type determine the direction of the transfer, from a specified CLC constant or variable to the parameter, or from the parameter to a CLC variable. The Parameter type radio buttons select from System, Axis, Task or Drive associated parameters.

Specify the parameter to be transferred in the Parameter ID Number data entry box as an integer or equivalent label.

If the System parameter type is selected, specify the source or target variable by entering an integer or floating point constant, variable (Ix or Fx), global variable (GIx or GFx), or an equivalent label in the appropriate data box.

If an Axis or Drive type parameter is selected, the additional data entry box enabled requires a valid integer constant, variable (Ix), global variable (GIx), or an equivalent label specifying the axis or drive ID.
If the **Task** parameter type is selected, a pull-down pick list (with scroll buttons) permits selection of one of the four CLC tasks (A, B, C or D).

A pop-up User Defined Labels dialog box may be opened by clicking once within the appropriate data entry box, then double-clicking immediately outside the entry box.

**Note**: Certain system parameters are read-only and cannot be changed, while others may be modified only with the CLC system in "Parameter" mode. See the chapters on Parameters and CLC I/O (system status and control registers) for more information.

### 6.2.25. Path

The Path icon is used to set up multi-axis coordinated straight line motion. Placing a Path icon on a task or subroutine workspace automatically displays a Coordinated Line Setup dialog box.

Motion may be Absolute or Relative and is defined by the two endpoints of the line of motion. These points must be existing entries in the absolute and/or relative point tables. Points are specified by an index into the point table using an integer constant, variable, global variable or an equivalent label. (Refer to the Calc icon.)

An absolute move begins from the endpoint of the previous path segment, or current position if the system is halted, and terminates at the absolute point specified.

The relative move begins at the endpoint of the previous path segment, or current position if the system is halted, and terminates at the relative offset point specified.

A pop-up dialog box listing the User Define Labels is available for the ABS and REL data entry boxes.
6.2.26. PID

PID icon and setup dialogs.

Program variable block initialized when program compiled.
6.2.27. PLS

The PLS icon initializes the programmable limit switch data structure when the program is compiled. The location of the PLS icon in the program is not important since it is not executed when the program is run. If more than one PLS icon is in the program, the last one encountered, while compiling, will overwrite any previous ones. The PLS is disabled while the output register is zero or when all bits have zero in both the on and off positions.

A programmable limit switch supports 16 outputs and a phase advance. The position input for the PLS can only be assigned to an ELS, Virtual, Real or Drive Based master. The outputs are updated every SERCOS cycle.

See Chapter 4. Visual Motion Menu Commands for more PLS information.
6.2.28. Position

The Position icon is used to obtain the current position of one of the tasks from the path planner. This is useful when operating the system in a teach mode without a Teach Pendant. It also could be used to check axis position while running a program. For instance, a loop could be setup whereby the path position is constantly obtained and compared to a target position. When the path position matches the target position, a bit is be toggled, causing a new branch test result to alter program flow.

Placing a Position icon on a task or subroutine workspace automatically displays a Get Path Position dialog box. The Task pop-down menu permits selecting one of the four CLC tasks (A - D). The absolute point table destination for the position data may be specified by an integer constant, variable, global variable or an equivalent label. A pop-up User Defined Labels dialog box is available.
6.2.29. PrmBit

The PrmBit icon is used to set or clear one or more bits in a CLC or DDS parameter. Parameter are specified by type (System, Axis, Task or Drive), group (axis number, drive number, or task) and number. System parameters have only type and number. For specific parameter information see the CLC or DDS manual.

The source may be a float or integer constant, global or program (Fx, GFx, GIx, Ix) variable, or equivalent label.

Bits may be specified in decimal (i.e. 1, 5, 12) or hexadecimal form (i.e. 0x1000, 0x8097, 0x0001), or equivalent label.

Note: Some parameters are read-only.
6.2.30. Prmint

The Prmint icon may be used to initialize individual CLC system, axis, drive, and task parameters. Parameters are initialized during SERCOS phase 2, before the user program executes.

```
Parameter Initialize
```

```
Parameter type:
- System
- Axis
- Task
- Drive

Parameter ID Number
{ integer or label }

Source
{ integer, float, Fx, GFx, GIx, Ix, or label }
```

The parameter type is selected by choosing one of the Parameter type radio buttons.

Selecting Axis or Drive enables a data entry box, permitting entry of an integer constant, variable (Ix), global variable (GIx), or equivalent label specifying the axis or drive.

Selecting Task enables a pop-down menu with selections for the four CLC tasks.

The Parameter ID Number data entry box requires an integer constant or an equivalent label, specifying a valid number (in the range of 1 to 65535) for the parameter to be initialized.

The Source data entry box specifies the value to be loaded to the parameter. The type specified must match the type of the parameter, i.e. a floating point constant, variable (Fx), global variable (GFx), or equivalent label for a floating point value; an integer constant, variable (Ix), global variable (GIx), or an equivalent label.
6.2.31. Probe

The Probe icon is used to configure the DDS-2 drive probe capability for a drive assigned to the specified axis. Once configured you may assign a CLC event to trigger from the drive's probe transition by using a ProbeEvt icon.

The Axis may be specified by an integer constant, variable (Ix), global variable (GIx), or an equivalent label.

One or two Capture Triggers may be enabled. Each may be assigned to none, or one of the two DDS-2 probe signals, with selection of trigger on rising or falling edge:

- Probe 1 low (0) to high (1)
- Probe 1 high (1) to low (0)
- Probe 2 low (0) to high (1)
- Probe 2 high (1) to low (0)

Probe inputs to a DDS-2 drive are typically used as inputs for "in-position" switches or "feelers" associated with the drive axis.
6.2.32. ProbeEvt

The ProbeEvt icon is used to enable a CLC event for triggering by a DDS-2 drive probe transition. The event function must have been entered by choosing the Add Event Function from Visual Motion's Edit menu.

The Axis triggering the event is specified by an integer constant, variable (Ix), global variable (GIx), or an equivalent label. The corresponding probe trigger must be assigned to the same axis using an Axis or Probe icon.

The DDS-2 probe event used as the Event Trigger Input is selected from the pop-down menu. Choices include none (trigger disabled), Probe 1 low-to-high, Probe 1 high-to-low, Probe 2 low-to-high, and Probe 2 high-to-low.

The Event to be triggered is specified by an integer constant, variable (Ix), global variable (GIx), or an equivalent label that provides an index into the event table.

The axis' position at the time of the probe trigger is immediately available by reading the event's "a" field from the event table, (i.e., EVT[x].a), or by using the Param icon to read the corresponding probe's position capture from the drive as a drive parameter (Dx.00130 - Dx.00133). Because this instruction does not enable a repeating event, it must be executed for each new edge to be scanned.
6.2.33. Ratio

The Ratio icon is used to set the ratio between two axes in a master/slave relationship, as when a gantry robot has a motor on each side of a supporting circular track. It is used in Tasks which also contain coordinated axes, because ELS and coordinated motion cannot be used together in the same Task.

The Ratio icon may also be used to link several axes to the same master axis or by chaining several drives together. For example: if drive 1 is master to drive 2, drive 3 can be made a slave to drive 2, thereby linking drive 3 to drive 1 through drive 2. Note that the response time of drives chained in this manner is additive, at least one SERCOS cycle (approximately 2ms) must occur between each master to slave link.

Placing a Ratio icon on a task or subroutine workspace automatically displays a Ratioed Axis Setup dialog box. The Master and Slave Axis should already be assigned as ratioed axes in the Axis icon. See the Axis Icon for more information.

The Master and Slave axes are selected by entering an integer constant, variable, global variable or an equivalent label. The ELS Virtual Master, axis 0, may be used as the Master Axis. A pop-up User Defined Labels dialog box is available for all four data entry boxes. One axis is selected as a master axis. A slaved axis must not be assigned to any task other than the task containing the master axis.

Rotation of the master axis controls the proportional rotation of the selected slave axis according to the formula:

\[
\text{Slave axis velocity} = \text{Master axis velocity} \times (\text{Slave ratio factor} \div \text{Master ratio factor})
\]

The Master and Slave Turns data entry boxes permit simple entry of the ratio between the axes. The ratio factors may be a floating point constant, variable, global variable, or an equivalent...
label. Individual data boxes for master and slave eliminate the need to normalize the ratio. For example, simply entering the number of teeth on each of two meshed gears allows VisualMotion to calculate the necessary coefficient. Each factor is in floating point format and is normalized before the division operation. This insures that the calculation maintains maximum precision with repeating decimals such as 2/3.

Entering the axis ratio factors using the Ratio icon automatically updates the master factor parameter (As.0031) and slave factor parameter (As.0032).

The slave axis is maintained in the DDS-2 drive's position loop mode at all times, even when the user program is not running. The drive's shaft position remains locked to the master axis within the torque limits of the DDS-2 drive.

6.2.34. Reg

The Reg icon is used to transfer data between the I/O registers and either the integer variable or global integer variable table. One to 1024 registers (16-bits each) may be transferred with a single command. Placing a Reg icon on a task or subroutine workspace automatically displays a Register Transfer dialog box.

Transfer type selects the direction of data transfer between the I/O register(s) and a CLC integer variable table. Transfer count specifies the number of consecutive 16-bit words to be transferred. Starting register number specifies the base, or lowest address of the source for the start of the transfer. Starting integer variable specifies the base address of the target of the transfer.

The Transfer count, Starting register number and starting integer variable must be an integer constant, variable (Ix), global variable (GIx), or an equivalent label.

6.2.35. RobotOrg
The RobotOrg icon is used in coordinated motion programs to construct a zero frame of reference from the x, y, z, roll, pitch and yaw coordinates of a relative point. This moves the effective origin of the robot from the default to the location specified by the programmed relative point.

For example, if REL[3] = {1, 2, 3, 0, 0, 0, ...} and this point is specified as the robot origin, then the robot origin would be offset by one unit along the x axis, two units along the y axis and three units along the z axis. Once the icon instruction is executed, all jogging, teaching and path locations are affected by the new origin.

### 6.2.36. RobotTol

The RobotTol icon is used in coordinated motion programs to construct a tool frame of reference from the x, y, z, roll, pitch and yaw coordinates of a relative point. This moves the effective end-of-arm tool to the location specified by the programmed relative point.

For example, if REL[4] = {0, 0, 10, 0, 0, 0, ...} and this point is specified as the robot tool location, then the robot tool location would be offset by ten units along the z axis from the faceplate of the robot. Once the icon instruction is executed, all jogging, teaching and path locations are affected by the end-of-arm tool location.
The Scissor icon is used to delete a line between two icons. Select the Scissor on the palette, position the tip of the Scissor cursor over the line to delete, press the left mouse button to delete.

6.2.38. Seq

The Sequencer icon is used to initialize and/or run a Sequencer list. A Sequencer list contains user-defined steps that can be edited "on-line" without having to edit, compile and download a program. Each step contains a list of functions that will be executed sequentially following the order of the list. Each step also can have up to five function argument which will be passed to the subroutine when the function is executed. Refer to "Programming Concepts" for more information.

The Sequencer name is a number or label equating to a number. The number has a range of 1 to n, where n is the number of sequences defined in the Size icon. If the Sequencer is only to run, it’s number can be specified by a program or global integer( Ix, GIx ).

If Initialize Sequencer or Initialize and Run is selected, the Sequencer can be Setup... by the Sequencer icon according to the following icon dialog boxes. If Run Sequencer is selected the Setup... option is inactive and the icon can run a Sequencer that has previously been setup using the Sequencer Editor under the Data menu or the Teach Pendant.

When Setup... is selected the following dialog boxes can be used to add, edit, delete or insert step lists within a Sequencer.
After selecting Add, a Step Name can be entered and edited as illustrated below. Each step has room for a twenty character name to uniquely identify it.

Functions need to have the "Callable from Sequence" access flag set in the START icon of the subroutine before they can be added to a Step List. Function arguments must also be defined in the START icon for them to be used.

After selecting Add, a function can be entered and edited as illustrated below. The function argument values must be within the limits predefined in the START icon of the subroutine. See SUB icon and Programming Concepts.

When a Sequencer is initialized by an icon this setup will overwrite any other changes that may have been previously done "on-line" every time the icon is executed. Select "Run Sequencer" to maintain on-line Sequencer editing.
6.2.39. Size

The Size icon allocates CLC memory for the different types of data storage each task requires. Placing a Size icon in the workspace opens a Task Data Storage Sizing dialog box. The entered values are the maximum number of variables, points, events, zones and Sequencer components that can be used in all the tasks.

The Sequencer components consist of lists, steps and functions. The values entered for lists and steps represent the total number of different lists or steps that are allowed in a program. If the same step is used more than once, in one or more lists, it still only counts as one towards the total. The total number of functions represent every time a function is executed in a step, even if the same function is used more than once. Each function will add to the total every time it is used. See Programming Concepts for more information.

The Size icon can only be used in the four main tasks; it cannot be used within a subroutine or event. In most cases, only one Size icon needs to be used per program. A size icon should be used in each Task that contains coordinated motion. When used, the Size icon should be placed immediately after the Start icon.

A program has a default value of fifty entries for integer and float variables. If any task uses more than 50 entries of any type, the Size icon must be used to allocate additional space. If a program uses less than fifty entries of any type, using the Size icon to reduce the allocation from the default to the actual number of data entries saves CLC memory space for other programs.

Variables each require 4 bytes, points 42 bytes, and events 118 bytes. The byte total of allocated memory is shown at the bottom of Tasks Data Storage Sizing dialog box. After a new value is entered and the cursor is moved outside the dialog box, a new byte total is calculated and displayed.
6.2.40. Start

The four CLC tasks (A through D), subroutines and event functions must all contain a Start icon. The Start icon indicates the beginning of program flow to the CLC compiler and also declares function arguments and local variables. The Start icon is equivalent to the CLC Text Language TASK/START command statement.

Stack based variables (function arguments and local variables) exist only while in the function (task, subroutines, or event) where they are declared. This type of variable is useful for temporary results within a function or to pass values to a function. The same label may be used for a function argument or local variable in different functions. The label may not be a keyword or other label (user, register, or bit). The total number of function arguments and local variables in a function is limited to 16. Up to 5 function arguments may be passed to a subroutine.
6.2.41. Stop

The Stop icon is used to halt motion on one or all axes used in a task. It will return the drives to an AH (Drive halt) state. Placing a Stop icon on a task or subroutine workspace automatically displays a Stop Setup dialog box.

![Stop Setup dialog box]

**Coordinated Abort** and **Coordinated Stop** decelerate motion on path, stopping all axes associated with the coordinated motion. Selecting Coordinated Abort or Coordinated Stop enables a pop-down menu permitting selection of one of the four CLC tasks.

Restarting motion after a Coordinated Abort requires toggling the Cycle/Start bit of the associated Task Control register. Motion stopped using a Coordinated Stop may be resumed with a Go icon, although resuming timed events that are programmed for motion at operating speed may result in events occurring at unexpected times.

**Non-Coordinated abort** and **Non-Coordinated Stop** stops motion on the specified axis by decelerating the axis to zero velocity using the currently programmed rate. The axis to stop may be specified by a valid integer constant, variable, global variable or an equivalent label. Specifying a "-1" stops all axes in the task.

**NOTE:** After an **Abort**, all queued events and the "look-ahead" motion calculated by the path planner are lost, although position information may be read from the stopped axes. After a **Stop**, both queued events and the calculated path are retained.

**ELS Virtual Master** decelerates the virtual master and all assigned axes to a stop at a point specified by an index into the point table using an integer constant, variable, global variable or an equivalent label. Multiple revolutions will be traversed if necessary to maintain programmed deceleration.

6.2.42. Sub
The Sub icon is used to enter a subroutine function from the normal program flow. Placing a Sub icon on a task or subroutine workspace automatically displays a Subroutine dialog box.

After you enter a name for your subroutine, clicking on the Enter Function button automatically replaces your current task or subroutine workspace with a new subroutine workspace in the main Visual Motion window. The title bar of the main window changes to show the name of the new subroutine. You can now use Visual Motion icons to build a subroutine in exactly the same manner as building a main Task. To return from a subroutine workspace to one of the four tasks, select the task from the View menu.

**Function arguments** are stack based variables passed to a subroutine. The setup dialog box for the Sub icon has entries for the function arguments. Up to 5 arguments may be passed to a subroutine. The arguments can be constants or variables (float, integer, ABS index or REL index). The function arguments need to be declared in the subroutine Start icon dialog box. A minimum and maximum value must also be entered for each of these arguments. This limits the range of argument values used during Sequencer editing from the teach pendant or the Sequencer interface.

Function arguments are not allowed in tasks or events, however, an optional return argument may be passed back to a task from a subroutine. A return argument is a convenient way to "get position", utilizing a common subroutine from more than one task.

**Local variables** are stack based variables used in a function. The total number of local variables and function arguments is limited to 16. Local variables, floats and integers are initialized to zero.
on each entry into the function. This type of variable can be used to store temporary results in a re-entrant function called on by more than one task. Local variables can also be used to avoid possible conflicts with program or global variables between tasks.
6.2.43. Veloc

The Veloc icon is used to specify a rate for motion on a single non-coordinated axis. The axis may be specified by a valid integer constant, variable, global variable or an equivalent label. The velocity may be entered as a floating point constant, variable, global variable or an equivalent label. A pop-up dialog box of User Defined Labels is available for both data entry boxes.
6.2.44. VME

The VME icon is used to transfer CLC data, such as variable and point tables, between the CLC's local memory and the VME address space.

Before using a VME Bus Data Transfer, the CLC interface with the VME bus must be configured for your VME system using the VME Configure selection from Visual Motion's Setup menu.

![VME Bus Data Transfer](image)

The Transfer type radio buttons select the direction of the data transfer operation. Data may be transferred between the CLC's local memory (point tables or variables) and the VME global memory address space that is accessible to any VME card in the VME card rack.

The VME section of the dialog box allows setting the base address, the address width, data width, and data format of the data within the VME system memory space. These settings must match your VME space or runtime errors will occur. In some cases when the data format of the source and target don't match, source data is sign extended or chopped to fit into the target.

**Address Bus Width** provides a pop-down menu allowing selection of A32/A24 (both 32 bit, 4 gigabyte and 24 bit, 16 megabyte), or A16 (16 bit, 64 kilobyte) VME memory address space.
**Address** specifies the starting address for the transfer operation.

**Data Bus Width** provides a pop-down menu allowing selection of D8 (byte), D16 (word), or D32 (double word) data width for the transfer with VME memory.

**Format** permits setting the data format of the VME source or target for the transfer. Available formats include 32 bit floating point; 8, 16, or 32 bit integer; 8, 16, or 32 bit unsigned integer; or point for transferring the multibyte data contained in an absolute or relative point table entry. Data is automatically converted to the data type of the specified target.

**Transfer count** specifies the number of data items to transfer with a single VME icon.

**Byte Order** sets the high-byte, low-byte order of multi-byte data (words, double-words, etc.). The byte order must be set to be compatible with the target processor on the VME bus that will be accessing the data. I.E., Motorola (the CLC uses the Motorola 68000 series) or Intel (such as the 80x86 and Pentium).

**Local data** specifies the starting, or base address of the CLC data to be transferred. Local data may be specified as an integer index to the CLC's integer or floating point variables (Ix or Fx), or global variables (GIx or GFx), or an integer index to the CLC's absolute or relative point tables. You may also use a valid equivalent label (a pop-up User Defined Labels dialog box is available). You may also send a constant to VME memory by specifying an integer or floating point constant, or equivalent label as the Local data.

The Data Bus Format above must be set to match the format of the Local data.

**Data Format Conversion and Data Width Errors**

Data is automatically converted to the data type of the specified target. For example, saving a CLC floating point variable to a VME memory location in I32 format automatically converts the float to a 32-bit integer. The conversion truncates the data if it exceeds the range of the target data type. It is the programmer’s responsibility to check the data size and do any necessary data scaling before invoking the data transfer and associated automatic conversion.

A compile-time error will result if the VME source data width exceeds the selected data format width. In other words, don't attempt to move VME D16-width data into a U8 (unsigned integer) because the specified source data width is physically larger than the target width. For more information on the limitations of direct VME memory access see the CLC-V Backplane Communication section in the appendix, “CLC Hardware.”
6.2.45. VMEEvt

The VMEEvt icon is used to trigger CLC events between cards on the same VME bus. In order for the VME event to trigger, the event function must be have been previously defined and enabled on the target card. You may enter an Event function by choosing the Add Event Function selection from Visual Motion's Edit menu. Events are enabled by an Event icon.

Selecting Broadcast Event Type broadcasts an event trigger to one or more CLC cards in a multi-card VME CLC system. Cards are individually selected, or the All checkbox can be used to select all cards at once. The Priority radio buttons select the triggering event's priority on the targeted cards, independent of the priority of the local task generating the trigger. Task A has the highest priority, Task D the lowest.

A Broadcast VME event is restricted to the first 63 events in the event table of the target card. (I.E. the event index must equate to less than 64.) The targeted CLC card(s) must have a local event function defined for the specified event index number and the event must be enabled on the target card(s). The event index is entered as an integer constant, variable (Ix), global variable (GIx), or an equivalent label. A User Defined Labels pop-up dialog box is available.

Selecting a Short Address event type writes to the VME bus short address space at a specific memory address (i.e., the CLC's' mailbox 4). For the event to occur, the event must be set up as a VME Short Address Event on the target card and the event must be enabled.

Only one VME Short Address Event on each card should be enabled at one time. If a program requires more than one Short Address event on a target card, it is the programmer's responsibility to insure that only one event is enabled at one time.
6.2.46. Wait

The Wait icon is used to hold the execution of the program flow at the Wait icon until a specified condition has been satisfied. The condition may be related to a single axis' position, time, I/O state, or path planner state (for coordinated motion). Placing a Wait icon on a task or subroutine workspace automatically displays a Wait Control Box dialog box.

**Axis in Position** pauses program execution until the specified axis reaches the in position window of the associated drive. The axis may be specified by a valid integer constant, variable (Ix), global variable (GIx), or an equivalent label. If a "-1" is entered, program execution will wait until all axes assigned to the task are within their respective position windows before continuing.

Selecting **Coordinated State** pauses program execution and tests the state of the path planner for the specified point until the planner enters the selected processing state. The At point state pop-down menu provides the eight path planner test states listed below:

- Segment Ready - Path planner has processed and queued the specified point.
- Acceleration - Task's coordinated motion is accelerating into the segment ending at the specified point.
- Slew (constant speed) - Task's coordinated motion is traversing the segment ending at the specified point.
Blending - Task’s coordinated motion is traversing the blend segment calculated for the specified point.

Target Deceleration - Task's coordinated motion is in deceleration in the segment ending at the specified point.

Controlled Stop - Task's coordinated motion is decelerating on the segment ending at the specified point after a commanded stop.

Stopped - Task's coordinated motion is stopped after a commanded stop.

At Target - Task's coordinated motion is at the specified point.

Done - Task's coordinated motion has completed for the specified point.

**I/O State** pauses program execution until the specified I/O condition for the specified I/O register is satisfied. The I/O register is specified by entering a valid I/O register ID number or equivalent label. A pop-up Register Labels dialog box is available.

The contents of the specified register may be "anded" with a bit mask for the register contents. The bit mask allows "masking out" unwanted bits (by specifying "0" in the bit position in the mask), and permits the on/off condition to be effected by more than a single bit (by specifying "1" to enable the bit). A pop-up Bit Labels dialog box is available.

The I/O State radio buttons determine how the wait condition is satisfied. I/O State "on" (high or "1") requires that all the enabled bits are logical one. I/O State "Off" (low or "0") requires that all enabled bits are zero.

**Time** pauses program execution for a specified time delay. Enter the number of milliseconds in the Time Delay data entry box. The delay may be specified by an integer constant, variable (Ix), global variable (Glx), or an equivalent label. A pop-up dialog box of User Defined Labels is available for both data entry boxes.
CHAPTER 7. TEXT LANGUAGE PROGRAMMING

7.1. Overview

As an alternative to graphical Icon Programming, Visual Motion also provides a Text Language programming capability. Text Language permits writing CLC programs using Windows’ Notepad or a similar ASCII text-only editor that does not use extended characters for formatting. The command syntax combines features of Basic and assembly language, with built-in extensions such as an IF-ELSE-ENDIF construct.

7.2. Directives

Directive statements provide CLC instructions that control the compiler, control the loading of programs between the CLC and Host, and control the manner in which the CLC manages the different portions of a user program.

Some of the directives are only used to initialize or configure the system. These directives are removed from the instruction stream and are not executed during the normal operation of a program.

7.2.1. EQU (Equate)

The EQU equate directive permits you to assign a symbolic name to a literal constant. Equating is most beneficial when using a constant repeatedly. An equated symbol name is global to all CLC tasks, subroutines and events within your application program. The name may have up to 20 ASCII characters. Remember that no white spaces are allowed within the name.

EQU statements should be placed at the very beginning of your program and must appear before the name is referenced in the program. Should you later need to change the value of the constant, only the single instance of the value in the EQU statement at the beginning of your program has to be modified.

Every time a program is compiled, the symbolic names in the program are replaced with their equated literal value.
Examples:

```plaintext
RATIO1    equ  2.7348 ; a ratio between two coupled shafts
DONE      equ  0x1000 ; a bit mask for an I/O register
```

The compiler generates an error if a symbolic name is used in a program before it has been defined, or if the name has previously been used to define a variable, axis or identify a subroutine.

### 7.2.2. DEFINE

The DEFINE statement is used to assign a symbolic name to an integer or floating point variable in the corresponding variable table. Using a DEFINE permits you to use a mnemonic name instead of the "F[n]" or "I[n]" standard format for floats or integers. (See the section on Variables for details on the standard format.) A defined name may have up to 20 ASCII characters; no white spaces are allowed. A defined name is global to all CLC tasks, subroutines and events.

Unlike equates, defined names are downloaded and stored in separate variable tables in the CLC. Tables allow access to the variables by a user device through the CLC’s serial communications port; or, if used, through a teach pendant.

```plaintext
DEFINE   F05   Start_Veloc ; a floating point user variable
DEFINE   I03   Finished_Parts ; an integer user variable
```

A DEFINE statement only associates a symbolic name with a variable. It does not initialize the variable to any value. If you need the variable to have an initial value, the value must be set using an expression, after the DEFINE statement and before the variable is used in the program.

```plaintext
Start_Veloc   =   100 ; sets the defined variable F5 to 100
Finished_Parts =  5000 ; sets the defined variable I3 to 5000
```

As with the EQU statement, the compiler generates an error if a symbolic name is used in a program before it has been defined, or if the name has previously been used to define a variable, axis or identify a subroutine.
7.2.3. EVENT/START & EVENT/END

The EVENT/START and EVENT/END directives are used to tell the compiler that the program statements contained within the directives are an event function. Every EVENT/START statement must have an ending EVENT/END statement. Unlike a called subroutine, an event function should not have a RETURN statement at the end of the event function program statements. Proper termination of an event is handled by the compiler and CLC multitasking executive.

Example:

Event1:  ;Mark to identify the beginning of the event program statements
EVENT/START
.
.
;event program statements
.
EVENT/END

7.2.4. TASK/START & TASK/END

The TASK/START and TASK/END instructions are used to identify the respective beginning and ending of program statements associated with a task. The task instructions do not perform any initialization, but are necessary indicators for the CLC compiler. Each TASK/START must have a "Task_(task letter):" mark before the TASK/START statement. Each block of program statements must have a corresponding TASK/END statement. Task statements should not be nested.

Example:

Task_A: TASK/START  A ;Mark to identify the beginning of the task statements
.
.
TASK/END  A ;task program statements
.

7.3. CLC Instructions

7.3.1. Instruction Format

The entire CLC statement line is limited to a maximum length of 80 characters including the optional label or mark, instruction, arguments, and comment.

White space may be used to format lines for easier readability. You may use tabs or spaces for white space. However, do not use white spaces in the middle of a label, mark, instruction, option or argument. If you need to separate a label or mark’s characters for clarity, use the underscore character, " _ ".

Visual Motion Text Language program statements should be written using the following generalized format:

MARK: INSTRUCTION/OPTION ARG1, ARG2,...ARGn ;COMMENT

where:

MARK is an optional user-assigned symbolic name immediately followed by a colon (:). A mark is typically used as an entry point to a series of program statements that begin with the line containing the mark.

INSTRUCTION is the CLC instruction mnemonic for the instruction.

OPTION is a modifier that may be required by the instruction and is separated from the preceding instruction by a forward slash (/) (known as a "virgule" if you care about typesetting terminology). White space is not allowed between the instruction and its modifier.

ARG1, ARG2, ...ARGn are arguments for the instruction. The number of arguments is specific to the instruction, ranging from none to several. If more than one argument is supplied to an instruction the arguments must be separated by commas. The argument(s) must be separated from the instruction by one or more white spaces.

COMMENT is a text string that you may enter to describe the purpose of an instruction statement. A comment must begin with a semi-colon (;). Comments are used only as an aid to understanding the program. When generating the executable program code, the CLC compiler ignores everything from the semicolon to the end of the line.
Writing Program Statements
Instructions must be separated from their arguments by at least one white space, either tabs or spaces. If more than one argument is used each argument must be separated by a comma ",".

An equivalent label may be used in place of a literal specification (such as a constant, variable or table entry), unless otherwise noted.

Using Comments
While you’re in the process of writing a program you know what you intend the program statements to accomplish. You may not remember as well a month, or six months later. Using comments liberally always makes it easier to understand what the program is supposed to do. Multiple lines beginning with semicolon may be used to enter several lines of descriptive information about a subroutine, or as a "header" at the beginning of a program or major program section.

Example:

;HOMING SUBROUTINE
;This routine homes the x and y axes
;Expects as Inputs -
; I/O register N = a value, bit n = another value
;Changes outputs -
; I/O register N, bit n
;Changes system variables -
; name1
; name2
;etc.

SUBR01: ;a mark must be used to identify the beginning of
; a subroutine's statements

. ; subroutine program statements

RETURN ;return to the statement following the calling GoSub statement

The CLC instructions available for text language programming are listed in alphabetical order on the following pages. Each instruction is listed with a description, its syntax and examples.

Braces ("{" and "}") are used to contain optional arguments, where more than a single argument may be supplied.
For example:

```
AXIS/EVENT  axis, event1 (, event2, event3, event4)
```

where:

- `axis` and `event1` are required, and `event2`, `event3` and `event4` are optional.

Labels generated by the DEFINE or EQU instructions may be used in place of constant or variable values.
7.3.2. AXIS/EVENT

Syntax:

AXIS/EVENT axis, event1 {, event2, event3, event4}

where:

<table>
<thead>
<tr>
<th>argument</th>
<th>valid type(s)</th>
<th>range</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>axis</td>
<td>integer</td>
<td>1 to the maximum number of valid axes</td>
<td>axis number</td>
</tr>
<tr>
<td>event1</td>
<td>integer</td>
<td>1 to the maximum number of valid events</td>
<td>event table index</td>
</tr>
<tr>
<td>event2</td>
<td>integer</td>
<td></td>
<td></td>
</tr>
<tr>
<td>event3</td>
<td>integer</td>
<td></td>
<td></td>
</tr>
<tr>
<td>event4</td>
<td>integer</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Variables or labels used for arguments must equate to valid values at run-time or an error will result.

Description:
The AXIS/EVENT instruction can enable up to four specified Repeating Position events for a single-axis, ELS, ratio, or velocity mode axis. The events are armed immediately, and an event function will execute each time the axis reaches the absolute position stored in the "a" field of the event specified in the event table.

Example:

```
Axis_One equ 1
Cam_2 equ 15
Cam_3 equ 16

axis/event Axis_One,Cam_2,Cam_3 ;activates repeating events
    ; Cam_2 and Cam_3 on Axis_One
```


7.3.3. AXIS/HOME

Syntax:

AXIS/HOME  axis

where:

<table>
<thead>
<tr>
<th>argument</th>
<th>valid type(s)</th>
<th>range</th>
<th>description</th>
</tr>
</thead>
</table>
| axis     | integer  
- constant
- variable Ix, I[x]
- global variable GIx, GI[x]
- label   | 1 to the maximum number of valid axes              | axis number         |

Variables or labels used for arguments must equate to valid values at run-time or an error will result.

Description:
AXIS/HOME signals the DDS to perform its drive-internal homing routine. CLC program execution pauses until a homing completed signal (or error message) is received from the DDS.

Before AXIS/HOME may be used, the appropriate homing parameters must be set in the DDS drive. Refer to the DDS manual for a description of the internal homing function and the required drive parameters and settings.

AXIS/HOME may be used with both coordinated and non-coordinated motion.

Example:

```plaintext
:   
define   105,Axis_AX,
Axis_BX   equ   4
speed_BX   equ   100.00
accel_BX   equ   250.00
:
axis/home Axis_AX ;homes the DDS-2 drive specified by integer variable 5
axis/home Axis_BX ;homes the DDS-2 drive 4
```
### 7.3.4. AXIS/INITIALIZE

**Syntax:**

```
AXIS/INITIALIZE    axis, speed, accel
```

where:

<table>
<thead>
<tr>
<th>argument</th>
<th>valid type(s)</th>
<th>range</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>axis</td>
<td>integer&lt;br&gt;- constant&lt;br&gt;- variable Ix,I[x]&lt;br&gt;- global variable GIx,GI[x]&lt;br&gt;- label</td>
<td>1 to the maximum&lt;br&gt;number of valid&lt;br&gt;axes</td>
<td>axis number</td>
</tr>
<tr>
<td>speed</td>
<td>floating point&lt;br&gt;- constant&lt;br&gt;- variable Fx, F[x]&lt;br&gt;- global variable GFx, GF[x]&lt;br&gt;- label</td>
<td></td>
<td>initial speed</td>
</tr>
<tr>
<td>accel</td>
<td>floating point&lt;br&gt;- constant&lt;br&gt;- variable Fx, F[x]&lt;br&gt;- global variable GFx, GF[x]&lt;br&gt;- label</td>
<td></td>
<td>initial acceleration</td>
</tr>
</tbody>
</table>

Variables or labels used for arguments must equate to valid values at run-time or an error will result.

**Description:**

AXIS/INITIALIZE is used for single axis non-coordinated motion. The instruction associates the specified axis with the user task containing the instruction. Initial speed and acceleration specified by variables may be modified by the user program.
Example:

```
define F10, speed
define F11, accel
define I01, Grip_Event
Grip_Event equ 5
TaskA_Ax1 equ 3
speed equ 100.0
accel equ 1000.0

axis/initialize TA_Ax1, speed, accel
axis/move TA_Ax1, A, 100.0, Grip_Event
```
7.3.5. AXIS/MOVE (Single Axis, Non-Coordinated)

Syntax:

AXIS/MOVE  axis, mode, dist, {event1, event2, event3, event4}

where:

<table>
<thead>
<tr>
<th>argument</th>
<th>valid type(s)</th>
<th>range</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>axis</td>
<td>integer</td>
<td>1 to the maximum number of valid axes</td>
<td>axis number</td>
</tr>
<tr>
<td></td>
<td>- constant</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- variable Ix, I[x]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- global variable GIx, GI[x]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- label</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mode</td>
<td>ASCII character</td>
<td>A for absolute R for relative</td>
<td>specifies the type of move</td>
</tr>
<tr>
<td>dist</td>
<td>floating point</td>
<td></td>
<td>distance to move</td>
</tr>
<tr>
<td></td>
<td>- constant</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- variable Fx, F[x], GF[x]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>event1</td>
<td>integer</td>
<td>1 to the maximum number of valid events</td>
<td>table index of first event</td>
</tr>
<tr>
<td>event2</td>
<td></td>
<td></td>
<td>table index of second event</td>
</tr>
<tr>
<td>event3</td>
<td></td>
<td></td>
<td>table index of third event</td>
</tr>
<tr>
<td>event4</td>
<td></td>
<td></td>
<td>table index of fourth event</td>
</tr>
<tr>
<td></td>
<td>- constant</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- variable Ix, I[x]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- global variable GIx, GI[x]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- label</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Variables or labels used for arguments must equate to valid values at run-time or an error will result.

Description:
AXIS/MOVE is used for single axis non-coordinated motion. Digital drives support this feature internally; no CLC path generation is needed, reducing the computational load on the path planner. Because the motion profile is generated within the drive, time-related events cannot be used with the AXIS/MOVE instruction.
Example:
speed   equ   100.0
accel   equ   1000.0
grip_event  equ   5
TaskC_Ax3   equ   7
F50     equ 100.0

axis/initialize  TC_Ax3, speed, accel
axis/move        TC_Ax3, A, F50, Grip_Event
### 7.3.6. AXIS/RATIO

**Syntax:**

AXIS/RATIO axis1, axis2, Mfactor, Sfactor

where:

<table>
<thead>
<tr>
<th>argument</th>
<th>valid type(s)</th>
<th>range</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>axis1</td>
<td>integer</td>
<td>1 to the maximum number of valid axes</td>
<td>master axis number</td>
</tr>
<tr>
<td></td>
<td>- constant</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- variable Ix,I[x]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- global variable GIx,GI[x]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- label</td>
<td></td>
<td></td>
</tr>
<tr>
<td>axis2</td>
<td>integer</td>
<td>1 to the maximum number of valid axes</td>
<td>slave axis number</td>
</tr>
<tr>
<td></td>
<td>- constant</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- variable Ix,I[x]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- global variable GIx,GI[x]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- label</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mfactor</td>
<td>floating point</td>
<td>32 bits</td>
<td>master axis factor</td>
</tr>
<tr>
<td></td>
<td>- constant</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- variable Fx,F[x]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- global variable GFx,GF[x]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- label</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sfactor</td>
<td>floating point</td>
<td>32 bits</td>
<td>slave axis factor</td>
</tr>
<tr>
<td></td>
<td>- constant</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- variable Fx,F[x]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- global variable GFx,GF[x]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- label</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Variables or labels used for arguments must equate to valid values at run-time or an error will result.

**Description:**

AXIS/RATIO establishes a mathematical relationship between the number of revolutions of the specified slave axis that will result from the rotation of the specified master axis, according to the formula:

\[
\text{slave axis velocity} = \text{master axis velocity} \times \left( \frac{\text{Sfactor}}{\text{Mfactor}} \right)
\]

The AXIS/RATIO command automatically updates the master factor parameter As.0031 and
slave factor parameter As.0032. The separate factors are normalized before division so that the calculation maintains maximum precision with repeating decimals such as 2/3.

Specifying a negative number for one of the Factors produces reversed rotation in the slaved axis.

Multiple AXIS/RATIO statements may be used to link several slave axes to a single master.

A slaved axis must not be assigned to any task other than the task containing the master axis.

Example:

```
Task_A:
task/start A ;start of code for task A
task/axis 1, 2 ;assign axes for task A
axis/ratio 1, 3, 4.0, -5.0 ;axis 3 linked to axis 1 at 5/4
.
;Operation results in 5 revolutions of axis 3 (in the reverse direction) for each 4 revolutions of axis 1.
```
### 7.3.7. AXIS/SPINDLE (Continuous Velocity Mode)

**Syntax:**

AXIS/SPINDLE  axis, rpm

where:

<table>
<thead>
<tr>
<th>argument</th>
<th>valid type(s)</th>
<th>range</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>axis</td>
<td>integer</td>
<td>0 to the maximum number of valid axes</td>
<td>axis number</td>
</tr>
<tr>
<td></td>
<td>- constant</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- variable Ix,I[x]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- global variable G1x,G1[x]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- label</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rpm</td>
<td>floating point</td>
<td>32 bits</td>
<td>revolution per minute</td>
</tr>
<tr>
<td></td>
<td>- constant</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- variable Fx,F[x]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- global variable GFx,GF[x]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- label</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Variables or labels used for arguments must equate to valid values at run-time or an error will result.

**Description:**

The AXIS/SPINDLE instruction tells the specified axis (and consequently the DDS drive) to run at a constant velocity. The acceleration ramp profile is generated within the drive, according to the previously set drive acceleration parameters. The sign of the speed argument determines the direction of rotation.

This instruction also enables the axis to go; the ‘Axis/Start’ command need not be used.

The current speed of the axis can be monitored by reading the axis speed parameter from the drive using the PARAMETER/GET instruction.
Example:

```
speed equ 100.0
accel equ 1000.0
TA_Ax1 equ 4

axis/initialize TA_Ax1, speed, accel
axis/spindle TA_Ax1, -22.2
```


7.3.8. AXIS/START

Syntax:

AXIS/START axis

where:

<table>
<thead>
<tr>
<th>argument</th>
<th>valid type(s)</th>
<th>range</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>axis</td>
<td>integer, constant, variable Ix, I[x], global variable GIx, GI[x], label</td>
<td>0 to the maximum number of valid axes</td>
<td>axis number</td>
</tr>
</tbody>
</table>

Variables or labels used for arguments must equate to valid values at run-time or an error will result.

**Description:**
AXIS/START begins motion for the selected axis. The instruction signals the DDS drive(s) to begin motion toward the target position set by the AXIS/MOVE command. The axis must be programmed for non-coordinated motion. (E.G., single-axis or velocity mode.)

Once an axis has been started, each new AXIS/MOVE instruction begins a new movement, without requiring an AXIS/START.

If the specified axis is set to "-1", the instruction begins motion for all single-axis and velocity mode axes programmed in the task. A program may set target positions for several axes using multiple AXIS/MOVE commands, then begin motion on all axes at the same time with a single "-1" argument to AXIS/START.
Example:

```
.
.
TaskA_AX1 equ 1 ; use drive 1
TaskA_AX2 equ 3 ; use drive 3
TaskA_AX3 equ 5 ; use drive 5
.
.
axis/move TaskA_AX1, A, F50 ; set position for drive 1
axis/move TaskA_AX2, A, F55 ; set position for drive 3
axis/move TaskA_AX3, R, F60 ; set position for drive 5
.
.
axis/start -1 ; start motion, all axes
.
.
```
7.3.9. AXIS/STOP  
Syntax:  
AXIS/STOP  axis  

where:  

<table>
<thead>
<tr>
<th>argument</th>
<th>valid type(s)</th>
<th>range</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>axis</td>
<td>integer</td>
<td>0 to the maximum number of valid axes</td>
<td>axis number</td>
</tr>
<tr>
<td></td>
<td>- constant</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- variable Ix, I[x]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- global variable GIx, GI[x]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- label</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Variables or labels used for arguments must equate to valid values at run-time or an error will result.

Description:  
AXIS/STOP halts motion for the selected axis. The instruction signals the DDS drive(s) to decelerate the drive to zero velocity using the rate set in the drive’s deceleration parameter. The axis must be programmed for non-coordinated motion. (E.G., single-axis or velocity mode.)

Once an axis has been stopped, an AXIS/MOVE instruction does not effect movement of the axis until an AXIS/START instruction is executed.

If the specified axis is set to "-1", the instruction disables motion for all single-axis and velocity mode axes programmed in the task. A typical use is to synchronize motion on several axes, or to stop all motion on an I/O or error condition.
Example:

```
TaskA_AX1 = 1
TaskA_AX2 = 3
TaskA_AX3 = 5

axis/move TaskA_AX1, A, F50 ; set position for drive 1
axis/move TaskA_AX2, A, F55 ; set position for drive 3
axis/move TaskA_AX3, R, F60 ; set position for drive 5
axis/start -1 ; enables all drives

axis/stop -1 ; stops motion on all axes
```
### 7.3.10. AXIS/WAIT (Axis Wait For In-Position)

**Syntax:**

AXIS/WAIT  axis

where:

<table>
<thead>
<tr>
<th>argument</th>
<th>valid type(s)</th>
<th>range</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>axis</td>
<td>integer</td>
<td>0 to the maximum number of valid axes</td>
<td>axis number</td>
</tr>
<tr>
<td></td>
<td>- constant</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- variable Ix,I[x]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- global variable GIx,GI[x]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- label</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Variables or labels used for arguments must equate to valid values at run-time or an error will result.

**Description:**

AXIS/WAIT suspends task execution until the DDS drive associated with the specified axis signals the CLC that the axis is in position and zero velocity. The drive's in position and zero velocity values are set by the drive's Position Window and Zero Velocity parameters.

If the specified axis is set to "-1", the instruction waits until all axes in the task are positioned. A typical use is to synchronize motion on several axes.

If the axis position is never achieved the task remains waiting, suspended indefinitely.
Example:

```
.event1 equ 10
Turn_Gripper_On equ 12
TA_Ax1 equ 3
define F10, speed
define F11, accel
speed equ 00.0
accel equ 1000.0

Axis/Initialize TA_Ax1, speed, accel
Axis/Move TA_Ax1, R, 100.0, Event1, Turn_Gripper_On
Axis/Wait TA_Ax1

```
7.3.11. CALL
Syntax:

\[ \text{retval} = \text{CALL function\_label, arg1, arg2, \ldots argN} \]

where:

<table>
<thead>
<tr>
<th>argument</th>
<th>valid type(s)</th>
<th>range</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>function_label</td>
<td>label</td>
<td>any valid function label</td>
<td>name of function</td>
</tr>
<tr>
<td>arg1 ...argN</td>
<td>Fx, Ix, GFx, GIx, integer label, float label or point label</td>
<td>values passed to function</td>
<td></td>
</tr>
<tr>
<td>retval (optional)</td>
<td>Fx, Ix, GFx, GIx, integer label or point label</td>
<td>variable to receive return value of function</td>
<td></td>
</tr>
</tbody>
</table>

Description:
This instruction calls the function ‘function\_label’ with the arguments specified, and optionally returns a value.

Example:

`; Main task calling subroutine and returning value
Task_A: TASK/START A
    I1 = CALL sub, 5, 10, 20
    TASK/END A
; Subroutine to multiply input values
sub: FUNCTION/START U
    FUNCTION/ARG COUNT1, I, 1, 300
    FUNCTION/ARG COUNT2, I, 1, 300
    FUNCTION/ARG COUNT3, I, 1, 300
    LOCAL/VARIABLE TAB, I
    TAB = (COUNT1 * COUNT2) * COUNT3
    FUNCTION/END TAB
7.3.12. CAM/ACTIVATE
Syntax:

CAM/ACTIVATE  axis, cam, M, N, H, L

where:

<table>
<thead>
<tr>
<th>argument</th>
<th>valid type(s)</th>
<th>range</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>axis</td>
<td>integer</td>
<td>1 to the maximum number of valid axes</td>
<td>axis number</td>
</tr>
<tr>
<td></td>
<td>- constant</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- variable Ix, I[x]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- global variable GIx, GI[x]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- label</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cam</td>
<td>integer</td>
<td>1 to the maximum number of valid cam tables</td>
<td>specify cam table ID</td>
</tr>
<tr>
<td></td>
<td>- constant</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- variable Ix, I[x]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- global variable GIx, GI[x]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- label</td>
<td></td>
<td></td>
</tr>
<tr>
<td>M, N, H, L</td>
<td>floating point</td>
<td>32 bits</td>
<td>define coefficients in cam equation. If L is not specified, it is assumed L=0.</td>
</tr>
<tr>
<td></td>
<td>- constant</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- variable Fx, F[x]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- global variable GFx, GF[x]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- label</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Description:
The CAM/ACTIVATE instruction is used to associate a cam to an axis and to supply coefficients for using the cam. For further information on cam theory of operation, refer to the Cam Icon in the Icon Programming chapter.
7.3.13. CAM/ADJUST

Syntax:

CAM/ADJUST  axis, mode, type, degrees

where:

<table>
<thead>
<tr>
<th>argument</th>
<th>valid type(s)</th>
<th>range</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>axis</td>
<td>constant, integer,</td>
<td>any valid axis number</td>
<td>axis to adjust</td>
</tr>
<tr>
<td></td>
<td>GIx, Ix, label</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mode</td>
<td>integer</td>
<td>1 - master phase adjust</td>
<td>selects master or slave</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 - slave phase adjust</td>
<td>(Mph or Sph in equation)</td>
</tr>
<tr>
<td>type</td>
<td>integer</td>
<td>1 - absolute</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 - incremental</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>3 - continuous +</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>4 - continuous -</td>
<td></td>
</tr>
<tr>
<td>degrees/</td>
<td>Fx, GFx, GIx, integer</td>
<td></td>
<td>phase adjust target value</td>
</tr>
<tr>
<td>percent</td>
<td>or label</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Description:
This instruction selects which phase adjust to perform and starts the phase adjust. With type = 1, it functions the same as the ELS/ADJUST instruction. ELS/ADJUST functions the same as before for compatibility with existing programs.

There are two phase offset values for a cam axis: a master phase adjust, and a slave phase adjust. The master phase adjust shifts the position in the cam table relative to the master position. The slave phase adjust shifts the position of the slave axis. Since it is not related to the shape of the cam, the slave phase adjust is not multiplied by any of the cam factors.

The CLC can perform only one phase adjust at a time. This instruction will have no effect until the previous phase adjust is complete if the phase adjust type is different. Bit 4 in the axis status register is set to (0) when a phase offset is in progress, and (1) if the phase offset is complete.

Example:

```
CAM/ADJUST 1,2,1,35
```
7.3.14. CAM/BUILD

Syntax:

```
CAM/BUILD   cam_number, option, start_point, end_point, wait
```

where:

<table>
<thead>
<tr>
<th>argument</th>
<th>valid type(s)</th>
<th>range</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cam_number</td>
<td>integer</td>
<td>1-8 = cam stored</td>
<td>integer or constant containing number of cam</td>
</tr>
<tr>
<td></td>
<td>- constant</td>
<td>on CLC</td>
<td>to be built</td>
</tr>
<tr>
<td>option</td>
<td>integer</td>
<td>1 = use pcam utility</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- constant</td>
<td>2 = use spline utility</td>
<td>cam generation option</td>
</tr>
<tr>
<td>start_point</td>
<td>integer</td>
<td>starting index in the ABS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- constant</td>
<td>point table to use for cam generation</td>
<td></td>
</tr>
<tr>
<td>end_point:</td>
<td>integer</td>
<td>end index in the ABS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- constant</td>
<td>point table used for cam generation</td>
<td></td>
</tr>
<tr>
<td>wait:</td>
<td>constant</td>
<td>0 or 1</td>
<td>0 = don’t wait for completion of build instruction</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 = wait for the cam to be ready for activation</td>
</tr>
</tbody>
</table>

Description:

The CAM/BUILD command builds a cam internally on the CLC and stores it on the CLC in the cam number indicated. The CLC’s ABS point table is used to build the cam. The ABS point elements may be changed from within the program using CALC statements. Changes in the point table do not affect the cam until the CAM/BUILD command is executed.

For the pcam option, the X elements of the point table are the master positions, and the Y elements are the corresponding slave positions. The CLC builds a jerk-limited position profile between these points.

The cam generation may take one second or longer. The ‘wait’ option can be set to (0) to exit this instruction immediately and keep executing instructions while the cam is being built.

The CAM/STATUS instruction can be used to check if a cam is ready for activation. If the
‘wait’ option is set to (1), the program flow will be stopped in this instruction until the cam is ready for activation.

The CAM/BUILD command can be used to store a cam to an inactive location on the CLC. After the cam has been built, the CAM/ACTIVATE command can activate it for an axis.

Notes
Because the cam is stored in nonvolatile memory on the CLC, it is not necessary to execute this command each time through the program. A flag variable can be set and checked the next time through the program to avoid long delays when starting the program. For on-line changes, a register bit or variable should be checked each time through the program loop to avoid continually generating the cam, which consumes CLC resources and can slow down the program.

It is necessary to size the point table at compile-time to allow enough points for the profiles that will be needed.

Errors will be issued when:
The selected cam is currently active for any axis.
The point range exceeds the bounds of the point table.
Less than two points are defined.
The cam number is not valid (out of range or drive is not configured).
An error occurred when sending the cam to the drive.
When using PCAM option, and the first x position isn’t 0 and the last x position isn’t 360.
The x position exceeds the modulo of the master.
7.3.15. CAM/STATUS Command

Syntax:

Refer to IF CAM/STATUS Command

Description:
After a cam download or after the CAM/BUILD command, it takes some time to calculate and store the new cam. When a cam is activated on the CLC, it does not run until the cam is at the end of its cycle.

This command sets a task’s status word with a logical result of the cam status being check for.

Status: integer of status being checked for

0 = no valid cam is stored
1 = cam is being calculated
2 = cam is being sent to drive (drive cams only)
3 = cam is ready for activation
4 = cam is active and running (CLC cams: for any axis)

Errors will be issued when:
The cam number is not valid (out of range or drive is not configured).
7.3.16. CAPTURE/ENABLE

Syntax:

CAPTURE/ENABLE    axis, probe, event

where:

<table>
<thead>
<tr>
<th>argument</th>
<th>valid type(s)</th>
<th>range</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>axis</td>
<td>integer, constant, variable Ix,I[x], global variable GIx, GI[x], label</td>
<td>1 to the maximum number of valid axes</td>
<td>axis number</td>
</tr>
<tr>
<td>probe</td>
<td>integer, constant, variable Ix,I[x], global variable GIx, GI[x], label</td>
<td>1 to 4</td>
<td>specify DDS drive probe and edge transition</td>
</tr>
<tr>
<td>event</td>
<td>integer, constant, variable Ix,I[x], global variable GIx, GI[x], label</td>
<td>1 to the maximum number of valid events</td>
<td>enable specified event and trigger on probe transition</td>
</tr>
</tbody>
</table>

Variables or labels used for arguments must equate to valid values at run-time or an error will result.

Description:
The CAPTURE/ENABLE instruction enables a feedback capture event for an axis. Each axis can have up to four active capture events (one for each probe edge).

Specifying "0" for the event number disables capture for the specified probe transition.

Probe specifies which of the two probe inputs that will be enabled or disabled, and a transition direction (positive or negative going) for the input. An error is issued if the probe and edge were not selected as a trigger using a CAPTURE/SETUP command.

1 = Probe 1 positive transition  
2 = Probe 1 negative transition  
3 = Probe 2 positive transition  
4 = Probe 2 negative transition  

When the drive detects the selected transition on its probe input, the CLC transfers the probe
position into the event table element "a" (i.e., EVT[n].a) specified by the event number, and executes the optional associated event function.

The CALC statement can be used to read the position from the event table and perform any necessary calculations or adjustments. The probe position is also placed in a CLC axis parameter.

This instruction does not enable a repeating event. A CAPTURE/ENABLE statement must be executed for each new edge transition to be captured.

Example:

Capture/Setup Axis_1, In_Place ;setup for trigger

Capture/Event Axis_1, In_Place, Mark_edge ;capture position on trigger
7.3.17. CAPTURE/SETUP

Syntax:

CAPTURE/SETUP  axis, trigger1

where:

<table>
<thead>
<tr>
<th>argument</th>
<th>valid type(s)</th>
<th>range</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>axis</td>
<td>integer</td>
<td>1 to the maximum number of valid</td>
<td>axis number</td>
</tr>
<tr>
<td></td>
<td>- constant</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- variable Ix,I[x]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- global variable Glx, Gl[x]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- label</td>
<td></td>
<td></td>
</tr>
<tr>
<td>trigger1</td>
<td>integer</td>
<td>1 to 4</td>
<td>specifies the first DDS probe and trigger edge</td>
</tr>
<tr>
<td></td>
<td>- constant</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- variable Ix,I[x]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- global variable Glx,Gl[x]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- label</td>
<td></td>
<td></td>
</tr>
<tr>
<td>trigger2</td>
<td>integer</td>
<td>1 to 4</td>
<td>specifies the second DDS probe and trigger edge</td>
</tr>
<tr>
<td></td>
<td>- constant</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- variable Ix,I[x]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- global variable Glx,Gl[x]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- label</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Variables or labels used for arguments must equate to valid values at run-time or an error will result.

Description:
CAPTURE/SETUP is executed only at program activation. The instruction configures a CLC axis and associated DDS-2 drive to use the internal position feedback capture capability of the drive. One or two triggers can be selected, each may be triggered by either a low-to-high or high-to-low transition of either probe input.

The trigger selects a probe input edge to be enabled in the drive and allocated to SERCOS real-time bit 1. The CLC also allocates space in the drive's cyclic data for the captured probe position. Trigger1 may be specified by:

1 = Probe 1 positive transition  2 = Probe 1 negative transition
3 = Probe 2 positive transition  4 = Probe 2 negative transition
Example:

Capture/Setup  Axis_1, In_Place ; setup for trigger

Capture/Event  Axis_1, In_Place, Mark_edge ; capture position on trigger
7.3.18. DATA/SIZE (Configure Memory Allocations)

Syntax:

DATA/SIZE Integers, Floats, Apoints, Rpoints, Events, Zones

Where:

<table>
<thead>
<tr>
<th>argument</th>
<th>valid type(s)</th>
<th>range</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integers</td>
<td>integer</td>
<td>0 to available CLC memory</td>
<td>task memory allocated for Integer variables</td>
</tr>
<tr>
<td></td>
<td>- constant</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Floats</td>
<td>same as above</td>
<td>same as above</td>
<td>task memory allocated for Floating Point variables</td>
</tr>
<tr>
<td>Apoints</td>
<td>same as above</td>
<td>same as above</td>
<td>task memory allocated for Absolute Point Table entries</td>
</tr>
<tr>
<td>Rpoints</td>
<td>same as above</td>
<td>same as above</td>
<td>task memory allocated for Relative Point Table entries</td>
</tr>
<tr>
<td>Events</td>
<td>same as above</td>
<td>same as above</td>
<td>task memory allocated for Event Table entries</td>
</tr>
<tr>
<td>Zones</td>
<td>same as above</td>
<td>same as above</td>
<td>task memory allocated for Zone Table entries</td>
</tr>
</tbody>
</table>

Variables or labels used for arguments must equate to valid values at run-time or an error will result.

Description:

The DATA/SIZE instruction is used within each task to specify the amount of memory to be allocated to each data type required by the task. The total CLC memory allocated for a program is the sum of the allocations for each data type, in each of the four tasks. Once allocated, memory is global and may be accessed by all four tasks.

The DATA/SIZE command can only be used within the four main tasks. It must not be used within a subroutine or event function.
Example:

;Task A
Int_NumA equ 20
FP_NumA equ 20
ABS_NumA equ 30
REL_NumA equ 20

;Task B
ABS_NumB equ 10
REL_NumB equ 10
EVT_NumB equ 10

Task_A:
Task/Start A
Data/Size Int_NumA, FP_NumA, ABS_NumA, REL_NumA, 0
.;program code
Task/End A

Task_B:
Task/Start B
Data/Size 0, 0, ABS_NumB, REL_NumB, EVT_NumB
.;program code
Task/End B
Each integer or floating point entry requires 4 bytes of CLC memory.

Each absolute or relative point table entry requires 44 bytes of CLC memory.

Each event table entry requires 120 bytes of CLC memory.

Each zone table entry requires 28 bytes of CLC memory.

The CLC memory allocated for the program is:

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Task A</th>
<th>Task B</th>
<th>Task C</th>
<th>Task D</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>20</td>
</tr>
<tr>
<td>floating point</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>20</td>
</tr>
<tr>
<td>absolute points</td>
<td>30</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>40</td>
</tr>
<tr>
<td>relative points</td>
<td>20</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>30</td>
</tr>
<tr>
<td>events</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>zones</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>memory required in bytes</td>
<td>2360</td>
<td>2080</td>
<td>0</td>
<td>0</td>
<td>4440</td>
</tr>
</tbody>
</table>
7.3.19. DEFINE (Define Name For Variable)

Syntax:

DEFINE variable, label

where:

<table>
<thead>
<tr>
<th>argument</th>
<th>valid type(s)</th>
<th>range</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>variable</td>
<td>integer</td>
<td>a valid variable</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- variable</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Ix or Fx</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- global variable</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Glx or GFx</td>
<td></td>
<td></td>
</tr>
<tr>
<td>label</td>
<td>ASCII</td>
<td>20 characters maximum</td>
<td></td>
</tr>
</tbody>
</table>

Variables or labels used for arguments must equate to valid values at run-time or an error will result.

Description:

DEFINE assigns a name to an integer or floating point variable or global variable.

White spaces (spaces or tabs) are not allowed within the character string.

The assignment must occur in the program before the defined variable is used.

The compiler generates an error if a name is used before it is defined; or if a subroutine, or another variable or constant has been previously assigned the same name.
Example:

True equ 1
False equ 0
Define F10, Rate_Of_Change ; Names F10
Define I22, Logical_State ; Names I22

Task_A:
Task/Start A
Rate_Of_Change = COS(F10)/2.0 ; Set the rate
Logical_State = True ; Set state true

;program statements

Task/End A
7.3.20. DELAY (Suspend Task Execution)

Syntax:

DELAY  time

where:

<table>
<thead>
<tr>
<th>argument</th>
<th>valid type(s)</th>
<th>range</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td>integer</td>
<td>1 to 360000</td>
<td>time delay in milliseconds</td>
</tr>
<tr>
<td></td>
<td>- constant</td>
<td>(32 bit maximum)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- variable I[x]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- global variable GI[x]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- label</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Variables or labels used for arguments must equate to valid values at run-time or an error will result.

Description:
When the program execution flow encounters the DELAY statement, the task issuing the instruction is suspended for the specified period of time. After the delay, task execution resumes with the next statement.

Example:

```
  
  delay 10          ; Delay 10 milliseconds
  
  delay
```
### 7.3.21. ELS/ADJUST (Adjust ELS Axis)

**Syntax:**

ELS/ADJUST \[ slave_axis \], \[ fine_adjust \], \[ adjust_type \]

or,

ELS/ADJUST \[ slave_axis \], \[ phase_offset \], \[ adjust_type \]

where:

<table>
<thead>
<tr>
<th>argument</th>
<th>valid type(s)</th>
<th>range</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>slave_axis</td>
<td>integer\n- constant\n- variable Ix,I[x]\n- global variable Glx,Gl[x]\n- label</td>
<td>1 to the maximum number of valid</td>
<td>axis number of slave</td>
</tr>
<tr>
<td>fine_adjust</td>
<td>floating point\n- constant\n- variable Fx,F[x]\n- global variable GFx,GF[x]\n- label</td>
<td>-100% to +300%</td>
<td>new value of velocity sync. fine ratio adjust</td>
</tr>
<tr>
<td>phase_offset</td>
<td>floating point\n- constant\n- variable Fx,F[x]\n- global variable GFx,GF[x]\n- label</td>
<td>0° to +360°</td>
<td>new value of phase sync. phase offset</td>
</tr>
<tr>
<td>adjust_type</td>
<td>absolute\nincremental\ncontinuous +\ncontinuous -</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Variables or labels used for arguments must equate to valid values at run-time or an error will result.

**Description:**

For ELS drives in velocity synchronous mode, this instruction sets the Ratio Fine Adjust parameter (P-0-83) on the drive. This parameter is a percent from -100 to 300, allowing an adjustment from stopping the slave to four times the parametric ratio (S-0-236 and S-0-237, set in parameter mode).
For ELS drives in phase synchronous mode, this instruction adjusts the phase offset in degrees. The drive will shift in position relative to the master based on the value in this constant or float variable.

If either parameter needs to read, the parameter transfer instruction can be used. This is useful when a variable must be set to an initial phase offset value. The parameter for phase offset is As.151, where s is the slave axis number.

Example:

```
DIST        EQU 20.0 ; Sets equate


TASK_B:TASK/START B
ELS/INIT 1, 1, 2, 1, 1 ; ELS axis initialization
ELS/MODE 1, 1 ; Slave synchronized to master via ELS
ELS/ADJUST 1, -100 ; Fine adjust for velocity sync.
```

7.3.22. ELS/DECEL (Set Master Decel)

Syntax:

ELS/DECEL rate

where:

<table>
<thead>
<tr>
<th>argument</th>
<th>valid type(s)</th>
<th>range</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rate</td>
<td>floating point</td>
<td>32 bits</td>
<td>sets the ELS master axis deceleration rate in radians</td>
</tr>
<tr>
<td></td>
<td>- constant</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- variable Fx,F[x]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- global variable</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>GFx,GF[x]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- label</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Variables or labels used for arguments must equate to valid values at run-time or an error will result.

Description:
This command sets the deceleration rate for the ELS virtual master in radians/sect^2.

Example:

DEFINE F1, RATE ; Defines label 'RATE'

TASK_A: TASK/START A
.
ELS/INIT 1, 1, 2, 1, 2 ; ELS axis initialization
.
ELS/DECEL RATE ; Sets deceleration rate
.
TASK/END A
### 7.3.23. ELS/INIT (Initialize ELS Axes)

**Syntax:**

ELS/INIT els_type, slave_axis, master_axis, encoder_type, sync_type

**where:**

<table>
<thead>
<tr>
<th>argument</th>
<th>valid type(s)</th>
<th>range</th>
<th>description</th>
</tr>
</thead>
</table>
| els_type       | integer - constant - label     | 1 to 4                    | 1 = Virtual master through SERCOS (default)  
2 = Real master to each slave  
3 = Real master through SERCOS  
4 = Follow axis feedback                                                  |
| slave_axis     | same as above                  | 1 to maximum number of valid axes | the axis synchronized to the selected master                                                          |
| master_axis    | same as above                  | 1 to maximum number of valid axes | If els_type is set to 1 or 2, this value is ignored and the master axis is set to 0.  
If els_type is set to 3, master_axis must be set to the ID of the drive connected to the external feedback. Only drives with an external feedback option (such as the DDS) may be selected.  
If els_type is set to 4, master_axis must be set to the drive the slave axes are to follow. Any SERCOS drive may be selected. |
| encoder_type   | same as above                  | 1 or 2                    | 1 = Primary encoder (default)  
2 = Secondary encoder                                                      |

This value selects the encoder that is used when els_type is set to 3. ("Real Master Through SERCOS")
[ELS/INIT table cont.]

| sync_type          | same as above | 1, 2 or 3 | 1 = Velocity synchronization  
|--------------------|---------------|-----------|-----------------------------
|                    |               |           | 2 = Phase synchronization  
|                    |               |           | 3 = Cam Synchronization  
|                    |               |           | This value sets the type of  
|                    |               |           | synchronization between the master  
|                    |               |           | and the slave.                 |

Variables or labels used for arguments must equate to valid values at run-time or an error will result.

**Description:**
In a CLC System there can be one ELS master, either virtual or real. The CLC executes the ELS/INIT command at program activation to automatically configure the master and slave drives. This command may be placed in any task and allows some of the axes in a system to be in manual mode while the others are following the master.

The four ELS master axis types are:

**Virtual master through SERCOS** (default). The CLC generates the master position to be followed by ELS slave drives. The virtual master uses the programmed velocity, acceleration, deceleration and E-stop deceleration rates.

**Real master to each slave.** Each slave drive uses its own external feedback, which can be daisy-chained to several drives to provide a common position reference.

**Real master through SERCOS.** The CLC reads the position from an external encoder connected to any drive and sends it through SERCOS to all the slaves as the master position.

**Follow axis feedback.** The CLC reads the feedback position from the master axis and sends it through SERCOS as the master position.
Example:

SLAVE_AXIS EQU 1 ; Sets equate
MASTER_AXIS EQU 2 ; "

TASK_C:TASK/START C
DATA/SIZE 5, 5, 5, 5, 0 ; Allocates memory for task C
ELS/INIT 4, SLAVE_AXIS, MASTER_AXIS, 1, 2
; ELS axis initialization

TASK/END C
7.3.24. ELS/MODE (Set ELS Axis Mode)

Syntax:

ELS/MODE slave_axis, mode

where:

<table>
<thead>
<tr>
<th>argument</th>
<th>valid type(s)</th>
<th>range</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>slave_axis</td>
<td>integer</td>
<td>1 to the maximum number of valid axes</td>
<td>axis number of slave</td>
</tr>
<tr>
<td></td>
<td>- constant Ix, I[x], GI[x], GI[x]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- label</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mode</td>
<td>integer</td>
<td>1, 2, 3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- constant Ix, I[x], GI[x], GI[x]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- label</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Variables or labels used for arguments must equate to valid values at run-time or an error will result.

Description:
This instruction sets the operating mode for an axis. It switches the drive between single-axis mode and synchronous mode. If slave_axis is set to (-1), the mode is switched for all slave axes in the task.

To enable synchronization, set "mode" to 2. The drive is synchronized to the master in phase synchronous or velocity synchronous mode. When switching into phase synchronous mode, the Phase Offset parameter is automatically set to (slave position - master position).

To disable synchronization at any time during the program, set "mode" to 1. The slave switches to single-axis mode and is halted. To run the slave in single-axis mode, the single-axis instructions may be used.

Set “mode” to 3 to switch an axis that is configured for Single Axis Mode into Velocity Mode.

Example:

```
TASK_D:TASK/START D
  .
ELS/INIT 1, 1, 2, 1, 1  ; ELS axis initialization
ELS/MODE 1, 1          ; Single Axis mode
ELS/DECEL F1            ; Sets deceleration rate
```
7.3.25. ELS/STOP

Syntax:

ELS/STOP position

where:

<table>
<thead>
<tr>
<th>argument</th>
<th>valid type(s)</th>
<th>range</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>position</td>
<td>constant</td>
<td>range of floating point numbers</td>
<td>position in which the virtual master will stop</td>
</tr>
<tr>
<td></td>
<td>- variable Fx, F[x],</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- global variable Gfx,Gf[x]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- label</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Variables or labels used for arguments must equate to valid values at run-time or an error will result.

Description:
This command specifies the target position in which the master velocity will come to a stop. The deceleration value will reflect on when the master velocity begins to stop.

Example:

ELS/STOP 0
7.3.26. EVENT/DONE (Signal Event Completed)

Syntax:

EVENT/DONE  event

where:

<table>
<thead>
<tr>
<th>argument</th>
<th>valid type(s)</th>
<th>range</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>event</td>
<td>integer</td>
<td>1 to the maximum number of valid events</td>
<td>disable the event</td>
</tr>
<tr>
<td></td>
<td>- constant</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- variable Ix, I[x]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- global variable GIx, GI[x]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- label</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Variables or labels used for arguments must equate to valid values at run-time or an error will result.

Description:
The EVENT/DONE instruction changes the status of an active event. Time based events are taken out of the event timer queue, distance based events are made inactive if in the event queue or pending. EVENT/DONE is primarily used to disable repeating timer events, however, it immediately disables any event.

Executing an EVENT/DONE has no effect on an inactive event.
Example:

```
time equ 1 ; use 0 for time
dist equ 1 ; use 1 for dist

Task_A:
Task/Start A
EVT[3].t = time ; Set event 3 type to time
EVT[3].a = 10 ; Set event 3 time to 10ms
EVT[3].m = "Motion Event Message" ; Set the message for event 3
EVT[3].f = Gripper ; Set the subroutine mark for event 3
Event/trigger 3 ; Trigger event 3
Event/wait 3 ; Wait until done
Event/done 3 ; event done

Task/End A
Gripper: ; Define event 0 function subroutine
Event/Start ; Mark the beginning of the event
   plc/set 1, 2 ; Turn on gripper
Event/End ; Mark the end of the event
```

```
7.3.27. EVENT/END (Mark End of Event)

Syntax:

EVENT/END

Description:
Executing the EVENT/END instruction returns program execution to the invoking program, enabling normal cycling.

This instruction is required by the compiler to indicate the end of an event function, and as an interrupt return for the CLC run-time executive.

Example:

```
; time equ 1 ; Set equate
dist equ 3 ; Set equate

time equ 1 ; Set equate
dist equ 3 ; Set equate

Task_A:
    Task/Start A
    EVT[3].t = time ; Set the event type
    EVT[3].a = 10 ; Set the event time (ms)
    EVT[3].m = "Motion Event Message" ; Set the message for event
    EVT[3].f = Grip ; Set the function for event

    Event/Trigger 3 ; Trigger event 3
    Event/Wait 3 ; Wait until done

    ;additional program statements

Task/End A
    Grip: Event/Start ; Define event
        PLC/Set 1, 2 ; Turn on gripper
        Event/End ; End of event
```

```
7.3.28. EVENT/START (Start of Event function)

Syntax:

EVENT/START

Description:
EVENT/START is a compiler directive used within a program to indicate the beginning of an event function.

Example:

time equ 1 ; Set equate
dist equ 3 ; Set equate

Task_A:
Task/Start A
EVT[3].t = time ; Set the event type
EVT[3].a = 10 ; Set the event time (ms)
EVT[3].m = "Motion Event Message" ; Set the message for event
EVT[3].f = Grip ; Set the function for event
Event/Trigger 3 ; Trigger event 3
Event/Wait 3 ; Wait until done

;program statements

Task/End A

Grip: Event/Start ; Define event
   PLC/Set 1, 2 ; Turn on gripper
   Event/End ; End of event
7.3.29. EVENT/TRIGGER (Trigger a Task Event)

Syntax:

EVENT/TRIGGER event

where:

<table>
<thead>
<tr>
<th>argument</th>
<th>valid type(s)</th>
<th>range</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>event</td>
<td>integer</td>
<td>1 to the maximum number of valid events</td>
<td>start event timer</td>
</tr>
<tr>
<td></td>
<td>- constant</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- variable Ix, I[x]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- global variable GIx, GI[x]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- label</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Variables or labels used for arguments must equate to valid values at run-time or an error will result.

Description:

EVENT/TRIGGER starts a repeating timer event. The subroutine specified by the event is repeatedly called at the rate specified in the event table entry. Task A's events have the highest priority.

EVENT/TRIGGER has no effect on repeating timer events that are already active.

An EVENT/TRIGGERed subroutine does not execute and enters a wait state if the task's event queue is full.

Caution: The execution time of the event function must not exceed the specified event repeat time. If the event function does not return before the event is re-triggered, the multitasking executive will execute the event function again, as soon as the event function returns. Program task execution will be blocked. Remember that ALL pending events, for a task, will complete before any task resumes execution.
Example:

time equ 1 ; Set equate
dist equ 1 ; Set equate
.
.
Task_A:
Task/Start A
EVT[3].t = time ; Set the event type
EVT[3].a = 10 ; Set the event time (ms)
EVT[3].m = "Motion Event Message" ; Set the message for event
EVT[3].f = Gripper ; Set the function for event
.
; Program code
.
Event/trigger 3 ; Trigger event 3
Event/wait 3 ; Wait until done
.
; Yet more program code
.
Task/End A
.
.
Gripper: ; Define event 0 function
Event/Start
    plc/set 2, 6 ; Turn on gripper
Event/End
7.3.30. EVENT/WAIT (Pause Task for Event Done Signal)

Syntax:

EVENT/WAIT  event

where:

<table>
<thead>
<tr>
<th>argument</th>
<th>valid type(s)</th>
<th>range</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>event</td>
<td>integer</td>
<td>1 to the maximum number of valid</td>
<td>pause task for event</td>
</tr>
<tr>
<td></td>
<td>constant</td>
<td>event</td>
<td></td>
</tr>
<tr>
<td></td>
<td>variable Ix,I[x]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>global variable GIx,GI[x]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>label</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Variables or labels used for arguments must equate to valid values at run-time or an error will result.

Description:

Event/Wait suspends the task it is issued from until an event done instruction for the event is executed.

A CLC run-time error results if this instruction is directed toward an inactive event.

Example:

time equ 1 ; Set "time" to a constant
dist equ 1 ; Set "dist" to a constant
.
.
.
Task_A:
Task/Start A
EVT[3].t = time ; Set the event 3 type to time
EVT[3].a = 10 ; Set the event 3 time (ms)
EVT[3].m = "Motion Event Message"; Set the message for event 3
EVT[3].f = Gripper ; Set the subroutine to use for event 3
.
.
Event/trigger  3 ; trigger event 3
Event/wait  3 ; pause task until done
.
.
Task/End A
Gripper:
Event/Start
    plc/set 2, 6
Event/End

; event 3 subroutine called from another task

; turn on gripper, I/O register 2, bit 6
7.3.31. FUNCTION/ARG

Syntax:

FUNCTION/ARG label, type, min_value, max_value

where:

<table>
<thead>
<tr>
<th>argument</th>
<th>valid type(s)</th>
<th>range</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>label</td>
<td>ASCII string</td>
<td>1 to 20 characters</td>
<td>name of argument</td>
</tr>
<tr>
<td>type</td>
<td>ASCII character</td>
<td>‘I’ = integer  <code>F’ = floating point  </code>ABS’ = absolute point index  `REL’ = relative point index</td>
<td>specifies type of access</td>
</tr>
<tr>
<td>min_value</td>
<td>constant or label</td>
<td></td>
<td>optional minimum value of argument</td>
</tr>
<tr>
<td>max_value</td>
<td>constant or label</td>
<td></td>
<td>optional maximum value of argument</td>
</tr>
</tbody>
</table>

Variables or labels used for arguments must equate to valid values at run-time or an error will result.

Description:
These instructions define the arguments passed to a function. Each argument is defined on a separate line immediately following the FUNCTION/START. The arguments must be defined in the order in which they are to appear in the CALL instruction or in the sequence table. An optional minimum and maximum value are available to the user interface for validation. A maximum of 5 function arguments may be used. A total of 16 local variables (LOCAL/VARIABLE, FUNCTION/ARG) may be used per function. Function argument labels are accessible from the teach pendant in a sequencer.

Example:

```
; Function: plc_float
; Reads value from a 16-bit PLC register and return a scaled value
plc_float: FUNCTION/START U ; access from teach pendant
FUNCTION/ARG regnum, I, 150, 160 ; register number of PLC register
FUNCTION/ARG scaler, F, 1, 100000 ; scaling value from PLC to float
LOCAL/VARIABLE itemp, I ; local temporary variable
LOCAL/VARIABLE retval, I ; local temporary variable
PLC/READ regnum, 1, itemp ; read register from PLC
retval = itemp * scaler ; scale to floating point
FUNCTION/END retval ; return with retval
```
7.3.32. FUNCTION/END

Syntax:

FUNCTION/END   value (optional)

where:

<table>
<thead>
<tr>
<th>argument</th>
<th>valid type(s)</th>
<th>range</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>constant, floating</td>
<td></td>
<td>return value from function</td>
</tr>
<tr>
<td>(optional)</td>
<td>point or integer label</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Variables or labels used for arguments must equate to valid values at run-time or an error will result.

Description:
This instruction defines the end of a function. If no value is included, the function returns 0. The function may return only one value, which may be a floating point or integer. The value is returned in the variable specified in the CALL instruction.

Example:

```plaintext
FUNCTION/END retval ; return with retval
```
### 7.3.33. FUNCTION/START

**Syntax:**

```plaintext
function_label: FUNCTION/START access_type
```

**where:**

<table>
<thead>
<tr>
<th>argument</th>
<th>valid type(s)</th>
<th>range</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>function_label</td>
<td>label</td>
<td>any valid function label</td>
<td>name of function</td>
</tr>
<tr>
<td>access_type</td>
<td>ASCII character</td>
<td>‘U’ = Accessible in user function sequencer list</td>
<td>specifies type of access</td>
</tr>
<tr>
<td></td>
<td></td>
<td>‘N’ = Not accessible</td>
<td></td>
</tr>
</tbody>
</table>

**Description:**

This instruction indicates the start of a function. The arguments and return value are declared in the FUNCTION/ARG, LOCAL/VARIABLE and FUNCTION/END commands. If access_type is not included, the function is not accessible in the user function list.

**Example:**

```plaintext
; Main task calling subroutine and returning value
Task_A: TASK/START A
   I1 = CALL sub, 5, 10, 20
   TASK/END A

; Subroutine to multiply input values
sub: FUNCTION/START U
   FUNCTION/ARG COUNT1, I, 1, 300
   FUNCTION/ARG COUNT2, I, 1, 300
   FUNCTION/ARG COUNT3, I, 1, 300
   LOCAL/VARIABLE TAB, I
   TAB = (COUNT1 * COUNT2) * COUNT3
   FUNCTION/END TAB
```
### 7.3.34. GOSUB (Go To Subroutine)

**Syntax:**

GOSUB mark

Where:

<table>
<thead>
<tr>
<th>argument</th>
<th>valid type(s)</th>
<th>range</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mark</td>
<td>label</td>
<td>any valid label at the beginning of a subroutine</td>
<td>go to marked function with return</td>
</tr>
</tbody>
</table>

Variables or labels used for arguments must equate to valid values at run-time or an error will result.

**Description:**

Executing a GOSUB statement saves the current position in the program execution of the user task, then begins executing subroutine statements beginning with the statement identified by the GOSUB mark.

When executing the subroutine encounters a RETURN statement, program execution resumes at the next executable program statement following the GOSUB statement that called the subroutine.

**Example:**

```plaintext
L00:   If I01 >= I02 gosub Sub2
       Else
       I10 = 255         ; Set I10 to 255
       gosub Sub1       ; Call subroutine one
       Endif
         .
         .
Sub1:  I01 = I01*I02  ; Multiply
       RETURN
Sub2:  I01 = I01/I03  ; Divide
       RETURN
```


7.3.35. GOTO (Go To Mark)

Syntax:

GOTO <name>

where:

<name> a valid Mark identifying a unique position in a program.

Description:
The CLC stops executing the current series of program statements and immediately begins executing the series that starts with the statement identified by the mark in the GOTO instruction.

This is an unconditional branch. Since a GOTO does not save the current position in the program execution flow, branching to a subroutine or event function results in an error when the execution flow encounters the RETURN statement.

Example:

L00:  If I01 >= I02
      GoTo L10
    Else
      I10 = 255
      GoTo L15
    Endif
L10:  I01 = I01*I02
      .
      . ;program statements
      .
      GoTo     Start1
L15:  I01 = I02/I03
      .
      . ;alternative program statements
      .
      GoTo     Start2
7.3.36. IF (If-Else-Endif Conditional Branch)
Syntax:

IF value1, test, value2
    or
    PLC/TEST ( reg, bit )
    CAM/STATUS ( cam_number, test, condition )

    <program statements to execute if the expression evaluates to non-zero>

ELSE

    <program statements to execute if the expression evaluates to zero>

ENDIF

where:

<table>
<thead>
<tr>
<th>argument</th>
<th>valid type(s)</th>
<th>range</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value1</td>
<td>integer or floating point</td>
<td>any valid constant, variable,</td>
<td>greater than, greater than, or</td>
</tr>
<tr>
<td></td>
<td>- constant</td>
<td>range</td>
<td>equal to less than, less than,</td>
</tr>
<tr>
<td></td>
<td>- variable</td>
<td></td>
<td>or equal to not equal</td>
</tr>
<tr>
<td></td>
<td>Ix, I[x] or Fx, F[x]</td>
<td></td>
<td>equivalent to</td>
</tr>
<tr>
<td></td>
<td>- global variable</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>GIx, GI[x] or GFx, GF[x]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- label</td>
<td></td>
<td></td>
</tr>
<tr>
<td>value2</td>
<td>same as above</td>
<td></td>
<td></td>
</tr>
<tr>
<td>test</td>
<td>ASCII characters</td>
<td>&gt;</td>
<td>greater than</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&gt;=</td>
<td>greater than or equal to</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;</td>
<td>less than</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;=</td>
<td>less than or equal to</td>
</tr>
<tr>
<td></td>
<td></td>
<td>!=</td>
<td>not equal</td>
</tr>
<tr>
<td></td>
<td></td>
<td>==</td>
<td>equivalent to</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(CAM/STATUS limited to )</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>! =</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>= =</td>
<td></td>
</tr>
<tr>
<td>reg</td>
<td>integer, constant or label, Ix, GIx</td>
<td>1-1024 CLC-V</td>
<td>Source register to test</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1-512 CLC-P, CLC-D</td>
<td></td>
</tr>
<tr>
<td>bit</td>
<td>integer, constant or label, Ix, GIx</td>
<td>1-16</td>
<td>Bit with register to test</td>
</tr>
</tbody>
</table>

Table continued on the following page.
Variables or labels used for arguments must equate to valid values at run-time or an error will result.

**Description:**
The IF-ELSE-ENDIF structure provides conditional execution of the program statements between the IF and ENDIF keywords, depending upon the evaluation of a test relationship. The control structure also provides an optional ELSE keyword for conditional program branching between two alternative series of program statements.

If the expression is true (has a non-zero value), and there is no ELSE keyword, the program statements between the IF and ENDIF keywords are executed. If the optional ELSE keyword is used, the statements between the IF and ELSE keywords are executed. Program execution then continues with the first program statement after the ENDIF keyword.

If the expression is false (has a zero value) and there is no ELSE keyword, the program continues with the first instruction after the ENDIF keyword. If the optional ELSE keyword is used, the program statements between ELSE and ENDIF keyword are executed. The program execution then continues with the first program statement after the ENDIF keyword.

“IF” structures may be nested up to eleven deep. All IF statements must be balanced with a matching ENDIF.

**Example:**

```
  L00:  If I01 >= I02
        Goto L10
    Else
       I10 = 255
        gosub Sub2
    Endif

  L10:  I01 = I01*I02
        ;program statements
              ...

  Sub2:  I01 = I02*I03
        RETURN
```
PLC/TEST Variations:

Directly ties in the PLC/TEST command to an IF-Else-Endif conditional branch. Tests a bit in the specified I/O register table enter. If PLC/TEST checks for an "on" condition. If !PLC/TEST checks for an "off" condition.

Example:

```plaintext
; PLC/TEST variations
if( PLC/TEST 100, stewart )
  F30 = 5
endif
if( !PLC/TEST 100, 2 )
  F30 = 5
endif
if PLC/Test 100, 1
  F30 = 5
endif
if !PLC/Test 100, 2
  F30 = 5
endif
if( PLC/Test( 100, 1 ))
  F30 = 5
endif
if plc/test( I5, GI6 )
  else
    F31 =6
  endif
F31= 3.1415927
```
CAM/Status Variations:

Directly ties in the CAM/STATUS command to an IF-Else-Endif conditional branch.

After a cam download or after the CAM/BUILD command, it takes some time to calculate and store the new cam. When a cam is activated on the CLC, it does not run until the cam is at the end of its cycle.

IF CAM/Status checks the status of a specified cam. The test is limited to == (equivalent to) or != (not equal) arguments.

The status integer returns the specified cam condition:

Errors will be issued when:
The cam number is not valid (out of range or drive is not configured).

Example:

```
; CAM/Status variations
    if cam/status 1 != 4
        F30 = 5
    endif
    if cam/status ( 1) == 4
        F30 = 5
    endif
    if (CAM/STATUS ( 1) == 4 )
        F30 = 5
    endif
```
7.3.37. KINEMATIC (Use a Kinematic Definition for a Task)

Syntax:

KINEMATIC kinematic

where:

<table>
<thead>
<tr>
<th>argument</th>
<th>valid type(s)</th>
<th>range</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kinematic</td>
<td>integer</td>
<td>a valid kinematic</td>
<td>library number</td>
</tr>
<tr>
<td></td>
<td>- constant</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- label</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Variables or labels used for arguments must equate to valid values at run-time or an error will result.

Description:
Each task can have its own kinematic to allow for motion in Cartesian space. This instruction tells the path planner which set of equations in the optional kinematic library to use for motion.

This instruction is active at the start of a Task, and is removed before normal system cycling.

The KINEMATIC statement is allowed only in the main Tasks: A, B, C, or D.

Example:

```
kinematic 10 ; Use kinematic library number 10
```
7.3.38. LOCAL/VARIABLE

Syntax:

LOCAL/VARIABLE label, type

where:

<table>
<thead>
<tr>
<th>argument</th>
<th>valid type(s)</th>
<th>range</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>label</td>
<td>ASCII string</td>
<td>1 to 20 characters</td>
<td>name of variable used as local variable</td>
</tr>
<tr>
<td>type</td>
<td>ASCII character</td>
<td>‘I’ = integer  ‘F’ = floating point  ‘ABS’ = absolute point index  ‘REL’ = relative point index</td>
<td>type of variable used as local variable</td>
</tr>
</tbody>
</table>

Description:
Declares a local variable including its name and type. A total of sixteen function arguments and local variables may be used per function.

Example:

```
; Main task calling subroutine and returning value
Task_A: TASK/START A
   I1 = CALL sub, 5, 10, 20
   TASK/END A

; Subroutine to multiply input values
sub: FUNCTION/START U
   FUNCTION/ARG COUNT1, I, 1, 300
   FUNCTION/ARG COUNT2, I, 1, 300
   FUNCTION/ARG COUNT3, I, 1, 300
   LOCAL/VARIABLE TAB, I
   TAB = (COUNT1 * COUNT2) * COUNT3
   FUNCTION/END TAB
```
7.3.39. MESSAGE/DIAG (Task Diagnostic Message Definition)

Syntax:

MESSAGE/DIAG message

where:

<table>
<thead>
<tr>
<th>argument</th>
<th>valid type(s)</th>
<th>range</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>message</td>
<td>ASCII</td>
<td>up to 79 characters</td>
<td>diagnostic text message</td>
</tr>
</tbody>
</table>

* VM 5.0 {MESSAGE type,"text", [variable]}. Types are: 1 = status, 2 = diagnostic
  Text must be in quotes. Messages may have one formatted variable in its string. Use "%s" to format optional variable. Previous syntax supported for upward compatibility.

Description:

The MESSAGE/DIAG directive provides a user program with the ability to send a diagnostic message to the user that is associated with a specific statement in the user program. MESSAGE/DIAG embeds a tag or index into the instruction following the MESSAGE/DIAG statement.

A diagnostic message provided by the CLC can be used by the user interface to select and display the message for system maintenance or troubleshooting.

The diagnostic message may also be used for program debugging during program development in a manner similar to embedding print statements at critical points in the program.

Example:

```
Move/line  ABS[1]
Message 2, "Waiting for safe zone bit in PLC to be true"
;
;The diagnostic message lets the user know that the program has advanced
;to the program statements that check the safe zone I/O bit.
;
L10:    Plc/read       I01, 0
If I01 & 0x100
    Goto L20
Delay       1000
    Goto       L10
EndIf
L20:
```


7.3.40. MESSAGE/STATUS (Task Status Message Definition)

Syntax:

MESSAGE/STATUS    message

where:

<table>
<thead>
<tr>
<th>argument</th>
<th>valid type(s)</th>
<th>range</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>message</td>
<td>ASCII</td>
<td>up to 79 characters</td>
<td>status text message</td>
</tr>
</tbody>
</table>

* VM 5.0 {MESSAGE type,"text", [variable]}. Types are: 1=status, 2=diagnostic
  
  Text must be in quotes. Messages may have one formatted variable in its string. Use "%s" to format optional variable. Previous syntax supported for upward compatibility.

Description:

The MESSAGE/STATUS directive is similar to the MESSAGE/DIAG directive. MESSAGE/STATUS also embeds a tag or index into the instruction following the MESSAGE/STATUS statement. However, MESSAGE/STATUS character strings are stored in a different table than MESSAGE/DIAG messages and provide a different series of message index numbers.

By providing two tables, status messages may be used for prompting the system operator, diagnostic messages for program debugging and system maintenance.

Example:

```
Move/line    ABS[1]
Message 1, "Waiting for safe zone bit in PLC to be true"
;The status message lets the user know that the program has advanced
;to the program statements that check the safe zone I/O bit.
L10: Plc/read    I01, 0
    If I01 & 0x100
      Goto L20
    Delay    1000
    Goto    L10
    EndIf
L20:
```

7.3.41. MOVE/CIRCLE (Coordinated Move with Circular Interpolation)

Syntax:

MOVE/CIRCLE  ABS[index1], ABS[index2]

or,

MOVE/CIRCLE  REL[index1], REL[index2]

where:

<table>
<thead>
<tr>
<th>argument</th>
<th>valid type(s)</th>
<th>range</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>index1</td>
<td>integer</td>
<td>a valid absolute or relative point table entry</td>
<td>defines the mid-point of the circular arc</td>
</tr>
<tr>
<td></td>
<td>- constant</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- variable Ix</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- global variable GIx</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- label</td>
<td></td>
<td></td>
</tr>
<tr>
<td>index2</td>
<td>integer</td>
<td>a valid absolute or relative point table entry</td>
<td>defines the end-point of the circular arc</td>
</tr>
<tr>
<td></td>
<td>- constant</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- variable Ix</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- global variable GIx</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- label</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Variables or labels used for arguments must equate to valid values at run-time or an error will result.

**Description:**

The MOVE/CIRCLE instruction provides circular motion along a path in Cartesian space defined by three sets of coordinates. Program modifiable integer variables may be used to specify the starting and ending coordinates in the point tables.

The first form provides circular movement using absolute coordinates. Motion occurs from the end point of the last move or current position, through the absolute coordinate specified by the table reference of the first argument, and ends at the absolute coordinate specified by the table reference of the second argument.

The second form allows motion to begin at a relative offset from the end of the last move or current position, moves through a relative offset specified by the table reference of the second argument, and ends at an absolute coordinate specified by the table reference of the third argument.
Example:

; enter points into absolute point table
ABS[0]x = 0.0
ABS[1]y = 1.0
ABS[1]z = 0.0

; assume the current position is -1.0, 0.0, 0.0
MOVE/CIRCLE ABS[1], ABS[2]
; the move results in a semi-circular move
; from x=-1, y=0, through x=0, y=1; to x=1, y=0
7.3.42. MOVE/JOINT (Coordinated Move Joint Point to Point)

Syntax:

MOVE/JOINT ABS[index]

where:

<table>
<thead>
<tr>
<th>argument</th>
<th>valid type(s)</th>
<th>range</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>integer</td>
<td>a valid absolute point table entry</td>
<td>specifies the endpoint of move</td>
</tr>
<tr>
<td></td>
<td>- constant</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- variableIx</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- global variable GIx</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- label</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Variables or labels used for arguments must equate to valid values at run-time or an error will result.

Description:

MOVE/JOINT is an absolute point-to-point move, with only the endpoint of the move specified. The actual path taken to the specified point is indeterminent (i.e., not linear or circular) and may assume whatever form the path planner requires; however, once programmed and planned, the path is repeatable. The path is optimized to minimize time and uses accel and slew rates for the coordinated axes (as opposed to the line and circle coordinated motion commands which use the world rates).

Most commonly used with robotic applications, the move Joint command is also the only method of elbow repositioning.

Example:

```
ABS[1].X = 10.0 ; \     
ABS[1].Y = 20.0 ; --> Sets Absolute point 1
ABS[1].Z = 15.0 ; /     

MOVE/JOINT ABS[1] ; An absolute point-to-point
; Coordinated motion
```
7.3.43. MOVE/LINE (Coordinated Move with Straight Line Interpolation)

Syntax:

MOVE/LINE      ABS[index]

or,

MOVE/LINE      REL[index]

where:

<table>
<thead>
<tr>
<th>argument</th>
<th>valid type(s)</th>
<th>range</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>integer</td>
<td>a valid absolute or relative point table entry</td>
<td>specifies the endpoint of move</td>
</tr>
<tr>
<td></td>
<td>- constant</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- variable Ix</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- global variable GIx</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- label</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Variables or labels used for arguments must equate to valid values at run-time or an error will result.

Description:

The MOVE/LINE instruction provides motion along a straight line path in Cartesian space, defined by two sets of coordinates.

The first form provides linear movement to absolute coordinates. Motion occurs from the end point of the last move or current position, and ends at the absolute coordinate specified by the table reference of the last argument.

The second form begins motion at a relative offset from the end of the last move or current position, and ends at an absolute coordinate specified by the table reference of the last argument.

Note that program modifiable integer variables may be used to specify the starting and ending coordinates in the point tables.
Example:

; assume that the current position is -1.0, 0.0, 0.0

Move/Line      ABS[1]
; This results in a linear move from x=-1, y=0; to x=0, y=1
### 7.3.44. PARAMETER/BIT (Initialize Parameter Bit)

**Syntax:**

```
PARAMETER/BIT type, set, param, source, bit
```

where:

<table>
<thead>
<tr>
<th>argument</th>
<th>valid type(s)</th>
<th>range</th>
<th>description</th>
</tr>
</thead>
</table>
| type     | ASCII character | A, C, D, T | Parameter Type  
- A for Axis parameters  
- C for System parameters  
- D for Drive parameters  
- T for Task parameters |
| set      | integer, or  
label | 1, Ix, I[x], GIx,  
GI[x] | Parameter Set  
1 for System parameters  
a valid drive or axis number for  
Drive or Axis parameters  
A, B, C, or D character for  
Task parameters |
| param    | integer | a valid  
parameter | parameter ID number |
| source   | integer  
- constant  
- variable  
Ix, I[x], GIx, GI[x]  
- global variable  
Ix, I[x], GIx, GI[x]  
- label | zero  
or,  
non-zero | logical value for bit  
- zero value clears the bit  
- non-zero sets the bit |
| bit      | HEX format  
or, label | decimal  
1 - 16, or  
hexadecimal,  
(0X0001 to 0xFFFF) | decimal selects a single bit  
hexadecimal can select multiple  
bits  
e.g.,  
0x0300 = bits 9 and 10  
0x2003 = bits 1,2, and 14 |

Variables or labels used for arguments must equate to valid values at run-time or an error will result.
Description:
PARAMETER/BIT is used to set one or more bits in a parameter when the program is activated. The instruction tests for a zero/non-zero logical constant or variable to set or clear the bit(s) enabled by a specified bit mask.

Example:

```
TASK_A: TASK/START A
PLC/CLEAR 120, 8 ; Clears bit 8 of register 120
.

PARAMETER/BIT A, 1, 4, I1, 0x0002
; Sets Rotary mode for axis 1
.

AXIS/RATIO 1, 2, 2, 1 ; Axis 2 is linked to axis 1 at 1/2
.
.
TASK/END A
```
7.3.45. PARAMETER/GET (Load Parameter to a Variable)

Syntax:

PARAMETER/GET type, set, param, target

where:

<table>
<thead>
<tr>
<th>argument</th>
<th>valid type(s)</th>
<th>range</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>ASCII character</td>
<td></td>
<td>Type of parameter</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A, C, D, T</td>
<td>- A for Axis parameters</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- C for System parameters</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- D for Drive parameters</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- T for Task parameters</td>
</tr>
<tr>
<td>set</td>
<td>integer, or label</td>
<td>1</td>
<td>Parameter set</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ix, I[x], G1x, G1[x]</td>
<td>- 1 for System parameters</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- a valid drive or axis number</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- for Drive or Axis parameters</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- A, B, C, or D character for Task</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A, B, C or D</td>
<td>parameters</td>
</tr>
<tr>
<td>param</td>
<td>integer</td>
<td>a valid parameter</td>
<td>parameter ID number</td>
</tr>
<tr>
<td>target</td>
<td>integer or floating point</td>
<td>a valid variable</td>
<td>target variable to load with</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>parameter value</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>target variable data type must</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>match parameter data type</td>
</tr>
</tbody>
</table>

Variables or labels used for arguments must equate to valid values at run-time or an error will result.

**Description:**
Retrieves a parameter from the CLC’s system, task or drive parameter tables and stores the value in the specified variable.

Incorrectly specifying the parameter type will result in a compiler error.
Example:

```
Parameter/Get D, 1, STATUS, I01
If I01 == 0
    Goto L10
Else
    Gosub new_sub
Endif

L10: Move/Line ABS[2]
    I01 = 100
    Parameter/Set D, 1, POSITION, I01
```
7.3.46. PARAMETER/INIT (Initialize a Parameter)
Syntax:

PARAMETER/INIT type, set, param, source

where:

<table>
<thead>
<tr>
<th>argument</th>
<th>valid type(s)</th>
<th>range</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>ASCII character</td>
<td></td>
<td>Type of parameter</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A, C, D, T</td>
<td>A for Axis parameters</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- C for System parameters</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- D for Drive parameters</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- T for Task parameters</td>
</tr>
<tr>
<td>set</td>
<td>integer, or label</td>
<td>1</td>
<td>Parameter set</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ix,I[x],Glx,Gl[x]</td>
<td>1 for System parameters</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- a valid drive or axis number for Drive or Axis parameters</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A, B, C or D</td>
<td>- A, B, C, or D character for Task parameters</td>
</tr>
<tr>
<td>param</td>
<td>integer</td>
<td>a valid parameter</td>
<td>parameter ID number</td>
</tr>
<tr>
<td>source</td>
<td>integer or floating point</td>
<td>a valid variable</td>
<td>source variable to load to parameter</td>
</tr>
<tr>
<td></td>
<td>- constant</td>
<td></td>
<td>source variable data type must match parameter data type</td>
</tr>
<tr>
<td></td>
<td>- variable Fx,F[x]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- global variable GFx,GF[x],GIx,GI[x]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- label</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Description:
PARAMETER/INIT is used to load the source value into the specified parameter when the program is activated.

Example:

```
PARAMETER/INIT A, 1, 120, 32.0
; Sets axis 1 parameter 120 to
; initialize acceleration
```
7.3.47. PARAMETER/SET (Set a Parameter)

Syntax:

PARAMETER/SET type, set, param, source

where:

<table>
<thead>
<tr>
<th>argument</th>
<th>valid type(s)</th>
<th>range</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>ASCII character</td>
<td>A, C, D, T</td>
<td>Type of parameter</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- A for Axis parameters</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- C for System parameters</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- D for Drive parameters</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- T for Task parameters</td>
</tr>
<tr>
<td>set</td>
<td>integer, or label</td>
<td>1, A, B, C</td>
<td>Parameter set</td>
</tr>
<tr>
<td></td>
<td></td>
<td>or label</td>
<td>- 1 for System parameters</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- a valid drive or axis number for Drive or Axis parameters</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- A, B, C, or D character for Task parameters</td>
</tr>
<tr>
<td>param</td>
<td>integer</td>
<td>a valid</td>
<td>parameter ID number</td>
</tr>
<tr>
<td></td>
<td></td>
<td>parameter</td>
<td></td>
</tr>
<tr>
<td>source</td>
<td>integer or floating point</td>
<td>a valid</td>
<td>source variable to load to parameter</td>
</tr>
<tr>
<td></td>
<td></td>
<td>variable</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>source variable data type must match parameter data type</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Description:
PARAMETER/SET updates the CLC’s system, task or drive parameter table with a new value for the specified parameter.

Incorrectly specifying the parameter type will result in a compiler error.
Example:

```
Parameter/Get D, 1, STATUS, I01
If I01 == 0 Goto L10
Else
    Gosub New_Subroutine
    I01 = 100
L10: Move/Line ABS[2]
    I01 = 100
Parameter/Set d, 1, POSITION, I01
```

7.3.48. PATH/ABORT (Aborts Coordinated Motion)

Syntax:

PATH/ABORT task

where:

<table>
<thead>
<tr>
<th>argument</th>
<th>valid type(s)</th>
<th>range</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task</td>
<td>ASCII character</td>
<td>A, B, C or D</td>
<td>task ID character</td>
</tr>
</tbody>
</table>

Variables or labels used for arguments must equate to valid values at run-time or an error will result.

Description:

PATH/ABORT decelerates motion on path then aborts all segments placed in the task’s queue by the path planner. Motion must be restarted with a cycle start, it cannot be continued. Any events are lost.

Example:

```plaintext
TASK_D:TASK/START D

REL[1].X = 1.0 ; Sets relative point 1

IF (I1 == 3)
  PATH/ABORT D ; Aborts task D operation
```
7.3.49. PATH/POSITION (Get Current Path Absolute Position)

Syntax:

PATH/POSITION    task, ABS[index]

where:

<table>
<thead>
<tr>
<th>argument</th>
<th>valid type(s)</th>
<th>range</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task</td>
<td>ASCII character</td>
<td>A, B, C or D</td>
<td>task ID</td>
</tr>
<tr>
<td>index</td>
<td>integer</td>
<td>a valid absolute point table entry</td>
<td>index to a target absolute position table entry</td>
</tr>
<tr>
<td></td>
<td>- constant</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- variable Ix</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- global variable GIx</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- label</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Variables or labels used for arguments must equate to valid values at run-time or an error will result.

Description:
The current position of the path planner is returned to the specified point table entry. The current contents of the point table are overwritten. The other current specifications in the point table (rate, accel/decel, events, etc.) are not affected.

One task may obtain the position of another task by specifying the desired task ID.

Example:

L10:  path/position 1,ABS[6]    ; Get current position
     F[01] = ABS[6].x*ABS[6].x
     F[01] = F[01] + ABS[6].z*ABS[6].z
     F[01] = SQRT( F[01] );               ; Compute vector length
     if F[01] >= 100.0 the
        goto L10 ; Loop if larger than 100

NOTE:  Add one more point to this instruction. For example:

ABS[x] defined in Path/Position instruction =
# of absolute points defined in Data/Size +1
7.3.50. PATH/RESUME (Resume Coordinated Motion)

Syntax:

PATH/RESUME task

where:

<table>
<thead>
<tr>
<th>argument</th>
<th>valid type(s)</th>
<th>range</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task</td>
<td>ASCII character</td>
<td>A, B, C or D</td>
<td>task ID character</td>
</tr>
</tbody>
</table>

Description:
PATH/RESUME continues coordinated motion halted by a PATH/STOP command. Any queued time or distance-based events saved by the PATH/STOP will also resume when motion continues.

Example:

```
TASK_C:TASK/START C
.
PATH/STOP A ; Stops Coordinated motion of task A
DELAY 3000 ; Waits for 3 seconds
PATH/RESUME A ; Resumes Coordinated motion of task A
.
```
7.3.51. PATH/STOP (Halt Coordinated Motion)

Syntax:

PATH/STOP task

where:

<table>
<thead>
<tr>
<th>argument</th>
<th>valid type(s)</th>
<th>range</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task</td>
<td>ASCII character</td>
<td>A, B, C or D</td>
<td>task ID character</td>
</tr>
</tbody>
</table>

Description:
PATH/STOP commands the path planner to decelerate motion and halt motion on the path. PATH/STOP saves the "look-ahead" path planned by the path planner and any associated time or distance-based events. Motion and events on the path may be resumed by a PATH/RESUME command.

Example:

SAMPLE_RATE EQU 0.0 ; Sets equate
.
.
TASK_A:TASK/START A
.
.
IF (SAMPLE_RATE >= 1000.0)
    PATH/STOP B ; Halts Coordinated motion of task B
.
.
TASK/END A
7.3.52. PATH/WAIT (Pause Program for Motion)
Syntax:

PATH/WAIT            task, ABS[index], state

or,

PATH/WAIT            task, REL[index], state

where:

<table>
<thead>
<tr>
<th>argument</th>
<th>valid type(s)</th>
<th>range</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task</td>
<td>ASCII character</td>
<td>A, B, C or D</td>
<td>task ID</td>
</tr>
<tr>
<td>index</td>
<td>integer</td>
<td>a valid absolute or relative point table entry</td>
<td>point table entry specifying position to pause program</td>
</tr>
<tr>
<td></td>
<td>- constant</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- variable Ix</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- global variable GIx</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- label</td>
<td></td>
<td></td>
</tr>
<tr>
<td>state</td>
<td>integer</td>
<td>0 to 8</td>
<td>requested state of path planner</td>
</tr>
<tr>
<td></td>
<td>- label</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Variables or labels used for arguments must equate to valid values at run-time or an error will result.

Description:
PATH/WAIT tests the current state of the path planner for a specified point. Since the path planner is typically one or more segments ahead of physical motion, PATH/WAIT can be used to temporarily halt program execution until the path planner begins a specific type of processing for a specified point.
Example:

TASK_D: TASK/START D

MOVE/LINE ABS[1] ; Straight Line Coordinated motion
PATH/WAIT D, ABS[1], 8 ; Waits until point is done

TASK/END D

7.3.53. PID/CONFIGURE

PID/CONFIGURE #, type, control_register, status_register, loop_time, set_point_type, set_point, set_point_axis, feedback_type, feedback, feedback_axis, output_type, output, output_axis, control_block

Example:

PID/CONFIGURE 1, 1.130, 131, 8, 1, F5, 0, 1, F6, 0, 1, F7, 0, F10

See also Section
7.3.70. VAR/INIT
7.3.54. PLC/CLEAR (Clear I/O Register Bit)

Syntax:

PLC/CLEAR register, bit

where:

<table>
<thead>
<tr>
<th>argument</th>
<th>valid type(s)</th>
<th>range</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>register</td>
<td>integer</td>
<td>a valid I/O register</td>
<td>specifies an I/O register</td>
</tr>
<tr>
<td></td>
<td>- constant</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- label</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bit</td>
<td>integer</td>
<td>1 to 16</td>
<td>specifies a bit in the register</td>
</tr>
<tr>
<td></td>
<td>- constant</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- label</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Variables or labels used for arguments must equate to valid values at run-time or an error will result.

Description:
PLC/CLEAR clears the specified bit in the specified I/O register table entry to 0.

Example:

```
Cycle_On equ 10 ; Cycle on register
Go equ 6 ; Bit for go
Plc/Clear Cycle_On, Go ; Clear bit 6 of cycle on
```

.
7.3.55. PLC/READ (Read I/O Register(s))

Syntax:

PLC/READ register, count, target

where:

<table>
<thead>
<tr>
<th>argument</th>
<th>valid type(s)</th>
<th>range</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>register</td>
<td>integer</td>
<td>a valid I/O register</td>
<td>first register to read</td>
</tr>
<tr>
<td></td>
<td>- constant</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- label</td>
<td></td>
<td></td>
</tr>
<tr>
<td>count</td>
<td>integer</td>
<td>1 to maximum number of I/O registers</td>
<td>number of registers to read</td>
</tr>
<tr>
<td></td>
<td>- constant</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- label</td>
<td></td>
<td></td>
</tr>
<tr>
<td>target</td>
<td>integer</td>
<td>a valid integer variable</td>
<td>first target variable</td>
</tr>
<tr>
<td></td>
<td>- variable Ix, GIx, I[x], GI[x]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- label</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Variables or labels used for arguments must equate to valid values at run-time or an error will result.

Description:
PLC/READ copies the contents of one or more sequential 16-bit registers in the CLC I/O registers table to sequential entries in the integer variable table.

Example:

```plaintext

iobase equ 100
count equ 20
define I5,ioregs
;
plc/read iobase, count, ioregs ;Read the 20 CLC I/O registers, from
;100 through 120 to integer variables
;I05 through I25

```

This statement copies the contents of twenty 16 bit PLC I/O table entries (I/O registers 101 through and including 120) to the integer variables I01 through and including I20.
7.3.56. PLC/SET (Set I/O Register Bit)
Syntax:

PLC/SET register, bit

where:

<table>
<thead>
<tr>
<th>argument</th>
<th>valid type(s)</th>
<th>range</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>register</td>
<td>integer</td>
<td>a valid I/O register</td>
<td>specifies an I/O register</td>
</tr>
<tr>
<td></td>
<td>- constant</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- label</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bit</td>
<td>integer</td>
<td>1 to 16</td>
<td>specifies a bit in the register</td>
</tr>
<tr>
<td></td>
<td>- constant</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- label</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Variables or labels used for arguments must equate to valid values at run-time or an error will result.

Description:
PLC/SET sets a specified bit in the specified CLC I/O register table to 1.

Example:

```
Light_Reg equ 20
Light_On equ 5
.
.
plc/set Light_Reg, Light_On ; Turn on bit 5 of register 20
.
.
```
7.3.57. PLC/TEST (Test I/O Register Bit)

Syntax:

PLC/TEST register, bit, target

where:

<table>
<thead>
<tr>
<th>argument</th>
<th>valid type(s)</th>
<th>range</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>register</td>
<td>integer</td>
<td>a valid I/O register</td>
<td>source register to test</td>
</tr>
<tr>
<td></td>
<td>- constant</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- label</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- variable Ix, GIx</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bit</td>
<td>integer</td>
<td>1 to 16</td>
<td>bit within register to test</td>
</tr>
<tr>
<td></td>
<td>- constant</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- label</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- variable Ix, GIx</td>
<td></td>
<td></td>
</tr>
<tr>
<td>target</td>
<td>integer</td>
<td>a valid integer variable</td>
<td>variable to receive binary (1 or 0) test result</td>
</tr>
<tr>
<td></td>
<td>- variable Ix, I[x], GIx, GI[x]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- label</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Variables or labels used for arguments must equate to valid values at run-time or an error will result.

Description:
PLC/TEST tests a bit in the specified I/O register table entry and returns a logical value to the specified integer variable according to the state of the I/O register bit. (The value of the integer variable may then be used to control program flow.)

Example:

```
Plc/Test 20, 5, I03 ;set I03 flag to I/O register, bit 5 state
If I03 > 0 Gosub calculate ;do calculate if flag is set
Else
gosub subtract
EndIf
```

...
7.3.58. PLC/WAIT (Pause Program for I/O)

Syntax:

PLC/WAIT register, bit, state

where:

<table>
<thead>
<tr>
<th>argument</th>
<th>valid type(s)</th>
<th>range</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>register</td>
<td>integer - constant - label</td>
<td>a valid I/O register</td>
<td>source register to test</td>
</tr>
<tr>
<td>bit</td>
<td>integer - constant - label</td>
<td>1 to 16</td>
<td>bit within register to test</td>
</tr>
<tr>
<td>state</td>
<td>integer - variable Ix, I[x], GIx, GI[x]</td>
<td>zero or one</td>
<td>variable containing the logical value required to resume program execution</td>
</tr>
</tbody>
</table>

Variables or labels used for arguments must equate to valid values at run-time or an error will result.

Description:

PLC/WAIT may be used to pause program execution with a task until the state of the specified I/O bit equals the specified logical value.
7.3.59. PLC/WRITE (Write to I/O Register(s))

Syntax:

PLC/WRITE register, count, source

where:

<table>
<thead>
<tr>
<th>argument</th>
<th>valid type(s)</th>
<th>range</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>register</td>
<td>integer - constant - label</td>
<td>a valid I/O register</td>
<td>starting target register to write</td>
</tr>
<tr>
<td>count</td>
<td>integer - constant - label</td>
<td>1 to maximum number of registers</td>
<td>number of registers to write</td>
</tr>
<tr>
<td>source</td>
<td>integer - constant - variable Ix, I[x], GIx, GI[x] - label</td>
<td>a valid integer variable</td>
<td>starting source variable</td>
</tr>
</tbody>
</table>

Variables or labels used for arguments must equate to valid values at run-time or an error will result.

Description:
PLC/WRITE copies the contents of one or more sequential integer variables from the integer variable table to the I/O register table.

Example:

I[01] = 0x0001
I[02] = 0x6666
.
.
.
Plc/Write 2, 3, I[01]
.
.
.
This statement writes integer variable I[01] to I/O register 3, and I[02] to I/O register 4.
7.3.60. PLS/INIT (Compiler directive)

Syntax:

PLS/INIT switch, 0, register, type, axis, offset (PRIMARY)
   or
switch, element, on position, off position (SECONDARY)

where:

<table>
<thead>
<tr>
<th>argument</th>
<th>valid type(s)</th>
<th>range</th>
<th>description</th>
</tr>
</thead>
</table>
| switch      | integer
   - constant
   - label    | 1            |                                                            |
| register    | integer
   - constant
   - label    | output register |                                                            |
| type *      | integer
   - constant
   - variable Ix, | 1 - ELS
   2 - virtual axis
   3 - primary axis
   4 - secondary axis | type of axis that the
   limit switch is tracking |
| axis        | 0 if type 1 or 2
   else 1-40 | axis number |                                                            |
| offset      | 0-360                  | PLS phase advance |                                                            |

or

<table>
<thead>
<tr>
<th>argument</th>
<th>valid type(s)</th>
<th>range</th>
<th>description</th>
</tr>
</thead>
</table>
| switch      | integer
   - constant
   - label    | 1            |                                                            |
| element     | integer
   - constant
   - label    | 1-16         | switch element outputs (output register bits) |
| on position | integer
   - constant
   - variable Ix, | 0-360       | position in which switch element turns on |
| off position| integer
   - constant
   - variable Ix | 0-360       | position in which switch element turns off |

* Currently ELS is the only valid 'type'.

Description:
Both the primary and secondary PLS setup commands are executed by Visual Motions compiler to initialize the PLS table. The location of the PLS/INIT command is not important because only the Visual Motion compiler uses it. Use the runtime commands (PLS[1].t=1) to change the table in the program flow. The PLS is disabled while the output register is zero or when all bits have zero in both the on and off positions.

A programmable limit switch supports 16 outputs and a phase advance. The position input for the PLS is the ELS master, which can be an external encoder, an axis feedback or a virtual master. The outputs are updated every SERCOS cycle.

Example:
Task_A:TASK/START A
  local/variable  var1, I
  data/size 50, 50, 20, 20, 10, 0
  ;Primary PLS setup command
  pls/init 1,0,120,1,0,0.0
  ;Secondary PLS setup command
  pls/init 1,1,20,40
  pls/init 1,2,20,40
  pls/init 1,3,20,40
  pls/init 1,4,20,40
  pls/init 1,5,20,40
  pls/init 1,6,20,40
  pls/init 1,7,20,40
  pls/init 1,8,20,40
  pls/init 1,9,20,40
  pls/init 1,10,20,40
  pls/init 1,11,20,40
  pls/init 1,12,20,40
  pls/init 1,13,20,40
  pls/init 1,14,20,40
  pls/init 1,15,20,40
  pls/init 1,16,20,40
  ;Runtime commands
  PLS[1].r=120 ;register
  PLS[1].o=30.0 ;offset
  PLS[1].on1=30
  PLS[1].on2=30
  PLS[1].on16=120
  PLS[1].off1=120
  PLS[1].off16=300
  var1 =call loop25
  TASK/END A
.
7.3.61. RETURN (Return From Subroutine)

Syntax:

RETURN

Description:
A RETURN statement must be entered as the last instruction of a subroutine. When the subroutine execution encounters a RETURN statement, program flow exits the subroutine and resumes with the program statement following the GOSUB statement that called the subroutine.

Example:

. .
Sub1: Move/Line ABS[2]
    Plc/Set 1,1
    Return
    . .
7.3.62. ROBOT/ORIGIN

Syntax:

ROBOT/ORIGIN      REL[index]

where:

<table>
<thead>
<tr>
<th>argument</th>
<th>valid type(s)</th>
<th>range</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>integer</td>
<td>a valid relative point table entry</td>
<td>defines the zero frame of reference for a robot</td>
</tr>
<tr>
<td></td>
<td>- constant</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- variable Ix</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- global variable GIx</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- label</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Variables or labels used for arguments must equate to valid values at run-time or an error will result.

Description:

The ROBOT/ORIGIN instruction is used in coordinated motion programs to construct a zero frame of reference from the x, y, z, roll, pitch and yaw coordinates of a relative point. This moves the effective origin of the robot from the default to the location specified by the programmed relative point.

For example, if REL[3] = {1, 2, 3, 0, 0, 0, …} and this point is specified as the robot origin, then the robot origin would be offset by one unit along the x axis, two units along the y axis and three units along the z axis. Once the instruction is executed, all jogging, teaching and path locations are affected by the new origin.
7.3.63. ROBOT/TOOL

Syntax:

ROBOT/TOOL REL[index]

where:

<table>
<thead>
<tr>
<th>argument</th>
<th>valid type(s)</th>
<th>range</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>integer</td>
<td>a valid relative point table entry</td>
<td>defines the tool frame of reference for a robot</td>
</tr>
</tbody>
</table>

- constant
- variable Ix
- global variable GIx
- label

Variables or labels used for arguments must equate to valid values at run-time or an error will result.

Description:
The ROBOT/TOOL instruction is used in coordinated motion programs to construct a tool frame of reference from the x, y, z, roll, pitch and yaw coordinates of a relative point. This moves the effective end-of-arm tool to the location specified by the programmed relative point.

For example, if REL[4] = {0, 0, 10, 0, 0, 0, …} and this point is specified as the robot tool location, then the robot tool location would be offset by ten units along the z axis from the faceplate of the robot. Once the instruction is executed, all jogging, teaching and path locations are affected by the end-of-arm tool location.
7.3.64. SEQUENCER

Syntax:

SEQUENCER  seq_name

where:

<table>
<thead>
<tr>
<th>argument</th>
<th>valid type(s)</th>
<th>range</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>seq_Name</td>
<td>integer</td>
<td>determined by program limits</td>
<td>sequencer name</td>
</tr>
<tr>
<td></td>
<td>- constant</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- variable Ix</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- global variable GIx</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- label</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Variables or labels used for arguments must equate to valid values at run-time or an error will result.

Description:
The sequencer name is a number or label equating to a number. The number has a range of 1 to n, where n is the number of sequencers defined in the Size icon.

Example:

```
; .
; .
; Run sequencer IVORY
SEQUENCER  IVORY
TASK/END  A
```
7.3.65. SEQ/LIST

Syntax:

SEQ/LIST  seq_name, list_number, list_name
           or
           seq_name, 0, count

where:

<table>
<thead>
<tr>
<th>argument</th>
<th>valid type(s)</th>
<th>range</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>seq_Name</td>
<td>integer - constant -</td>
<td>determined by program</td>
<td>sequencer name</td>
</tr>
<tr>
<td></td>
<td>variable Ix - global</td>
<td>limits</td>
<td></td>
</tr>
<tr>
<td></td>
<td>variable GIx - label</td>
<td></td>
<td></td>
</tr>
<tr>
<td>list_Number</td>
<td>integer</td>
<td>valid list number or 0</td>
<td>number of step list</td>
</tr>
<tr>
<td></td>
<td>for count</td>
<td></td>
<td></td>
</tr>
<tr>
<td>list_Name</td>
<td>label or count (integer)</td>
<td></td>
<td>name of step list or if range=0 this is count</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(total number of lists)</td>
</tr>
</tbody>
</table>

Variables or labels used for arguments must equate to valid values at run-time or an error will result.

Description:

This instruction assigns a sequencer step list to an existing sequencer along with a number which determines the order the sequencer will follow when executed. The name of the step list is also included in this command.

A step list can be used by more than one sequencer. It can also be used repeatedly within a given sequencer.

Example:

IVORY equ 1
Task A: TASK/START A
; Declare 10 sequencers, 10 steps, 50 function steps
  DATA/SIZE 50, 50, 0, 0, 4, 0, 10, 10, 50
; Build sequencer IVORY
; Sequencer IVORY will have 5 steps
SEQ/LIST IVORY, 0, 5
SEQ/LIST IVORY, 1, Mold_Open
SEQ/LIST IVORY, 2, Move_In
SEQ/LIST IVORY, 3, Grab_Part
SEQ/LIST IVORY, 4, Move_to_Drop
SEQ/LIST IVORY, 5, Drop_Part
.....
7.3.66. SEQ/STEP

Syntax:

SEQ/STEP   list_name, step_number, function_name, arg1, arg2, ...arg5
or
list_name, 0, count

where:

<table>
<thead>
<tr>
<th>argument</th>
<th>valid type(s)</th>
<th>range</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>list_Name</td>
<td>label</td>
<td></td>
<td>name of step list</td>
</tr>
<tr>
<td>step_Number</td>
<td>integer or 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>function_Name</td>
<td>label or count</td>
<td>any valid function</td>
<td>name of function or if step_number=0 this is count (the total number of steps)</td>
</tr>
<tr>
<td>or count</td>
<td>label or count (integer)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>arg1, arg2, ...arg5</td>
<td>integer, float - constant</td>
<td>determined by program limits</td>
<td>function arguments</td>
</tr>
</tbody>
</table>

Variables or labels used for arguments must equate to valid values at run-time or an error will result.

Description:
This command defines a sequencer step within a step list. It identifies an existing function or subroutine and passes on up to five function arguments when executed. If the given step_number is

Example:

; Step Mold_Open will have 4 functions in it
SEQ/STEP    Mold_Open,0,4
SEQ/STEP    Mold_Open,1,vac_on,1,,
SEQ/STEP    Mold_Open,2,waitcool,1000,,
SEQ/STEP    Mold_Open,3,chk_part,,
SEQ/STEP    Mold_Open,4,move_rdy,,

; Step Move_In will have 4 functions in it
SEQ/STEP    Move_In,0,4
SEQ/STEP    Move_In,1,chk_mold,,
SEQ/STEP    Move_In,2,movechk,1,1000,,
SEQ/STEP    Move_In,3,inmoldms,100,555,,
SEQ/STEP    Move_In,4,vac_on,4,,

; Step Drop_Part will have 1 function in it
SEQ/STEP    Drop_Part,0,1
SEQ/STEP    Drop_Part,1,f1234567890123456789,,
;
7.3.67. TASK/AXES (Task Axes Definition)
Syntax:

```
TASK/AXES type, axis1, {axis2, axis3}
```

where:

<table>
<thead>
<tr>
<th>argument</th>
<th>valid type(s)</th>
<th>range</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>integer</td>
<td>1 to 6</td>
<td>type of motion</td>
</tr>
<tr>
<td></td>
<td>- label</td>
<td></td>
<td>1 - single axis motion</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2 - coordinated motion</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3 - velocity mode (no positioning)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4 - for ratioed axes (master/slave)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>5 - ELS Slave mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>6 - Torque Mode</td>
</tr>
<tr>
<td>axis1</td>
<td>integer</td>
<td></td>
<td>number of valid axes</td>
</tr>
<tr>
<td>axis2</td>
<td>integer</td>
<td>1 to the maximum</td>
<td>number of valid axes</td>
</tr>
<tr>
<td>axis3</td>
<td>integer</td>
<td>number of valid</td>
<td>number of valid axes</td>
</tr>
<tr>
<td></td>
<td>- constant</td>
<td>number of valid</td>
<td>number of valid axes</td>
</tr>
<tr>
<td></td>
<td>- label</td>
<td>number of valid</td>
<td>number of valid axes</td>
</tr>
</tbody>
</table>

Variables or labels used for arguments must equate to valid values at run-time or an error will result.

**Description:**

Task/AXES defines the axes that are used within a task and how they are used. A maximum of six axes may be assigned using one TASK/AXES statement. A task may contain several TASK/AXES statements when the task requires additional axes or uses axes in ratio mode.

When type 4 (ratio mode) is specified only two axis arguments may be supplied; the second argument is the master axis and the third argument is the slave axis. Axes used in ratio mode must have been previously defined (in any task) by another TASK/AXIS statement. The mathematical ratio between the two axes must be set by an AXIS/RATIO statement.
Tasks that do not use motion control do not require a TASK/AXES statement. However, all drives connected to the SERCOS ring must be registered to the CLC by declaring each drive to its axis in a TASK/AXES statement, even if the axes are not used. This insures that the CLC will correctly respond to drives automatically identified by the SERCOS initialization procedures.

The TASK/AXES statement may be used only within the main Tasks: A, B, C, or D.

This instruction is only active during download and is removed from the instruction stream before normal program cycling.

**Example:**

```plaintext
TaskB_Ax1   equ 3       ; Define Task B axis one
TaskB_Ax2   equ 4       ; Define Task B axis two
TaskB_Ax3   equ 6       ; Define Task B axis three

Task_B:
  task/start  B                                ; Start B
  task/axes   TaskB_Ax1, TaskB_Ax2, TaskB_Ax3  ; Assign axes to task B
```

...
7.3.68. TASK/END (Mark the End of a Task)
Syntax:

```
TASK/END <task>
```

where:

<table>
<thead>
<tr>
<th>argument</th>
<th>valid type(s)</th>
<th>range</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task</td>
<td>ASCII character</td>
<td>A, B, C or D</td>
<td>task ID character</td>
</tr>
</tbody>
</table>

**Description:**
TASK/END indicates the end of a task, but not the end of the program. Event(s) can be outside of the instruction. All CLC instructions must appear between TASK/START and TASK/END statements that specify one of the four tasks.

Active during download, this instruction is removed from the instruction stream before normal program cycling.

**Example:**

```
Task_A:
Task/Start   A
  ;
  ; Task A's program statements
  ;
Task/End A
;
Task_B:
Task/Start   B
  ;
  ;
```
7.3.69. TASK/START (Define the Start of a Task(s))
Syntax:

TASK/START  <task>

where:

<table>
<thead>
<tr>
<th>argument</th>
<th>valid type(s)</th>
<th>range</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task</td>
<td>ASCII character</td>
<td>A, B, C or D</td>
<td>task ID character</td>
</tr>
</tbody>
</table>

Description:
TASK/START indicates the start of a task, but not the end of the program. Event(s) can be outside of the instruction. All CLC instructions must appear between TASK/START and TASK/END statements that specify one of the four tasks. Each task must be identified by a "Task_(task letter):" mark at the beginning of the task’s program.

Active during download, this instruction is removed from the instruction stream before normal program cycling.

Example:

Task_A:
Task/Start  A
  ;
  ;  ;program statements
  ;
Task/End  A
;
Task_B:
Task/Start  B
  ;
  ;
7.3.70. VAR/INIT

VAR/INIT var_start, arg1, arg2, arg3, ......., arg20
var_start: First variable in a block of program variables( Fx, Ix ) to be initialized.
arg1-arg20: initializing value.

example:
VAR/INIT F10, 1, 0, 1, 0, 1, 0, 0, 0, 32000, 0, 1, 0

This instruction is used to initialize Ix and Fx values at compile time, it can be used to initialize the control block of the PID. For type 1 PID arg1 to arg13 have the following definitions:

arg1: Command scaler value.
arg2: Command bais value.
arg3: Feedback scaler value.
arg4: Feedback bais value.
arg5: Kp value.
arg6: Ki value.
arg7: Kd value.
arg8: Ki limit value.
arg9: Minimum output value.
arg10: Maximum output value.
arg11: Preset value.
arg12: Output scaler value.
arg13: Output bais value.
7.3.71. VME/BROADCAST (Broadcast Event to VME)

Syntax:

VME/BROADCAST VME_Units, priority, event

Where:

<table>
<thead>
<tr>
<th>argument</th>
<th>valid type(s)</th>
<th>range</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VME_Units</td>
<td>ASCII hexadecimal</td>
<td>0x0001 to</td>
<td>selects one or more VME units to receive the broadcast event</td>
</tr>
<tr>
<td></td>
<td>- label</td>
<td>0xFFFF</td>
<td>a selection mask is calculated by inclusive &quot;ORing&quot; each unit number shifted</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>left two bit positions</td>
</tr>
<tr>
<td>priority</td>
<td>ASCII character</td>
<td>A, B, C or D</td>
<td>sets priority according to task identifier</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>A = highest, D = lowest</td>
</tr>
<tr>
<td>event</td>
<td>floating point</td>
<td>1 to 63</td>
<td>an event index to one of the first 63 events</td>
</tr>
<tr>
<td></td>
<td>- constant</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- variable Ix, I[x]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- global variable</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Variables or labels used for arguments must equate to valid values at run-time or an error will result.

Description:

VME/BROADCAST is used to send a broadcast event trigger that can be received simultaneously, by one or more CLC cards on a VME bus. The event functions must be enabled on the target CLC cards.

Example:

```
DEFINE I1, EVT1 ; Defines label 'EVT1'

TASK_C: TASK/START C
DATA/SIZE 2, 2, 5, 3, 3, 0 ; Allocates memory for task C

VME/BROADCAST 0, C, EVT1 ; CLC unit 0 is affected
```
### 7.3.72. VME/EVENT (Send Event to VME Unit)

**Syntax:**

```
VME/EVENT   VME_Unit
```

Where:

<table>
<thead>
<tr>
<th>argument</th>
<th>valid type(s)</th>
<th>range</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VME_Unit</td>
<td>integer</td>
<td>0 to 15</td>
<td>specifies the target unit on the VME bus</td>
</tr>
<tr>
<td></td>
<td>- constant</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- variable</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- global variable</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- label</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Variables or labels used for arguments must equate to valid values at run-time or an error will result.

**Description:**

VME/EVENT sends an event trigger to the specified VME unit number using CLC mailbox 4 in the VME short address space. This event must be present and enabled on the target card. The event is bound to the last task enabling the event.

**Example:**

```
TASK_B: TASK/START B
.
.
IF (TEST == SUCCESS)
   VME/EVENT   1 ; CLC unit 1 is affected
.
.
TASK/END B
```
7.3.73. VME/READ (VME Bus Data Read)

Syntax:

VME/READ Axx, address, Dxx, format, byte_order, count, target

Where:

<table>
<thead>
<tr>
<th>argument</th>
<th>valid type(s)</th>
<th>range</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Axx</td>
<td>ASCII characters</td>
<td>A32, A24, or A16</td>
<td>VME source address width</td>
</tr>
<tr>
<td>address</td>
<td>hexadecimal</td>
<td>depends on address width</td>
<td>VME source memory address</td>
</tr>
<tr>
<td>Dxx</td>
<td>ASCII characters</td>
<td>D32, D16, or D8</td>
<td>VME source data width</td>
</tr>
<tr>
<td>format</td>
<td>ASCII characters</td>
<td>F32, I32, I16, I8, U32, U16, U8, or Point</td>
<td>VME source data format</td>
</tr>
<tr>
<td>byte_order</td>
<td>ASCII character</td>
<td>M or I</td>
<td>M = Motorola (big endian)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>I = Intel (little endian)</td>
</tr>
<tr>
<td>count</td>
<td>integer</td>
<td></td>
<td>number of data items to read</td>
</tr>
<tr>
<td></td>
<td>- constant</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- variable Ix, I[x], GIx, GI[x]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- label</td>
<td></td>
<td></td>
</tr>
<tr>
<td>target</td>
<td>- variable</td>
<td></td>
<td>the starting address of the first item in the list of data to read</td>
</tr>
<tr>
<td></td>
<td>- Ix, I[x] or Fx,F[x]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- global variable</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- GIx,GI[x] or GFx,GF[x]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- absolute point table</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- ABS[n]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- relative point table</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- REL[n]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- register Rx, R[x]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- label</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Variables or labels used for arguments must equate to valid values at run-time or an error will result.
Description:
VME/READ copies data, such as one or more of the integer or floating point variables, or absolute or relative point table entries, from the specified VME memory address to the specified target CLC local memory.

The instruction specifies the number and width of the data items transferred. Proper word and double word byte order for 16 and 32 bit data sizes is controlled by the M or I (big or little endian) argument. Therefore, both Motorola and Intel architectures are easily supported. For more information on the limitations of direct VME memory accesses see the CLC/VME Direct Data Access section in the chapter on CLC/VME SYSTEMS.

Format specifiers:

   F32  32-bit floating point
   I32  32-bit integer
   I16  16-bit integer
   I8   8-bit integer
   U32  32-bit unsigned integer
   U16  16-bit unsigned integer
   U8   8-bit unsigned integer

Example:

VME/READ A32,0x402000, D16, U16, M, 10, I01

Ten sixteen bit unsigned data items are read as integers into variables I[01] to I[11]. (Since the CLC uses a 680x0 "M" is specified and byte swapping is not required.)
7.3.74. VME/WRITE (VME Bus Data Write)

Syntax:

VME/WRITE       Axx, address, Dxx, format, byte_order, count, source

Where:

<table>
<thead>
<tr>
<th>argument</th>
<th>valid type(s)</th>
<th>range</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Axx</td>
<td>ASCII characters</td>
<td>A32, A24, or A16</td>
<td>VME source address width</td>
</tr>
<tr>
<td>address</td>
<td>hexadecimal</td>
<td>depends on address</td>
<td>VME source memory address</td>
</tr>
<tr>
<td></td>
<td>Ix, I[x], GIx, GI[x]</td>
<td>width</td>
<td></td>
</tr>
<tr>
<td>Dxx</td>
<td>ASCII characters</td>
<td>D32, D16, or D8</td>
<td>VME source data width</td>
</tr>
<tr>
<td>format</td>
<td>ASCII characters</td>
<td>F32, I32, 116, I8,</td>
<td>VME source data format</td>
</tr>
<tr>
<td></td>
<td></td>
<td>U32, U16, U8, or</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Point</td>
<td></td>
</tr>
<tr>
<td>byte_order</td>
<td>ASCII character</td>
<td>M or I</td>
<td>M = Motorola (big endian)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>I = Intel (little endian)</td>
</tr>
<tr>
<td>count</td>
<td>integer</td>
<td></td>
<td>number of data items to read</td>
</tr>
<tr>
<td></td>
<td>- constant</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- variable Ix, I[x],</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>GIx, GI[x]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- label</td>
<td></td>
<td></td>
</tr>
<tr>
<td>source</td>
<td>- variable</td>
<td></td>
<td>the starting address of the first item in the list of data to write</td>
</tr>
<tr>
<td></td>
<td>Ix, I[x] or Fx, F[x]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- global variable</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>GIx, GI[x] or</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>GFx, GF[x]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- absolute point</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>table ABS[n]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- relative point</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>table REL[n]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- register Rx, R[x]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- label</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Variables or labels used for arguments must equate to valid values at run-time or an error will result.
Description:
VME/WRITE copies data, such as one or more of the integer or floating point variables, or absolute or relative point table entries, from the specified source CLC local memory to the specified VME memory address.

The instruction specifies of the number and width of the data items transferred. Proper word and double word byte order for 16 and 32 bit data sizes is controlled by the M or I (big or little endian) argument. Therefore, both Motorola and Intel architectures are easily supported. Format specifiers:

- F32 32-bit floating point
- I32 32-bit integer
- I16 16-bit integer
- I8 8-bit integer
- U16 16-bit unsigned integer
- U8 8-bit unsigned integer

For more information on the limitations of direct VME memory accesses see the CLC/VME Direct Data Access section in the chapter on CLC/VME SYSTEMS.

Example:

VME/Write  A32, 0x402000, D16, U16, M, 10, I[01]

The contents of the CLC integer variables I[1] through and including I[10] are copied to the specified VME memory space. Since D16 is specified only the lower 16 bits of each 32 bit integer variable is written to VME memory. (Since the CLC uses a 680x0, M is specified and byte swapping is not required.)
CHAPTER 8. DIAGNOSTICS / PROGRAM DEBUGGING AND MONITORING

8.1. System Diagnostics - Codes and Messages

The CLC provides three types of diagnostic messages: Status Messages, Warnings, and
Shutdowns. Diagnostic messages are preceded by an identifying code number. Indramat assigns
these code numbers using the following groups:

- 8.1.1. Status Messages (001-199)
- 8.1.2. Warning Messages (201-399)
- 8.1.3. Shutdown Messages (400 - 599)

A second error code is often included within the primary error message.

"X" indicates a hexadecimal error code
"D" indicates a decimal error code

The Host can request the currently active diagnostic message for the CLC system and for each
user task. In addition, any parameters pertaining to Drive Diagnostics can be accessed through
drive service channel (Dx.x) parameters. Refer to the Drive manual for descriptions of drive
diagnostics.

See the Parameters section for more detailed descriptions of the CLC System and Task
parameters. For example:

Parameter **C-0-0122**: Diagnostic Message
Parameter **C-0-0123**: Diagnostic Code.
Parameter **C-0-0124**: Extended Diagnostics.
Parameter **T-0-0122**: Task Diagnostic Message (where s = A, B, C or D for Task A - D)
Parameter S-x-0095: Drive Diagnostic Message (where x = 1 - 8 for Drive 1 - 8)
8.1.1. Status Messages (001-199)

A Status Message indicates the normal operating status of an axis, task, or the system when there are no errors. A change in status that generates a new status message overwrites the previous message. No user acknowledgment is required for a change in a status message.

001 Initializing System
The CLC is initializing th001 Initializing System executive firmware, the SERCOS ring, and other devices at power-up or exit from parameter mode.

002 Parameter Mode
The CLC is in parameter mode, and the drives are in Phase 2.

003 Initializing Drives
SERCOS has been reconfigured and the ring is being initialized.

004 System is Ready
The system has been initialized and is ready for operation.

005 Manual Mode
All four user program tasks are in manual mode.

006 Automatic Mode: ABCD
The user program tasks indicated at the end of the message are in automatic mode, and the rest are in manual mode. For example, "Automatic Mode: B" indicates that only Task B is in automatic mode.

007 Program Running: ABCD
The user program tasks indicated at the end of the message are running, and the rest are not running or are single-stepping.

008 Single-Stepping: ABCD
The user program tasks indicated at the end of the message are in single-step mode. The other tasks are not running.

009 Select Parameter Mode to Continue
An error during system initialization occurred and was cleared, but the error condition was not corrected. Switch into Parameter Mode to continue.
010 Breakpoint Reached: ABCD
The user tasks indicated at the end of the message have reached a user program breakpoint, and the rest of the tasks are not running.

8.1.2. Warning Messages (201-399)

Warning messages are issued when an improper system condition exists. The condition is important enough to be brought to an operator’s immediate attention, but not critical enough to shut down the system. However, a warning may be a notification of an impending shutdown condition. Warnings typically allow normal system operation to continue.

A warning sets the error bit associated with the affected task or the system and displays the warning message. Once issued, the error condition must be corrected and acknowledged to the system. The user acknowledges and clears a warning with a low-to-high transition of the Clear All Errors bit of the CLC’s System Control Register.

After a warning condition has been corrected and acknowledged, the user program can be resumed at the point where the error occurred. In SERCOS, warnings are Class 2 Diagnostics.

Warning messages can be cleared by correcting the warning condition, or by setting the CLC’s clear error input.

201 Invalid jog type or axis selected
This message is issued before a coordinated I/O jog when an invalid type or axis is selected.

202 Drive D is not ready
This message is issued before a coordinated I/O jog when a drive is not enabled.

203 Power Lost During Program
This function is not currently implemented.

204 SERCOS Ring was disconnected
The SERCOS ring was disconnected before a shutdown error was cleared. The ring is now initialized. To continue, activate the clear input. This message allows detection of an intermittent break in the fiber optic ring.
205 Parameter transfer warning in Task A
There is an error in the parameter transfer instruction. This indicates a warning condition that
does not shut down the task. The parameter format, parameter number, or stored value may be
invalid. A communication error message is displayed in the diagnostic message for the task (A-
D) in which the error occurred (T-0-0122). Information on the actual parameter number that
caused the error is provided in extended diagnostics (C-0-0124).

206 Battery is low: replace it soon
A low voltage on the RAM backup battery has been detected at power-up or initialization from
parameter mode. Replace the battery to prevent any loss of data.

207 Axis D position limit reached
The negative or positive travel limit of axis D was reached, preventing a jog from occurring.

8.1.3. Shutdown Messages (400 - 599)

A Shutdown is issued in an emergency situation or when the system or drives cannot operate
correctly. During a shutdown, the CLC switches the user program tasks into manual mode,
decelerates all motion to zero velocity, and sets the error bit in the status register.

If the shutdown condition results from an E-stop or DDS-2 drive shutdown condition, the CLC
also disables the drives, disabling motor torque and engaging the brake.

A low to high transition on the Clear All Errors bit in the System Control Register will clear a
shutdown. The CLC automatically sends a 'Reset Class 1 Diagnostics' command to each drive
that has an error.

400 Emergency Stop
The Emergency Stop input is active (low). The E-Stop circuit has been opened due to
activation of the E-Stop push button or external logic. All drives on the ring are disabled.
Release the E-Stop button or correct the error condition.

401 SERCOS Controller Error: DD
The SERCOS communications controller has indicated an error on the SERCOS ring. Check
the fiber optic connections, the addresses set on the drives, and the drive configuration.
402 SERCOS Config. Error: see ext. diag. or
402 SERCOS Interface Error: XXXX (versions before 01.20)
An error in the SERCOS service channel has occurred when the CLC was initializing the timing and scaling parameters. The extended diagnostics (C1.124) gives a description of the error.

If the extended diagnostic indicates a timing error or data limit error, check the amount of data or drives on the ring and the minimum cycle time parameter. Otherwise, check the fiber optic connections, the addresses set on the drives, and drive firmware versions.

403 System Error
This error is not issued in current CLC versions and is reserved for future use.

404 Invalid Switch into Phase D
The SERCOS communications controller did not allow a phase switch. Check if power is applied to the drives and if the fiber optic connections and the drive addresses are correct. If drive parameters were just downloaded, switch back into parameter mode to reinitialize the interface. If the above conditions are O.K., the SERCOS interface board may be faulty.

NOTE: This error is issued only in versions that do not use the SERCOS ASIC (firmware versions less than 01.20).

405 Phase D: Drive did not respond
A time-out in the SERCOS ring has occurred when the CLC was initializing timing and scaling parameters. Check the fiber optic connections, the addresses set on the drives, and the drive firmware versions. This distinguishes a communication error from an actual phase switch error.

406 System Error
This error is not issued in current CLC versions and is reserved for future use.
407 Drive D Phase 3 Switch Error
The SERCOS phase 3 switch command failed for the drive indicated. This usually indicates that configuration parameters for the drive are invalid or have not been saved. Check the Drive Status message (parameter Dx.95) for drive ‘D’ for a description of the error.

If the Drive Status indicates that parameters are invalid or lost, display the Phase 2 error parameter list for Drive ‘D’. Switch into parameter mode and change the invalid parameters or download a valid parameter file to the drive.

If the drive is not communicating, check the connections and the addresses. If drive parameters were just downloaded, switch back into parameter mode to reinitialize the interface.

408 SERCOS Controller is in test mode
The Indramat DAS2 SERCOS Controller is in test mode. Set the mode switch on the front of the board to a position where this error does not occur. Note: This error is not issued in versions that use the SERCOS ASIC.

409 SERCOS Disconnect Error
The SERCOS fiber optic ring was disconnected or the drives were powered down while in Phase 3 or 4. A more descriptive message will be displayed in the extended diagnostics (C1.124 - Indicates the first drive in which the drive data failed).

410 System Error
This error is not issued in current CLC versions and is reserved for future use.

411 Drive D Phase 4 Switch Error
The SERCOS phase 4 switch command failed for the drive indicated. This usually indicates that configuration parameters for the drive are invalid or have not been saved. Check the Drive Status message (parameter Dx.95) for drive ‘D’ for a description of the error.

If the Drive Status indicates that parameters are invalid or lost, display the Phase 3 error parameter list for Drive ‘D’. Switch into parameter mode and change the invalid parameters or download a valid parameter file to the drive.

If the Drive Status indicates that there is a feedback error, voltage error, or other hardware error; correct the problem and switch into and out of parameter mode to reinitialize the interface.

412 No drives were found on ring
No drives were found when the CLC initialized the SERCOS ring to Phase 1. Check the addresses set on the drives, in the CLC program, and in the CLC parameters. Also, check that power is applied to the drives and the fiber optic connections are correct.
413 I-O board was not found
The selected I-O board was not found on the VME bus. The correct I-O device must be enabled and the address selected on the device must match the CLC parameter. A VME arbiter must be present in the rack (on CLC/V, switch SW5-8 must be on). See the I-O device descriptions for more information.

414 Parameters were lost
CLC System, Task, and Axis parameters were lost, and defaults have been loaded. The RAM backup battery has failed or was not connected, or an internal system error or new software version has corrupted the memory.

415 Drive D was not found
A drive (D) that is used in a program or selected in the system parameters was not found on the SERCOS ring. Check the fiber optic connections, the address switches on the drives, and the user program and parameters.

416 Invalid Instruction at XXXX
An invalid user program instruction was found by the CLC during compilation. Recompile the program from the PC and download it again. If the error still occurs, check the source program for an instruction that may not be supported in this firmware version.

417 SYSTEM ERROR: pSOS #XXXX
An internal CLC operating system error has occurred. Call Indramat Service for assistance.

418 No program is active
No active user program was found on the CLC during initialization. Download and activate a program from the user interface, then clear the error.

419 Invalid Program File
A checksum or file format error was found in the active program file. Recompile the program from the PC and download it again. If the error still occurs, call Indramat Service for assistance.

420 Drive D Shutdown Error
The drive has issued a shutdown error, which disables motion. Check the SERCOS Drive Status message (parameter Dx.95) for a description of this error. Refer to the drive manual for more information.

421 User Program Stack Overflow
The subroutine call stack for a user program task has overflowed. Check the program for the following conditions:
- there is not a return for every subroutine call
- a subroutine is calling itself
-program flow has caused multiple returns
-more than 256 subroutines are nested.

See the diagnostic message (Error! Reference source not found.) or task error bit for each task to find out which task has this error.

**422 Parameter transfer error in Task A**
There is an error in the parameter transfer instruction. The parameter format, parameter number, or stored value may be invalid. A communication error message is displayed in the diagnostic message for the task (A-D) in which the error occurred (T-0-0122). Information on the actual parameter number that caused the error is provided in extended diagnostics (C-0-0124).

**423 Unimplemented Instruction**
The instruction is not implemented in this version. Recompile the program without this instruction indicated by the current instruction pointer or update the CLC firmware or PC software.

**424 System Error**
This error is not issued in current CLC versions and is reserved for future use.

**425 Instruction Error: see Task A diag.**
An error has occurred in a user program instruction. A more specific message is displayed in the diagnostic message for the task (A-D) in which the error occurred (T-0-0122). This error usually applies to coordinated motion instructions.

**426 Drive D is not ready**
Drives must be enabled before motion commands are issued to them in a user program. Check the Axis Disable bit in Axis D's Control Register, Axis D's status bits, the fiber-optic ring, and the power circuit.

**427 Calc: invalid table index D**
In a user program calculation expression, the index to a point or event table is invalid. See the diagnostic message for each task to find out which task has this error, then check the variable that is used to index the table.

**428 Calc: division by zero**
In a user program calculation instruction, an attempt was made to divide a number by zero. See the diagnostic message for each task to find the task and the instruction, then check the variables used in the expression.
429 Calc: too many operands
In a user program calculation instruction, more than 1000 operands and operators were in the string. See the diagnostic message for each task to find the task and the instruction.

430 Calc instruction: invalid operator
An invalid arithmetic operator was found in a user program calculation instruction. Check the compiler and firmware version numbers, and call Indramat service for assistance.

431 Calc error: see Task A diag.
An error has occurred in a user program calculation instruction. See the task diagnostic message for a communication error message.

432 Calc: too many nested expressions
In a user program calculation instruction, more than 16 operations were pending. See the diagnostic message for each task to find the task and the instruction. Then check the number of operands in the expression, looking for unbalanced parentheses.

433 Setup instruction outside of a task
The following commands must be placed in a task’s main program: TASK/AXES, KINEMATIC, and DATA/SIZE. This error is issued if any of these commands are found in a subroutine. Move the instructions to Task A, B, C, or D, following the TASK/START instruction or Axis Setup icon.

434 Axis D configured more than once
Axis D was selected more than once in a TASK/AXES command (axis setup icon). Modify the program so that the axis is selected once.

435 Axis D not associated with a task
Axis D was not associated with a task using the TASK/AXES command but was used in another command. Modify the program so that the axis is selected.

436 General Compiler Error: XXXX
An error was found in a compile-time instruction (TASK/AXES, KINEMATIC) after program activation. See the task diagnostic message for a description. If there is no task diagnostic message, call Indramat for assistance.

437 Axis D not controlled by this task
Single-axis motion was started from a task not associated with an axis. Motion can only be started from a task with axes selected in the TASK/AXES command.
438 Invalid Axis Selected: D
Axis D was not found on the SERCOS ring or is an invalid axis number. This error is issued during single-axis or ELS motion commands. Check the constant or variable that contains the axis number.

439 Invalid Motion Type: D
The axis type does not match the type of motion used by the instruction. This error is issued when a single-axis command is given to a coordinated motion axis, for example.

440 I-O Transfer Error: see task diag.
An error occurred while reading or writing an I-O register. See the task diagnostic message for a description.

441 DMA error while reading from local RAM

442 DMA error while reading from VME address

443 DMA error while writing to local RAM

444 DMA error while writing to VME address

445 DMA Access Time-out Error

446 DMA Time-out Error

447 VME SYSFAIL Detected

448 VME Communication Handshake Error (D)

449 VME Bus Error
A VME bus error occurred while communicating to another card in pass-through mode through the serial port or during a VME transfer instruction. Check the extended diagnostics for the type of error and the address at which it occurred. If VME transfers were not being performed or if the address does not match that in the program, an internal CLC system error has occurred. Notify Indramat Service of this system error.

450 Event D: invalid event type
The event type selected in the event table is not valid or does not match the type of motion or event. This error is also issued if an event/trigger (event arm) is executed for a motion-based event.

451 Invalid event number D
The event number is not within the bounds selected with the data/size command for this task.
452 More than D event timers armed
Only ‘D’ repeating timer events can be armed at one time. Check the program flow to make sure that triggered events are being disabled.

453 Homing param. transfer error: D
A SERCOS communication error occurred during a drive-controlled homing command. ‘D’ indicates the communication error code returned by the drive. Try to home the axis again. If this error still occurs, call Indramat for assistance.

454 Axis D homing not complete
The drive did not successfully complete the homing sequence. See the drive diagnostics for a status or error message.

455 Invalid VME Data Transfer Class
During a VME/READ or VME/WRITE instruction, the transfer class (e.g. I16, F32, etc.) is invalid.

456 Invalid VME Address
During a VME/READ or VME/WRITE instruction, the VME address does not lie within the valid VME address range.

457 Table Bounds Error During VME Read
The variable or point table index exceeds the size of the table configured in the DATA/SIZE instructions.

458 Table Bounds Error During VME Write
The variable or point table index exceeds the size of the table configured in the DATA/SIZE instructions.

459 Axis D target position out of bounds
The programmed position in an axis/move command exceeds the drive’s travel limits. Adjust the travel limits or check the variable or constant containing the position.

460 Invalid program D from binary inputs
The program selected from the Binary Program Select bits does not exist on the card or is greater than the maximum number of programs.

461 System Error
This error is not issued in current CLC versions and is reserved for future use.

462 System Error
This error is not issued in current CLC versions and is reserved for future use.
463 Ratio command: invalid ratio
In the RATIO command, one of the factors is too large or the master factor is zero.

464 Can't activate while program running
A new program cannot selected through the Binary Program Select inputs unless the program is stopped.

465 Drive D config. error, see ext. diag, or
465 Drive D: telegram type not supported (versions before 01.20)
Drive D does not support a product-specific option or a drive configuration calculation has failed. Product-specific options include ELS, single-axis motion, or I-O cards.

The extended diagnostic message (C1.124, or in Status-System menu) describes the error in more detail. It often shows the parameter that failed along with a short message describing the error. If it indicates that a parameter is invalid or a configuration is not supported, check the axis configuration with the drive hardware or software.

If the extended diagnostic indicates an error such as ‘Handshake time-out’ or ‘Drive is not responding’, the SERCOS ring may have been disconnected during initialization. Check the fiber optic connections and the addresses of the drives on the ring.

466 Drive D: scaling type not supported
Drive D does not support an option such as ELS or single-axis motion, which are product-specific. Check the axis configuration with the drive hardware or software. Note: This error is issued only in versions that do not use the SERCOS ASIC (firmware versions less than 01.20).

467 Invalid ELS Master Option
An option in the ELS/INIT command is invalid, not supported, or inconsistent with the other options.

468 ELS adjustment out of bounds
The phase offset or fine ratio adjustment exceeded the bounds allowed by the drive. The fine adjust must be between -100 and 300%.

469 Axis D accel \( <= 0 \) or \( > \) maximum
The acceleration or deceleration programmed for axis D is negative, zero, or exceeds the maximum acceleration or deceleration parameter (Ax.21 or Ax.22).

470 Axis D velocity \( > \) maximum
The velocity programmed for axis D is exceeds the maximum velocity parameter (Ax.20).
471 Invalid VME Base Address Page: 0xXXXX
The base address page selected in the VME parameter is invalid. See the VME descriptions.

472 VME Event Trigger Rejected
A CLC did not respond to the VME broadcast event message. See the VME event description.

473 VME Event Trigger For Unit D Failed
Unit D did not respond to the VME mailbox event message. See the VME event description.

474 Drive D cyclic data size too large
Too much data is configured in the SERCOS cyclic telegram. The drives currently support up to 16 bytes of configurable data. Remove I-O or registration options from the parameter or program configuration.

475 Axis D capture already configured
An axis has been configured for the feedback capture function in a previous user program command. Only one capture/setup command is allowed for each axis.

476 Axis D: Real Time Bit Setup Error
A SERCOS error occurred while the CLC was configuring the drive’s real time bits for the feedback capture function. Clear the error, enter parameter mode to reinitialize SERCOS, and then exit parameter mode.

477 Axis D: probe edge not configured
This error, issued in the capture/enable instruction, indicates that the selected probe edge for the event has not been configured with the capture/setup instruction.

478 Calc: operand out of range
The operand of a calculation function is out of the range of valid arguments, as when a square root or a logarithmic of a negative number is attempted.

479 Drive D: too many cyclic data elements
The DDS 2.1 currently allows 4 cyclic data elements for the AT and MDT. Remove options such as I-O cards and probing. Refer to the SERCOS Cyclic Telegram Configuration

480 SERCOS Error: MDT is too large
The DDS 2.1 currently allows 104 bytes in the MDT. Remove options such as I-O cards and probing, or reduce the number of drives on the ring. Refer to the SERCOS Cyclic Telegram Configuration
481 Event D is already armed
An event that is currently armed has been armed again using event/trigger (event arm) or the VME event instructions.

482 Checksum Error in Program
The currently active program’s checksum doesn’t match the checksum that is stored in memory. This indicates that a system error has caused the CLC to overwrite memory. Call Indramat service for assistance.

483 Parameter Init. Error: see Task A diag
There is an error in the parameter initialization or bit initialization instruction; which is executed when exiting parameter mode. The parameter format, parameter number, or stored value may be invalid.

A communication error message is displayed in the diagnostic message for the task (A-D) in which the error occurred (T-0-0122). Information on the actual parameter number that caused the error is provided in extended diagnostics (C-0-0124).

In many cases, this error is issued when a drive is not on the ring or the drive parameter is not found for a type of drive.

484 CLC SYSTEM ERROR
This error indicates a problem in the CLC executive firmware. See the extended diagnostics parameter (C-0-0124) for more information, and call the Indramat service department for assistance.

485 SERCOS I-O: too many registers configured
More than 50 SERCOS I-O registers were configured in the CLC, which exceeds the system limit. This includes both drive-resident I-O and SERCOS I-O slaves.

486 SERCOS Device D is not a drive
The SERCOS device with address D was enabled in the user program or parameters as an axis, but an I-O slave or other type of slave was detected.

487 Cam D is invalid or not stored
In the cam/activate command, the selected cam (‘D’) is not stored on the card or does not contain valid data. Check the variable or constant that selects the cam. Check that there is a valid cam with index ‘D’ stored on the CLC.

488 Cam Error: See Task A diag.
An error was issued during a cam command in task (A-D). See the task diagnostic message (T-0-0122) for a description.
489 More than D cam axes selected
The CLC limits the number of axes configured as CLC Cam Axes to ‘D’.

490 System Memory Allocation Error
The dynamic memory space on the CLC has been exhausted. Call Indramat Service for assistance.

491 PC Communication Handshake Error
The CLC/P did not respond to an ASCII message. Check the address configuration on both the PC (config.sys and system.ini) and the CLC/P (address jumper switches).

492 Programs were lost
User programs and data were lost. The RAM backup battery has failed or was not connected, or an internal system error has corrupted the memory. For the CLC/V, the card may have been removed from the VME rack.

493 Data was restored from Flash
User programs and parameters have been restored from Flash EPROM. If the card has just been installed in the VME rack and a valid program is active, clear this error and proceed. If the card has not just been installed, this indicates that the VME standby battery has failed and the previous program and data has been replaced with that stored in Flash.

494 Sequencer init. error: see task T diag
An error has occurred in a sequencer/initialize instruction in task ‘T’. The task diagnostic (T-00122) and the extended diagnostic (C-0-0124) give a more detailed description of the error.

495 Sequencer error: see task T diag.
An error has occurred in a sequencer/execute instruction in task ‘T’. The task diagnostic (T-00122) and the extended diagnostic (C-0-0124) give a more detailed description of the error.

496 Can’t Execute this Instruction from an Event
This user program instruction cannot be executed from within an event. See the task error descriptions and the current program instruction. Some operations, such as sequencer initialization, cannot take place during an event. Move the instruction into a main user task or subroutine

497 Limit switch config. error, see ext. diag
This error is issued at activation of a program when one of the PLS parameters defined in the program is invalid. It is also issued when the ELS setup is incorrect for PLS operation. Parameter C-0-0124 provides a detailed description of the error as an extended diagnostic message.
498 Drive D Shutdown Warning
This error is issued when any drive has a Class 2 shutdown warning. The tasks that stop for errors switch into manual mode and perform a controlled stop of all axes. A drive warning indicates a condition that will later cause a shutdown, but is serious enough to require immediate attention. Since the warning may have already been cleared on the drive, the extended diagnostic (C-0-0124) latches the class 2 diagnostic bits (drive parameter S-0-0012) from the drive so that this condition can be corrected.

Note: Class 2 warnings may not be detected by the CLC if drive parameter S-0-0012 is being continuously read by the user interface or user program, since the diagnostic change bit is reset whenever this parameter is read.

499 Axis number D not supported in this version
This version of CLC software is limited to less than D axes. The axis number is limited to the number of axes allowed. Currently, the standard version of CLC allows 4 axes, and the enhanced version 40 axes.

500 Axis D is not referenced
Axis D has not been homed, the reference position has not been set, or the reference position has been lost. The reference position bit in drive parameter S-0-0403 is zero. To enable or disable this error, use parameter A-0-0006.

501 Drive D communications error
An error in drive communications has occurred while the CLC was reading or writing a service channel parameter for an internal operation. Parameter C1.124, extended diagnostics, has a detailed description of the error.

502 ELS and cams not supported in this version
The ELS and cam features are not supported in this version of the CLC. GPS and GPE are the only firmware versions that include these features.

503 Executing empty block #D
This error is reserved for use by the TRANS01-D control. See the documentation for this version.

504 Communication Timeout
During a timed serial port transmission, the serial port has not responded within the time set in parameter C-0-0016. Timed transmissions used for jogging through Visual Motion. If this error occurs, increase the timeout value in C-0-0016.
505 Axis D is not configured
A user program command was issued to Axis D, but axis D is not configured in the program. Modify the user program so that the correct axis is addressed, or exclude the axis from the system using parameter A-0-0007.

506 I-O Mapper initialization error
The I-O mapper was invalid at initialization, due to loss of memory or an incompatibility in the mapper version.

507 Option Card Power Supply Error
There is an external power supply or output driver error on a DEA-08.1C, DEA-09.1C, or DEA-10.1C expansion cards connected to the CLC-D. This error is issued only in Run Mode (phase 4). All inputs are read as 0, and all outputs are turned off.

Power Supply Error on DEA/C:
The +24V signal voltage on each CLC/D must fall in the following range:

<table>
<thead>
<tr>
<th>External Supply Voltage</th>
<th>Min.</th>
<th>Typical</th>
<th>Max.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>+18V</td>
<td>+24V</td>
<td>+32V</td>
</tr>
</tbody>
</table>

Output Driver Error on DEA/C:
An output driver error turns the ‘ERR’ LED on the DEA/C card on. This indicates that the current drawn by the outputs has caused the output drivers to shut down. There is a protection circuit that prevents damage to the card in this condition. This error is issued if the current is greater than 300mA for more than 1 microsecond.

Troubleshooting
When a 507 error occurs, check parameter C-0-0031 to find the cards that have the error condition. Check the ERR LED on the DEA/C. If it is on, check the current draw of the devices connected to the outputs. If the ERR LED is off, check the +24V external power supply signal to see if it is connected and if it falls in the range above.

8.1.4. Fatal System Errors

When a microprocessor exception or an unrecoverable system error occurs, the CLC may stop communicating with Visual Motion and teach pendant interfaces. If possible, control is passed to a CLC-resident monitor routine that can provide debugging information to an ASCII terminal connected to the Host serial port. If a fatal error repeatedly occurs and cannot be recovered, call Indramat Service for assistance in debugging.
8.2. Finding Program Problems

Locating and eliminating program errors can be a frustrating effort. Therefore, design a program before you start writing it (or making modifications). Attempting to incrementally fix an ill conceived structure usually results in a complex collection of logic that eventually even the author doesn’t understand.

Start with simple, basic program blocks. Test the blocks, independently if possible, even if testing requires writing a bit more program just for test purposes. A tested and dependable section of a program allows you to focus on just the potential problem areas. If the program compiles correctly, make sure that the problem lies with the program, not the hardware. If necessary, write short test programs to test individual hardware functions.

Use a program branch and the CLC’s message capabilities to insert a message into your program. Momentarily stopping the program and checking critical values can tell you where things are going wrong.

Think through the implications of using triggered events. Remember that events and the execution of event functions typically occurs asynchronously to program tasks. You cannot always depend on the timing of triggered events. It may be necessary to add additional program code to provide synchronization.

The following Task parameters can also be used to help with program debugging:

- T-0-0130 Current Instruction Pointer
- T-0-0131 Current Instruction
- T-0-0132 Instruction Pointer at Error
- T-0-0133 Composite Instruction Pointer
- T-0-0135 Current Subroutine
- T-0-0136 Stack Variable Data
- T-0-0137 Task Subroutine Breakpoint
- T-0-0138 Sequencer Information
- T-0-0200 Last Active Event Number

See Chapter 3. Parameters for more information.

See Appendix B - Communication Error Codes and Messages
If the program does not compile, or compiles with errors, use VisualMotion's "Display Code" selection from the File menu to check that the compiler is generating the instructions you intend. Remember that the compiler doesn't check your program's logic, the compiler can only check for proper syntax and use. VisualMotion's compilers typically provide error or warning dialog boxes that refer to line numbers in the displayed code. The following section provides the syntax of the code displayed.

8.2.1. Test Code

A typical example of additional code for testing program functionality is the use of counters. One way to implement a counter would be to change the state of an I/O bit after a distance event has occurred. After each move, which is supposed to trigger the event, increment an integer variable called “move_count”. Then use a branch statement to test whether the I/O bit did, in fact, change state. If it did, then increment an integer variable called “event_count”. The final value of “event_count” can be compared to “move_count” to see if, in fact, the event occurred once for every move.

8.3. CLC Compiler Base Code

Compiling an Icon or Text Language program produces a text file output listing in CLC Base Code, using mnemonics and syntax similar to assembly language. The Base Code resulting from the compilation of a program may be viewed using Window's Notepad by selecting "Display Code" from the VisualMotion File menu. Base Code may also be viewed using a compatible ASCII-only text editor.

Base Code is typically used as an aid to debugging when checking a program for logical errors. Base Code files are view-only program listing files. Editing a Base Code file has no effect on a subsequent re-compilation of the program.

The labels in a Base Code listing result from both user defined labels and the labels that are generated internally by the Visual Motion compiler.

8.3.1. Base Code instruction mnemonics and valid arguments

The following lists the Base Code instruction mnemonics and valid arguments. Instructions requiring more than one argument show the arguments separated by commas ",". Alternative forms for arguments are shown by enclosing a general form for each argument in square brackets, separated by a vertical bar "|".

**ABORT_PATH**

`ABORT_PATH [task]`

Halts coordinated motion in the specified task.
ACCEL
ACCEL  [axis | label | Ix | GIx],[rate | label | Fx | Gfx]

Sets the acceleration rate for the specified axis.

AXES
AXES   task mode, axis, axis, axis

Specifies the axes to be assigned to this task and how they will be used. (All axes used in a task must be declared for that task.)

task mode =
1 for single axis non-coordinated motion
2 for coordinated axis for multi-axis coordinated motion
3 for velocity mode, rotation only - no axis positioning
4 for ratioed slave axis
5 for ELS mode
6 for Torque mode
axis = a valid identifier for an axis, from 1 to the maximum number of axes

AXES_GROUP
AXES_GROUP   task mode, axis, axis, axis, axis, axis

AXES_GROUP
AXES_GROUP is similar to the AXES command; however it permits up to six axes.

AXIS_EVENT
AXIS_EVENT   [axis | label | Ix | GIx], event #1, event #2, event #3, event #4

Enables up to four Repeating Position events for a single-axis, ELS, ratio, or velocity mode axis.

Event # = [an integer Event # | label | Ix | GIx]
**AXIS_WAIT**

*AXIS_WAIT*  
[axis | label | Ix | Gix]

If the argument is a positive integer representing a valid axis, program execution waits for the axis to be within its preset DDS position window. If the argument is -1, program execution waits until all axes in the task are within their position windows. The position window is defined by the DDS parameters: Position Window and Zero Velocity. **AXIS_WAIT** will wait indefinitely if used with velocity mode (axis task mode 3) since positioning is not used.

**BNE**

*BNE*  
label (subroutine or event)

Branches to label if the task's status word is set to "not equal".

**BEQ**

*BEQ*  
label (subroutine or event)

Branches to label if the task's status word is set to "equal".

**BGT**

*BGT*  
label (subroutine or event)

Branches to label if the task's status word is set to "greater than".

**BLT**

*BLT*  
label (subroutine or event)

Branches to label if the task's status word is set to "less than".

**BGE**

*BGE*  
label (subroutine or event)

Branches to label if the task's status word is set to "greater than or equal".

**BLE**

*BLE*  
label (subroutine or event)

Branches to label if the task's status word is set to "less than or equal".
BRA
BRA label (subroutine or event)

Branch to label, always (no matter what)

CALL_FUNC
CALL_FUNC func_offset, ret_pointer, arg_count, arg1_ptr, ... argn_ptr

Calls the function at func_offset with a return pointer and a variable number of arguments.

<table>
<thead>
<tr>
<th>func_offset</th>
<th>offset in bytes from current program counter to start of function</th>
</tr>
</thead>
<tbody>
<tr>
<td>ret_pointer</td>
<td>pointer to int or float return variable if (0), there is no return value</td>
</tr>
<tr>
<td>arg_count</td>
<td>number of arguments passed to the function if (0), there are no arguments there can be between 1 and 5 arguments</td>
</tr>
<tr>
<td>arg1_ptr...</td>
<td>pointer to argument passed to function can be int, float, global int, global float, constant int, constant float, local int, local float, absolute or relative point label used as initial value of local variable</td>
</tr>
<tr>
<td>argn_ptr</td>
<td></td>
</tr>
</tbody>
</table>

CALC
CALC evaluates the equation

CAP_ENABLE
CAP_ENABLE axis, probe, event#

Enables the event on the axis for the probe transition. When the transition occurs, the event triggers.

axis = from 1 to the maximum number of axes
probe:
1 = probe 1, 0 --> 1
2 = probe 1, 1 --> 0
3 = probe 2, 0 --> 1
4 = probe 2, 1 --> 0
event = [event # | label | Ix | GIx]
CAP_SETUP

CAP_SETUP axis, probe

At program activation the drive is configured to capture feedback position on it's probe transition and to include position data in its cyclic telegram data.

axis = from 1 to the maximum number of axes

probe:

1 = probe 1, 0 -> 1
2 = probe 1, 1 -> 0
3 = probe 2, 0 -> 1
4 = probe 2, 1 -> 0

CLEAR

CLEAR [Ix | Glx | Fx | GFx | label]

Sets integer or float variable to zero

COMP

COMP [Ix | Glx | Fx | GFx | label], [Ix | Glx | Fx | GFx | label]

Set the task's status word to the logical result of 1st argument minus 2nd argument

DATA_SIZE

DATA_SIZE I, F, ABS, REL, EVT, ZONE

Sets the amount of memory allocated for each type of data in one of the four program tasks.
(The total program requirement is the sum of the DATA_SIZE allocations for each task in the program.)

I = the number of integer variables allocated for this task
F = the number of floating point variables allocated for this task
ABS = the number of absolute point table entries allocated for this task
REL = the number of relative point table entries allocated for this task
EVT = the number of event table entries allocated for this task
ZONE = the number of zone table entries allocated for this task

DEC

DEC [Ix | Glx | label]

Subtracts 1 from the specified integer variable
DECEL

DECEL \( axis, rate \)

Sets the deceleration rate for the axis
\( \text{axis} = \) [integer constant | label | Ix | GIx]
\( \text{rate} = \) [floating point constant | label | Fx | GFx]

ELS_ADJUST

ELS_ADJUST \( axis, offset \)

Sets the phase or velocity offset for the ELS axis.
\( \text{axis} = \) [integer constant | label | Ix | GIx]
\( \text{offset} = \) [floating point constant | label | Fx | GFx]

ELS_INIT

ELS_INIT \( \text{els type}, \text{slave axis}, \text{master axis}, \text{encoder}, \text{sync type} \)

Initializes the relationship between master and slave axes.
\( \text{els type}: \)
\( 1 = \text{virtual master} \)
\( 2 = \text{real master (daisy-chained)} \)
\( 3 = \text{real master (SERCOS)} \)
\( 4 = \text{follow axis feedback} \)
\( \text{slave axis} = \) [integer constant | label]
\( \text{master axis} = \) [integer constant | label]
\( \text{encoder}: \)
\( 1 = \text{primary encoder} \)
\( 2 = \text{secondary encoder} \)
\( \text{sync type}: \)
\( 1 = \text{velocity} \)
\( 2 = \text{phase} \)

ELS_MODE

ELS_MODE \( axis, mode \)

Sets the mode for the specified ELS axis.
\( \text{axis} = \) [integer constant | label | Ix | GIx]
\( \text{mode}: \)
\( 1 = \text{single axis} \)
\( 2 = \text{ELS synchronization} \)
END

Defines the end of the program for this task.

EVENT_DONE

EVENT_DONE event

Marks the specified event status as complete.

\[ \text{event} = \{\text{integer constant} \mid \text{Ix} \mid \text{Glx} \mid \text{label}\} \]

EVENT_ENABLE

EVENT_ENABLE event

Activates the specified repeating timer event.

\[ \text{event} = \{\text{integer constant} \mid \text{Ix} \mid \text{Glx} \mid \text{label}\} \]

EVENT_END

Defines the end of an event routine program code.

EVENT_START

Marks the beginning of an event routine program code.

EVENT_WAIT

EVENT_WAIT event

Pauses task execution until the specified active event completes.

\[ \text{event} = \{\text{integer constant} \mid \text{Ix} \mid \text{Glx} \mid \text{label}\} \]

FUNC_ARG

\[ \text{func\_label: FUNC\_ARG label, type, <min value>, <max value>} \]

Declares local variables.

<table>
<thead>
<tr>
<th>func_label</th>
<th>text label of function</th>
</tr>
</thead>
<tbody>
<tr>
<td>label</td>
<td>text string identifier of local variable</td>
</tr>
<tr>
<td>type</td>
<td>‘F’ = float, ‘I’ = integer, “ABS” = ABS point index, “REL” = REL point index</td>
</tr>
<tr>
<td>min value</td>
<td>optional minimum value of argument</td>
</tr>
<tr>
<td>max value</td>
<td>optional maximum value of argument</td>
</tr>
</tbody>
</table>
**FUNC_END**

```
func_label:  FUNC_END  return value
```

Indicates the end of a function and optional return value.

<table>
<thead>
<tr>
<th>func_label</th>
<th>text label of function</th>
</tr>
</thead>
<tbody>
<tr>
<td>return value</td>
<td>return argument</td>
</tr>
</tbody>
</table>

**FUNC_START**

```
func_label:  FUNC_START
```

Indicates the start of the function named by ‘func_label’.

<table>
<thead>
<tr>
<th>func_label</th>
<th>text label of function</th>
</tr>
</thead>
</table>

**GET_PARAM**

```
GET_PARAM  type, set, ID number, destination
```

Copies the specified parameter data to the specified integer or floating point variable (the variable type must match the parameter type).

type:
- A = axis
- C = system
- D = drive
- T = task

set = axis or drive ([integer constant | Ix | label]), or task ID letter
ID number = identifying parameter number (range 1 to 65535)
destination = destination variable, [Ix | GIx | Fx | GFx | label]

**GO**

```
GO  axis
```

Starts continuous motion on the axis. The axis must be configured as non-coordinated or velocity mode.

axis = [integer constant | Ix | GIx | label]
HOME

HOME axis

Enables motion homing the specified axis. (The homing parameters must have been set in the DDS drive.)

axis = [integer constant | Ix | GIx | label]

INC

INC [Ix | label]

Adds 1 to the specified integer variable.

LOCAL/VARIABLE

func_label: LOCAL/VARIABLE label, type

Declares local variables.

<table>
<thead>
<tr>
<th>func_label</th>
<th>text label of function</th>
</tr>
</thead>
<tbody>
<tr>
<td>label</td>
<td>text string identifier of local variable</td>
</tr>
<tr>
<td>type</td>
<td>‘F’ = float, ‘I’ = integer</td>
</tr>
</tbody>
</table>

MESSAGE_PORT

func_label MESSAGE_PORT target, string, <pointer>

Outputs formatted string to designated port.

<table>
<thead>
<tr>
<th>func_label</th>
<th>text label of function</th>
</tr>
</thead>
<tbody>
<tr>
<td>target</td>
<td>1 = diagnostic message.</td>
</tr>
<tr>
<td></td>
<td>2 = status message.</td>
</tr>
<tr>
<td></td>
<td>3 = serial host port(Port A).</td>
</tr>
<tr>
<td></td>
<td>4 = serial teach pendant port(Port B).</td>
</tr>
<tr>
<td>string</td>
<td>formatted text string to display, formatting types are %d, %f, %x</td>
</tr>
<tr>
<td>pointer</td>
<td>optional single argument - Rx, Fx, Ix, GFx, or GIx</td>
</tr>
</tbody>
</table>

MOVE_JOINT

MOVE_JOINT ABS point

Moves the joint based on an absolute point (Six-axis CLC only.)

ABS point = [integer constant | Ix | GIx | label], an entry in the absolute point table
KINEMATIC

KINEMATIC

kinematic library number

Selects the set of equations specified by the library number from an optional kinematic library. Used to translate Cartesian coordinates for custom coordinated motion applications such as robots.

MOVEA_AXIS

MOVEA_AXIS

axis, distance, event, event, event, event

Starts single axis absolute motion for the specified axis, and activates the specified events.

axis = [integer constant | Ix | GIx | label]
distance = [floating point constant | Fx | GFx | label]
event = [integer constant | Ix | GIx | label]

MOVER_AXIS

MOVER_AXIS

axis, distance, event, event, event, event

Starts single axis relative motion for the specified axis, and activates the specified events. An event is specified by an integer number index into the event table or a label for an integer variable containing the index.

axis = [integer constant | Ix | GIx | label]
distance = [floating point constant | Fx | GFx | label]
event = [integer constant | Ix | GIx | label]

MOVEA_PATH

MOVEA_PATH

ABS point

Starts coordinated motion from the current position to the point specified in the absolute point table.

ABS point = [integer constant | Ix | GIx | label]

MOVER_PATH

MOVER_PATH

REL point, ABS point

Starts coordinated straight line motion from the current position to the point specified by the vector sum of the absolute and relative points.

ABS point = [integer constant | Ix | GIx | label]
REL point = [integer | Ix | GIx | label]
MOVEA_CIRCLE
MOVEA_CIRCLE ABS point, ABS point

Starts coordinated motion from the current position, through the first specified point, ending at the second specified point.
ABS point = [integer constant | Ix | GIx | label]

MOVER_CIRCLE
MOVER_CIRCLE REL point, REL point, ABS point

Starts coordinated motion from the current position, through the point specified by the vector sum of the ABS point and the first REL point, ending at the point specified by the vector sum of the ABS point and the second REL point.
ABS point = [integer constant | Ix | label]
REL point = [integer constant | Ix | label]

MSG_DIAG
MSG_DIAG ASCII text string

Sets the current diagnostic message to the specified ASCII text string.

MSG_STATUS
MSG_STATUS ASCII text string
I
Sets the current status message to the specified ASCII text string.

PARAM_BIT
PARAM_BIT type, set, ID number, source, I/O mask

Sets the parameter bit specified by the type, set, ID number and I/O mask to the value in the specified source variable at initialization.
type:
A = axis
C = system
D = drive
T = [A | B | C | D] (task ID letter)
set = [integer constant | Ix | GIx | label] for axis or drive; or [A | B | C | D] for task
ID number = [integer constant] for a parameter number in the range 1 to 65535
source = [integer constant | floating point constant | Ix | GIx | Fx | GFx | label]
I/O mask = specifies 1 to 16 bits in an I/O register
PARAM_INIT

PARAM_INIT type, set, ID number, source

Sets the specified parameter to the value in the specified variable at initialization.

- **type:**
  - **A** = axis
  - **C** = system
  - **D** = drive
  - **T** = [A | B | C | D] (task ID letter)

- **set** = [integer constant | Ix | GIx | label] for axis or drive; or [A | B | C | D] for task ID number = [integer constant] for a parameter number in the range 1 to 65535

- **source** = [integer constant | floating point constant | Ix | GIx | Fx | GFx | label]

PID_CONFIG

PID_CONFIG #, type, control_register, status_register, loop_time, set_point_type, set_point, set_point_axis, feedback_type, feedback, feedback_axis, output_type, output, output_axis, control_block

- **#:** PID loop number, range 1-10.
- **type:** PID loop type, currently only 1 is valid.
- **control_register:** label or number of register used for control of this loop.
- **status_register:** label or number of register used for status of this loop
- **loop_time:** update time of this loop, multiples of 8 millisecond.
- **set_point_type:** Type of set point, 1=variable, 3=unsigned register, 4=signed register
- **set_point:** Axis parameter, register, variable, or equivalent label to be used as the set point of this loop.
- **set_point_axis:** For axis parameters, axis number; else 0.
- **feedback_type:** Type of feedback, 1=variable, 2=axis parameter, 3=unsigned register, 4=signed register
- **feedback:** Axis parameter, register, variable, or equivalent label to be used as the feedback of this loop.
- **feedback_axis:** For axis parameters, axis number; else 0.
- **output_type:** Type of output, 1=variable, 2=axis parameter, 3=unsigned register, 4=signed register
- **output:** Axis parameter, register, variable, or equivalent label to be used as the output of this loop.
- **output_axis:** For axis parameters, axis number; else 0.
- **control_block:** First variable in a block of 20 float variables(Fx) to be used for this loop.

See also VAR_INIT.
POSITION

*POSITION task, ABS point*

Copies the current position coordinates of the specified task to the specified ABS point table entry. The contents of the point table entry are overwritten.

task = [A | B | C | D]
ABS point = [integer constant | Ix | GIx | label]

RATIO

*RATIO master axis, slave axis, master ratio, slave ratio*

Sets the ratio between the specified master and slave axes.

master axis = [integer constant | Ix | GIx]
slave axis = [integer constant | Ix | GIx]
master ratio = [floating point constant | Fx | GFx]
slave ratio = [floating point constant | Fx | GFx]

READ

*READ register, count, target variable*

Copies the contents of the specified I/O register(s) to the lower 16 bits of the specified integer variable(s). The upper word of the variable(s) are zero-filled. Only a contiguous block of registers can be moved.

register = an integer constant specifying the number of the starting source I/O register
count = a positive integer constant for the number of register to copy
target variable = the starting integer variable table entry for the destination of the data

RESUME_PATH

*RESUME_PATH task*

Restarts previously halted coordinated motion in the specified task.

task = [A | B | C | D]

RETURN

Marks the end of a subroutine's program code, and returns program execution to the calling program.
SET

SET  I/O state, register, I/O mask

Sets the specified register's bits, that are enabled by the I/O mask, to the state specified by I/O state.
I/O state = 16 bit binary word of bits to set in the specified register. 0 = off, 1 = on.
register = an integer number specifying an I/O register
I/O mask = 16 bit binary word specifying the bits that may be changed. 1 = enabled

SET_PARAM

SET_PARAM  type, set, ID number, source

Copies the specified parameter's value to the specified integer or floating point variable. The
source variable data type must match the destination parameter data type.
type:
  A =  axis
  C =  system
  D =  drive
  T =  task
set = [Ix | GIx | label] for axis or drive; or [A | B | C | D] for task ID letter
ID number = identifying parameter number, within the range: 1 to 65535
source = [integer constant | floating point constant | Ix | GIx | Fx | GFx | label]

START

Marks the beginning of a task or subroutine.

STOP

STOP  axis

Signals the DDS to halt single-axis or velocity mode motion on the specified axis if the
argument is a positive integer (1 - 8). If the argument is -1, motion is halted for all single-axis
and velocity mode axes in the task. Signaling the DDS to halt motion decelerates the axis to
zero velocity using the deceleration rate programmed in the appropriate DDS drive parameter.
axis = [integer constant | Ix | GIx | label]

STOP_PATH

STOP_PATH  task

Stops coordinated motion in the specified task.
task = [A | B | C | D]
TEST
TEST  register, I/O mask

Sets the task’s status word to the result of a logical AND of the specified register and the I/O mask.
register = a positive integer constant for a modifiable CLC register
I/O mask = 16 bit binary word

VAR_INIT
VAR_INIT ar_start, arg1, arg2, arg3, ........, arg20

var_start: First variable in a block of program variables( Fx, Ix ) to be initialized.
arg1- arg20: values to initialize with.

VEL
VEL  axis, rate

Sets the velocity specified by rate in the specified task axis.
axis = [integer constant | Ix | GIx | label]
rates = [floating point constant | Fx | GFx | label]

VME_EVENT
VME_EVENT  card, priority, event

Sends a broadcast event across the VME backplane to cards selected with the specified priority
and event number.

card = CLC unit as a power of 2 (i.e. unit 1 = 2, unit 3 = 8), or 0xFFFF for all cards
priority = [A | B | C | D] (task ID letter)
event = [integer constant | Ix | GIx | label], the lesser of 64 or the maximum number of events

VME_READ
VME_READ  [Ax],[address],[Dxx],[format],[order],[count],[target]

Reads one or more VME memory locations to the specified target register(s).

Axx = [A32 | A24 | A16], the VME source memory address width
address = the VME memory address of the source data, in hexadecimal
(Depends on the source address width.)

Dxx = [D32 | D16 | D8], the VME memory source data width
format = [F32 | I32 | I16 | I8 | U16 | U8], the format of the VME source data
order = [M | I] (for Motorola or Intel), the VME source data byte order

count = [integer constant | Ix | GIx | label], the number of locations to read
target = [Ix | GIx | Fx | GFx | label], the beginning CLC table variable to receive the data
VME_WRITE

VME_WRITE [Axx],[address],[Dxx],[format],[order],[count],[source]

Writes to one or more VME memory locations from a specified source register(s).
Axx = [A32 | A24 | A16], the VME target memory address width
address = the VME memory address of the target data, in hexadecimal
          (Depends on the source address width.)
Dxx = [D32 | D16 | D8], the VME memory target data width
format = [F32 | I32 | I16 | I8 | U16 | U8], the format of the VME source data
order = [M | I] (Motorola or Intel), the VME target data byte order
count = [integer constant |Ix | GIx | label], the number of locations to read
source = [Ix | GIx | Fx | GFx | label], the beginning CLC table variable to read to VME memory

WAIT

WAIT time

Suspends task execution for the number of milliseconds specified by time.
time = [integer constant |Ix | GIx | label]

WAIT_IO

WAIT_IO register, I/O mask, I/O state

Suspends task execution until the specified I/O conditions are met.
register = an integer constant for a CLC register
I/O mask = identifies 1 to 16 bits in an I/O register
I/O state = 0 --> off; non-zero --> on
WAIT_PATH

WAIT_PATH task, ABS or REL point, condition

Suspends task execution until the specified path planner conditions are met.

- task = [A | B | C | D]
- ABS point = [integer constant | IX | GIx | label], reference to an absolute table entry
- REL point = [integer constant | IX | GIx | label], reference to a relative table entry

- condition:
  0 = Ready
  1 = Accel
  2 = Slew
  3 = Blending
  4 = Target decel
  5 = Controlled stop
  6 = Stopped
  7 = At target
  8 = Done

WRITE

WRITE register, count, source

Copies the data in the specified integer variable(s) to the specified I/O register.

- register = an integer number for the starting destination I/O register
- count = a positive integer constant for the number of registers to copy
- source = an integer number for the starting source integer variable table entry
8.4. Icon Language Warnings and Error Messages

The following warning messages are generated by the VisualMotion Icon Compiler. After receiving a warning message you may continue or exit the compilation.

- Data missing in one or more fields, do you still wish to continue?
- Caution! Changing Modes may halt motion. Continue?
- Caution! Changing Modes may start motion. Continue?

The following error messages are displayed by VisualMotion’s Icon Compiler.

- Valid Entries are '0' or '1'.
- Invalid name!
- Cannot change task or open dialog box while dialog box is open
- Axis undefined or not unique.
- Valid event numbers are 1 to 100.
- Valid axis numbers are 1 to 8.
- Valid number range is 1 - 32767.
- Valid percents are 1 - 100.
- Labels must start with an alpha character!
- Label name already exists!
- Number missing or out of range.
- Selected Icon is not a subroutine or no icon selected!
- Data Field Empty!
- Label type must be defined!
- Task name undefined.
No filename specified.

Non-Branch icons have only two output connections.

Branch icons have only two output connections.

Point out of range.

Connection could not be made, try connecting adjacent ---?

Only connections between icons or adjacent blocks can be ---?

Finish icon not found or open path!

Start icon not found or multiple Start icons found!

Icon program not found!

Cannot open code file!

Unknown icon term _______.

Missing axis selection.

Open in program flow, at or near highlighted icon, ---?

Branch Icon has missing connection or one in wrong dire ---?

No axis selected!

Time Delay out of range!

Could not initialize update timer!

Operation type not selected!

Drive numbers doesn't match.

Should drive number be c ---?

Can't open file _______.

Source or target not selected!
Valid range ___ - ___

Valid range ___ - ___

CLC card parameters cannot be changed!

File syntax other than parameters!

File of different type parameters!

CLC card is not communicating!

No selection made!
8.5. Text Language Error Messages

The following are error messages produced by the Visual Motion text compiler. Line numbers refer to code displayed by selecting "Display Code" from the Visual Motion File menu. For further information on the format of the code displayed see 8.3. CLC Compiler Base Code.

8.5.1. First Pass Errors

  CLC code converter error log file!
  Unable to open source file!
  Line [nnn], Maximum number of terms reached!
  Line [nnn], unknown mnemonic operator - [xxx]
  Line [nnn], unknown, missing or wrong argument - [xxx]
  Line [nnn], missing point argument!
  Line [nnn], missing closing bracket "]")!
  Line [nnn], additional arguments - [xxx...]
  Line [nnn], point number '??' out of range (1-nn)
  Line [nnn], missing arguments!
  Line [nnn], unknown IF conditional terms - [?] [?]!
  Line [nnn], ELSE or ENDIF without IF term!
  Line [nnn], maximum number of nested IFs exceeded!
  Line [nnn], sequencing error, IF, ELSE, or ENDIF imbalanced
  Line [nnn], missing message text!
  Line [nnn], incompatible circle arguments - [xxx]
  Line [nnn], variable out of range - [variable name]
  Line [nnn], right side of EQU must be a number - [____]!
  Line [nnn], label [label name] not found
  Line [nnn], arguments must be integer or constant!
  Line [nnn], bit number [nn] out of range (1-nn)
  Line [nnn], register number [nn] out of range (1-nn)
  Line [nnn], integer variable number [nn] out of range (1-nn)
  Line [nnn], register number + count exceed range (1-nn)
  Line [nnn], axis number [nn] out of range (1-n)
  Line [nnn], mode number [nn] out of range (0-n)
  Line [nnn], mark "_____" also defined on line [nn]
  Mark [_____] on line [nnn] was not referenced in program!
  Mark [_____] used on line [nnn] is not declared!
  Line [nnn], event number [nn] out of range (1-nn)
  Line [nnn], delay value [n...] out of range (1-n...)
  Line [nnn], too many arguments!
8.5.2. Second Pass Compiler Errors

'xxxx' - unresolved mark reference.
Line xx, all probe types zero or not unique!
Line xx, argument 'yyyy' out of range!
Line xx, axes missing or not unique!
Line xx, axis number 'yyyy' out of range(www,yyyy,1-zzzz).
Line xx, bit number 'yyyy' out of range(1-16)!
Line xx, 'compare' arguments must be floats, integers, or constants!
Line xx, event element 'yyyy' missing or entered with spaces!
Line xx, event element 'yyyy' unknown!
Line xx, event EVT[].yy data is not changeable in program!
Line xx, event function 'yyyy' not found in program!
Line xx, event message 'yyyy' must start with quotes!
Line xx, event number 'yyyy' out of range!
Line xx, float number 'yyyy' conversion error!
Line xx, hex number 'yyyy' conversion error!
Line xx, integer number 'yyyy' conversion error!
Line xx, invalid argument 'yyyy'!
Line xx, invalid cam number 'yyyy'! Range 1 to 8.
Line xx, invalid count or count plus register exceeds range!
Line xx, invalid Encoder type 'yyyy', 1=primary, 2=secondary!
Line xx, invalid ELS type 'yyyy', range 1 to 4!
Line xx, invalid sync type 'yyyy', 1=velocity, 2=phase, 3=cam!
Line xx, invalid VME Address 'yyyy'!
Line xx, invalid VME address width 'yyyy'!
Line xx, invalid VME byte order 'yyyy'!
Line xx, invalid VME count 'yyyy'!
Line xx, invalid VME data width 'yyyy'!
Line xx, invalid VME data format 'yyyy'!
Line xx, left term 'yyyy' of equation must not be constant!
Line xx, maximum number of terms reached.
Line xx, maximum size(20) of term exceeded!
Line xx, message exceeds 80 characters!
Line xx, missing argument(s)!
Line xx, missing beginning quotes of message!
Line xx, missing closing bracket ']'!
Line xx, missing closing curly brace '}'!
Line xx, missing closing quotes of message!
Line xx, missing mark name!
Line xx, Parameter <type> must be 'A', 'C', 'D' or 'T'
Line xx, point element 'yyyy' missing or entered with spaces!
Line xx, point element 'yyyy' unknown!
Line xx, register number 'yyyy' out of range(1-zzzz)!
Line xx, table or array index out of range 'yyyy'!
Line xx, table or array label index out of range 'yyyy'!
Line xx, Task must be 'A', 'B', 'C' or 'D'!
Line xx, too many arguments!
Line xx, unknown mnemonic operator - 'yyyy'!
Line xx, unknown or out of range variable 'yyyy'!
Line xx, unresolved index 'yyyy'!
Line xx, unresolved index label 'yyyy'!
Line xx, unsupported data transfer! VME bus width 'yyyy', VME format 'zzzz', local
Line xx, unsupported structure transfer!
Line xx, valid modes are 1=single axis, 2=ELS synchronized!
Line xx, variable table 'yyyy' index unknown!
Mark table filled - yyyy, reduce number of subroutine calls.
No main task( A, B, C, or D ) found!
Sequencing error in output file!
Size of program exceeds compiler space!
Unable to allocate memory for compiler!
Unable to open source file.
Write to file error!

'xxxx' - unresolved mark reference.
The mark 'xxxx' was used as a destination in a branch or subroutine call, but was not found in
the code. Check for possible spelling error or missing subroutine.

**Line xx, all probe types zero or not unique!**
The probe arguments are both zero or are the same.

**Line xx, argument 'yyyy' out of range!**
The argument 'yyyy' is out of range, check syntax in manual.

**Line xx, axes missing or not unique!**
In a AXES_GROUP command for ratioed axis, the slave axis argument is zero or is the same
as the master axis.

**Line xx, axis number 'yyyy' out of range(www,xxx,1-zzzz).**
The axis number or label 'yyyy' has not been resolved to a valid number. The numbers
'www', 'xxx', and range 1 to 'zzzz' are valid axis numbers.

**Line xx, bit number 'yyyy' out of range(1-16)!**
On line 'xx', the string 'yyyy' is evaluated to number outside of the valid range for register
bits.
Line xx, 'compare' arguments must be floats, integers, or constants!
Compare arguments must be Fx, GFx, GIx, Ix or equivalent labels or constants. Compares are derived from "IF" statements in textual language programs or "BRANCH" icons in GUI programs.

Line xx, event element 'yyyy' missing or entered with spaces!
On line 'xx', the compiler has not found a ""]" in the event string 'yyyy'. It uses this to position to the start of the event element. The event element { s, t, d, a, f, m }must follow immediately.

Line xx, event element 'yyyy' unknown!
The event element 'yyyy' was not found in the event element table, check manual for exact syntax.

Line xx, event EVT[.yy data is not changeable in program

Line xx, event function 'yyyy' not found in program!
The event function 'yyyy' was not found in the program. Check spelling and capitalization.

Line xx, event message 'yyyy' must start with quotes!
The compiler is expecting a quote to start the ASCII string for the event message, but did not find it.

Line xx, event number 'yyyy' out of range!
On line 'xx', the string 'yyyy' was evaluated to be out of the range for events defined for this program. Events and other variables are declared in the "DATA/SIZE" command in a textual language program or by the "SIZE" icon in GUI programs.

Line xx, float number 'yyyy' conversion error!
The string 'yyyy' for conversion to a float was determined to contain one of the following errors:
No numeric characters.
More than one exponent symbol 'E'('e').
More than two sign symbols'.
More than one decimal point.
Alpha characters other than 'E'('e').

Line xx, hex number 'yyyy' conversion error!
On line 'xx', the string 'yyyy' is greater than 10 characters long or contains non-hexadecimal characters. Valid strings start with 0x and contain ASCII characters 0-9, A-F or a-f ( 0x1BF8 ).
Line xx, integer number 'yyyy' conversion error!
The string 'yyyy' for conversion to an integer was determined to contain one of the following errors:
No numeric characters.
Number of numeric characters exceed 10.
The converted number exceeds 0x7FFFFFFF.

Line xx, Invalid argument 'yyyy'!

Line xx, Invalid cam number 'yyyy'! Range 1 to 8.
The CAM number 'yyyy' was evaluated to be less than one or greater than 8.

Line xx, Invalid count or count plus register exceeds range!
The count of registers to be transferred was evaluated to be less than one or when added to the starting register exceeds the maximum register range (1024 registers for CLC/V, 512 registers for CLC/D and CLC/P).

Line xx, Invalid Encoder type 'yyyy', 1=primary, 2=secondary!
The ELS master encoder type 'yyyy' was evaluated to be less than one or greater than 2.

Line xx, Invalid ELS type 'yyyy', range 1 to 4!
The ELS type 'yyyy' was evaluated to be less than one or greater than 4.

Line xx, Invalid sync type 'yyyy', 1=velocity, 2=phase, 3=cam!
The ELS sync type 'yyyy' was evaluated to be less than one or greater than 3.

Line xx, Invalid VME Address 'yyyy'!
The VME address 'yyyy' was evaluated to be less than one or greater than 0xFCFEFFFF.

Line xx, Invalid VME address width 'yyyy'!
The address width 'yyyy' was not found in the table of VME address widths.
{ "A16", "A24", "A32"}

Line xx, Invalid VME byte order 'yyyy'!
VME byte order 'yyyy' must start with 'I' or 'M', 'I' is for Intel order, 'M' is for Motorola. It can be a single character or the name, Intel or Motorola. It is case sensitive, so 'I' and 'M' must be capitalized.

Line xx, Invalid VME count 'yyyy'!
The count of VME objects to transfer 'yyyy' was evaluated to be less than one or greater than 32767.
Line xx, Invalid VME data width 'yyyy'!
The data width 'yyyy' was not found in the table of VME bus widths.
{ "D32", "D16", "D8" }

Line xx, Invalid VME data format 'yyyy'!
The data format 'yyyy' was not found in the table of VME data formats.
{ "I32", "I16", "I8", "U32", "U16", "U8", "F32", "POINT" }

Line xx, Left term 'yyyy' of equation must not be constant!
A calculation must have a variable (Fx, GFx, GIx, Ix) or changeable table element (ABS[1].x, EVT[3].d, etc.) as its term to the left of the equal sign.

Line xx, Maximum number of terms reached.
When parsing the line 'xx', the number of terms exceeded 32. A term is one or more alphanumeric characters followed by a space, comma or other non-alphanumeric character. This error usually only occurs in message statements with many short words. Try a message with fewer words.

Line xx, Maximum size(20) of term exceeded!
While parsing line 'xx' for arguments a string of more than 20 characters was encountered. Arguments and argument labels are limited to 20 characters. Check label length and use of commas between arguments.

Line xx, Message exceeds 80 characters!
The number of characters used in the message exceeds 80 characters. This count includes spaces.

Line xx, missing argument(s)!
One or more additional arguments were expected.

Line xx, missing beginning quotes of message!
On line 'xx', quotes were expected to denote the start of the message. Diagnostic, status and event messages are specified within quotes in textual language programs. Also use quotes when using the "CALC" icon to set an event message.

Line xx, missing closing bracket ']'!
The closing bracket used to denote the end of the index of a data structure, was not found.

Line xx, missing closing curly brace '{'}!
The closing brace used to denote the end of initialization data for a data structure was not found. Other causes are extra arguments or the wrong character.
Line \textit{xx}, missing closing quotes of message!
On line 'xx', quotes were expected to denote the end of the message. Diagnostic, status and event messages are specified within quotes in textual language programs. Also use quotes when using the "CALC" icon to set an event message.

Line \textit{xx}, missing mark name!
The argument of branch command does not start with an alpha character. Check for missing or misspelled argument.

Line \textit{xx}, Parameter <\textit{type}> must be 'A', 'C', 'D' or 'T'
The parameter class was not found to be 'A', 'C', 'D', 'T', or equivalent label. Check for missing or misspelled argument.

Line \textit{xx}, point element 'yyyy' missing or entered with spaces!
On line 'xx', the compiler has not found a "]." in the point string 'yyyy'. It uses this to position to the start of the point element. The point element \{ x, y, z, b, s, a, d, j, e1, e2, e3, e4, r, p, ya, el \} must follow immediately.

Line \textit{xx}, point element 'yyyy' unknown!
The point element 'yyyy' was not found in the point element table, check manual for exact syntax.

Line \textit{xx}, register number 'yyyy' out of range(1-zzzz)!
The register number 'yyyy' is less than one or greater than the maximum register 'zzzz'.

Line \textit{xx}, table or array index out of range 'yyyy'!
The table or array index 'yyyy' is less than one or greater than number declared by DATA/SIZE command or by the default declaration.

Line \textit{xx}, table or array label index out of range 'yyyy'!
The table or array index label 'yyyy' is evaluated to be less than one or greater than number declared by DATA/SIZE command or by the default declaration.

Line \textit{xx}, Task must be 'A', 'B', 'C' or 'D'!
The compiler is expecting a task argument( A, B, C, or D ) and has not found it. This may result from a missing argument or arguments out of sequence.

Line \textit{xx}, too many arguments!
More terms than expected were found following the command. Check for extra arguments, extra commas or terms with spaces in them.
Line xx, unknown mnemonic operator - 'yyyy'!
On line 'xx', the string 'yyyy' is assumed to be a command, but was not found in the list of valid commands. This error is most often generated from textual language programs when the command is misspelled or from incorrect syntax.

Line xx, unknown or out of range variable 'yyyy'!
On line 'xx', the string 'yyyy' is not of the type expected. Check for argument type( float where integer should be used, etc. ) , or for missing or misspelled arguments.

Line xx, unresolved index 'yyyy'!
The index 'yyyy' could not be resolved, check for missing or misspelled label. Label are case sensitive and cannot contain spaces.

Line xx, unresolved index label 'yyyy'!
The index label 'yyyy' could not be resolved to an integer or integer variable, check for missing or misspelled label. Label are case sensitive and cannot contain spaces.

Line xx, unsupported data transfer! VME bus width 'yyyy', VME format 'zzzz', local
The selected VME data transfer is not supported in this product. Check VME format and local format for possible erroneous selection.

Line xx, Unsupported structure transfer!
The data structures equated to each other are not of the same type. The data structure transfers supported are: Point to Point, Event to Event, and Zone to Zone.

Line xx, Valid modes are 1=single axis, 2=ELS synchronized!
The second argument of the "ELS/MODE" command is missing or out of range. This can also be generated if the first argument is invalid and appears as two or more arguments to the compiler.

Line xx, variable table 'yyyy' index unknown!
The closing bracket is missing or other delimiter found in the index term of a variable or register with index format.

Mark table filled - yyyy, reduce number of subroutine calls.
The total number of marks used exceed the table space provided. Marks are the location tags of the start of tasks, event functions and subroutines, or, the destination of a branch or goto. Try to optimize your program to reduce the number of branches. If the problem persist, contact your Rexroth Indramat representative.

Upon successful completion of the compile, the number of marks and labels used are displayed in the completion window.
No main task( A, B, C, or D ) found!
After compiling the program, no marks were found for Task_A, Task_B, Task_C, or Task_D.
One or more of these task marks must be used. If it’s a textual language program, check
spelling and the underscore. The marks for the main tasks are case insensitive.

Sequencing error in output file!
While computing byte offsets for branches and subroutine calls, an unknown command op-code
was encountered. This error can occur in a corrupted Windows memory system or a compiler
bug. Try rebooting your computer and compiling again. If the problem persist, contact your
Rexroth Indramat representative.

Size of program exceeds compiler space!
The compiler has 48k of space available for program development, this error is outputted when
that space is filled. Variables and tables are not included in this space. Try reworking your
program to fit it in the space.

Unable to allocate memory for compiler!
The 2nd pass compiler uses a large block of memory( 48K ) allocated from the Windows
operating system to build the program. When Windows fails to allocate this memory, this error
is outputted. Try closing other applications or rebooting Windows to free needed memory.

Unable to open source file.
This error is issued on failing to open the file "CLCCODE.TXT". Some possible causes are:
- File "CLCCODE.TXT" is not in the "\CLC" directory. This file is created by compiling a
textual or icon program.
- The maximum number of file are already open. DOS file "CONFIG.SYS" configures the
maximum number of files.
- The file is already open and cannot be shared.

Write to file error!
This error is outputted when the number of bytes sent to the output file doesn't match the
number of bytes written in the output file. Check for available hard drive disk space or write
protection on the output file( \clc\progrm\*.exc ).
A1. CLC DDE SERVER

A1.1 Dynamic Data Exchange

The Microsoft Windows operating system specifies a method for transferring data between applications which is called dynamic data exchange (DDE). DDE is a message protocol that developers can use for exchanging data between Windows-based applications. The CLC communication server uses the dynamic data exchange management library (DDEML) which is built on top of the DDE protocol. The DDEML provides services that the message-based DDE protocol does not support. Under the DDEML a client application requests information from a server application, or it sends unsolicited data to the server. The client does this by passing predefined ASCII strings to the server through the DDEML.

Before a client and server can exchange data, they must first agree upon what they are going to talk about. This is done by establishing a conversation. Conversations are defined by a service name and a topic name. The CLC server application uses this information to specify how and who to communicate with. After having established a conversation, the client application can now pass data. This is done by specifying an item name. The item name identifies the specific data to be passed.

There are three basic types of data transactions which can be initiated by the client application. A request transaction is used to obtain data from the server. The server application knows how to obtain the requested information. The second type of transaction is an advise link. After a client application establishes an advise link with a server, it is up to the server to poll the data for changes. If the server finds that the data has changed it will notify the client application. The third type of transaction is a poke. A poke transaction is used to send data for a specific item to the server.
A1.1.1 The Dynamic Data Exchange Server

CLC_DDE is a Windows based Dynamic Data Exchange (DDE) Server application which is used to communicate with Indramat’s CLC motion control cards. It has been implemented using windows dynamic data exchange management library (DDEML).

**Key Features**
- Serial connection to a CLC card with support for an RS485 auto switching adapter.
- Support for a modem connection to a CLC card (AT protocol).
- VME back plane communications from a XYCOM PC (Requires XVE984.DLL).
- VME back plane communications from a GE FANUC Plug & Play PC (Requires VPCMTK.DLL).
- Direct PC AT bus communication to a CLC-P card (Requires CLC_P.DLL).
- Connection for editing a CLC compiled program file off line (Requires CLC_FILE.DLL).
- Demonstration connection for testing client applications off line (Requires DEMO.INI).
- Access to server parameters and status through DDE.
- Supports Request, Advise and Poke transactions.

A1.1.2 Dynamic Data Exchange Interface

A windows application, known as a client, can pass information between other applications known as servers using Dynamic Data Exchange (DDE). A client establishes a conversation with a server specifying a Service and a Topic. Once a conversation has been started, a client may request or send information by specifying an item.

**Service Name**
The CLC communication server supports two DDE service names. The standard service name is CLC_DDE. This should be used for all connections except when connecting to a CLC compiled program file. For this case use CLC_FILE.

**Topic Name**
When the standard service name is used to exchange CLC data, the topic name identifies the method of connection to the CLC card and the card unit number. Valid strings consist of a communication device name and a unit number. Valid device names are SERIAL, AT_MODEM, XYCOM, GE_P&P, DEMO or ISA and valid card unit numbers are '0' to 'F'. Connections which use the CLC_FILE service should specify the CLC program file as the topic name. If the file is not located in the same directory as clc_dde.exe then the complete path should be included. To exchange server data the service name should be CLC_DDE and the topic name should be SERVER. This is the only topic which will not support an advise link. See section SERVER Topic Name.
Example:

"SERIAL_0" Serial connection to a CLC card designated as unit '0'.
"XYCOM_B" Xycom PC in VME rack talking to a CLC_V card designated as unit 'B'.
"ISA_1" PC talking over the ISA bus to a CLC_P card designated as unit 1.
"SERVER" Exchange CLC_DDE server information.

**Item Name**
The item name identifies the specific data to exchange. When exchanging CLC data the item name consists of a string which contains the class, subclass and data identifiers of the information for the CLC card. The strings follow the ASCII serial protocol. Refer to Appendix B. Direct ASCII Communications for an explanation of these codes. When exchanging server data the item name should consist of the section and entry name from the INI file (clc_dde.ini). The two names must be divided by a pipe (‘|’) character. Not all server data has read/write capabilities.

Example:

"RX 0.10" Specifies register 10 in hexadecimal format.
"TP 2.20" Specifies task B parameter 20.
"CP 1.122" Specifies card parameter 122.
"SERIAL|Baudrate" Specifies the baud rate to use for serial connections.

**NOTE:** Serial connections directed at different units will be passed through the VME backplane to the proper unit (CLC-V only). This allows communications with any CLC_V card in the VME rack with only one serial connection.
A1.2 The Communication Servers Main Window

CLC_DDE displays the unit number and current status for the selected CLC control card. To display the status for a different CLC card or to disable this feature, open the server configuration dialog box under the settings menu item. Select the desired connection/unit from the CLC status display combo box.

When CLC_DDE is in an icon state the tip of the arrow will change colors depending on the communication state. A green tip means that the server is actively communicating, and a red tip indicates that the server is in an error state. If the monitored CLC card’s status indicates an error state while the application is an icon, the server window will be restored to the normal state.

When the CLC DDE Server is running, either the icon , or the dialog box below is displayed.

![CLC DDE Server Dialog Box]

If the icon is displayed, double-clicking the icon restores the dialog box. The CLC DDE Server dialog box contains three selections on the main menu bar: File, Settings and DDE.
A1.2.1 Settings Menu - CLC Server Configuration

The CLC Server Configuration allows setting of various system parameters as well as providing performance status information.

Communications

**CLC Status Display:** Selects the CLC device/unit (i.e. serial_0) combination to be displayed in the status window of the server. The request will be inserted into the standard client advise loop queue. This feature can be turned off by selecting “Disable Status”.

**CLC Response Time-out:** The amount of time in seconds that the server will wait for a completed response from the CLC control card before diagnosing a disconnect. The valid range of values is 1-900 seconds.

**CLC Back Plane Relay Time-out:** CLC-V control cards have the ability to redirect incoming serial messages over the VME back plane to other CLC-V cards in the same rack. This allows a host to address multiple control cards with one serial connection. These transmissions may require more time than a direct serial link. The relay time-out value is used for these transactions. The valid range of values is 1-900 seconds.

**Communication Retry Attempts:** The number of times the server will re-send a message before it issues an error. The valid range of values is 0-255.
**Error Handling**

**Intercept CLC Errors And Display:** Checking this box will cause the server to intercept CLC error responses and displayed them in a message box. Request and poke transactions will return failure to the client application. Advise links will remain active, however they will return nothing until the error is resolved. The error response will be written to the error log file if that feature is enabled. If this box is not checked the error string will be returned to the client.

**Make Error Messages System Modal:** Checking this box will cause all server generated message boxes to have system modal attributes. This means that all applications will be suspended until the user responds to the message box. The window can not be forced to the background.

**Log Errors To File:** Checking this box will cause the server to log all server errors to a file. The current system date and time will be associated with each log entry. As a default this feature is not enabled.

View Log File: Pressing this button will cause the current error log file to be displayed in notepad.

**DDE**

**Maximum Conversations:** This is a static display of the maximum number of allowed DDE conversations as specified in the INI file. The server will refuse any DDE connection requests in excess of this value.

**Maximum Advise Items:** This is a static display of the maximum number of allowed DDE advise links as specified in the INI file. The server will refuse any requests for advise links in excess of this value.

**Self Terminate If No Active Conversations:** Checking this box will cause the server to close itself when the last DDE conversation with it has terminated. This is the default state.
A1.2.2 Settings Menu - Serial Communications

The Serial Communications dialog box allows the user to select the serial communication parameters the server will use. When this dialog box is open all communications are suspended. If changes are made to the configuration they will take affect when the “Save” button is pressed.

**Baud Rate:** Check the proper baud rate to use when communicating serially with a CLC card.

**Serial Port:** Select the serial communications port to use on the PC.

**Use Serial Event:** Checking this box causes Windows to notify the server when a completed message is in the receive queue. This will increase the number of serial messages sent over polling for a response. Slower computers may not be able to utilize this feature.

**RS485 Converter:** This option should be used when an RS232 to RS485 converter is present. A delay will be inserted between messages which is equal to at least one character transmission at the selected baud rate. This is necessary to ensure that the CLC card has had sufficient time in which to turn the RS485 transmitter off and enable the receiver. Please note that the converter must toggle the transmitter and receiver automatically, and also that echo back must be disabled.
A1.2.3 Settings Menu - VME Communications

The VME Communications dialog box allows the user to edit parameters which the server uses when talking over the VME bus using a XYCOM embedded PC. When this dialog box is open all communications are suspended. If changes are made to the configuration they will take affect when the “Save” button is pressed. The dynamic link library “XVME984.DLL” must be in the CLC directory or the windows path.

**CLC Parameters**
- *CLC Unit Number*: The CLC unit number for the currently displayed data.
- *Short Address Page*: The address page in short VME memory space where the selected CLC card resides.
- *Base Address Page*: The address page in Standard or Extended memory space where the CLC’s shared RAM is located.

**NOTE**: The default server settings correspond to the default CLC control card settings and should not need to be altered.

**XYCOM Options**
- *VME Handshaking Interrupt*: Select the VME interrupt which all CLC-V control cards should use to terminate a communication response. If this option is not used, the server will poll for a communication response every 55 milliseconds. Refer to your XYCOM owners manual to configure the computers BIOS to acknowledge the selected VME interrupt.
- *Allow Extended VME Addressing (A32)*: Check this box if the XYCOM PC can support A32 addressing.
- *Release Bus Every Cycle*: Check this box if the PC should release the VME bus after every cycle. This will increase communication overhead due to the additional bus arbitration cycles.
A1.2.4 Settings Menu - PC Bus Communications

The PC Communications dialog box allows the user to view CLC status indicators and set communication parameters. When this dialog box is open all communications are suspended. If changes are made to the configuration they will take affect when the “Save” button is pressed. The dynamic link library “CLC_P.DLL” must be in the CLC directory or the windows path.

---

**PC Bus Communications**

- **CLC Status Indicators**
  - **CLC Heart Beat**: This indicator will blink indicating that the selected CLC control card is running.
  - **CLC Executing pROBE+ Monitor**: This indicator will be marked if the selected CLC control card has faulted and is running the pROBE+ monitor.
  - **CLC Is In Parameter Mode**: This indicator will be marked when the selected CLC control card is in parameter mode.
  - **CLC Is In An Error State**: This indicator will be marked when the selected CLC control card is in an error state. Card parameter 122 will contain the specific error message.
  - **CLC Unit Number**: Use this pull down list to select the unit number to display the status indicators for.
  - **Use PC Handshaking Interrupt (IRQ 9)**: When selected, this option will force all CLC-P control cards to terminate communication responses with a PC interrupt (IRQ 9). Hardware jumper S5 must be inserted on the CLC-P card for this option to work properly. If this option is not used, the server will poll for a communication response every 55 milliseconds.

**NOTE**: When using the interrupt option on the CLC-P control card, no other hardware devices may use IRQ 9.
A1.2.5 DDE Menu

DDE Conversations
The DDE Conversations dialog box displays the Conversation, Service and Topic Handles for all of the current DDE conversations. The Item Count column shows the total number of active advise links, request transactions and poke transactions. Double click on a specific conversation entry in order to view the item transaction list. A second method is to select the conversation and then use the “expand” button. This dialog box is useful when creating client applications which talk to the CLC communications server.

DDE Conversation Item Dialog
The DDE Conversation Item dialog box can be used to view the item transaction list for a conversation. The Service name, Topic string, Item string, clipboard Format and Transaction Type are displayed in text format. Use the “Next” and “Previous” buttons to cycle through the current list.
Communication Monitor

The DDE Communication Monitor displays all of the current DDE conversations. The monitor can display DDE requests and/ or responses depending the selection made under the Settings menu.

The active window builds a communications log of all DDE conversations that occur while the monitor is running. Selecting Clear will empty the log. Selecting Stop will stop the conversation monitoring and allow users to scroll through the log. The Monitor window can be resized to enlarge the active viewing area.
A1.3 AT Modem Configuration Dialog

CLC_DDE supports communications with a telephone modem which uses the AT protocol. The server will initiate the modem link and instruct it to dial the desired number by sending standard AT commands. The AT Modem Configuration dialog box is automatically displayed when a DDE conversation which specifies the “AT_MODEM_x” topic is started. The box will again appear when the conversation is terminated. The dialog contains setup data and connection status. To initiate the modem connection first enter the baud rate, serial port and phone number. The next step is to select the “Connect” button and watch the status box. After the sending and receiving modems have connected press the “Cancel” button to close the dialog box.

The modems used for communication must respond to the AT protocol. CLC_DDE will initialize the sending modem and establish a connection with the receiving modem. The receiving modem should be configured in auto answer mode. The CLC card must be configured to the same baud rate as the receiving modem.

Baud Rate:
Select the baud rate to use to talk to the sending modem.

Serial Port:
Select the serial port to use to talk to the sending modem.

Telephone:
Enter the complete phone number to dial including any numbers required to get an outside line. Placing a comma in the number will insert a delay.

Attempt To Connect On Start Up:
Check this box if you wish CLC_DDE to automatically attempt a connection when a conversation is started. The telephone number is saved in the INI file. If this box is not check the user will need to select the “Connect” button.
A1.4 SERVER Topic Name

The “SERVER” topic name allows a DDE client application access to CLC_DDE’s parameter set and status. The server will accept request and poke transactions. When accessing a parameter the client application should specify the section and entry names from the INI file. The two names must be separated by a pipe character (‘|’). When requesting status information the client should use “STATUS” as the section name (i.e. “STATUS|ErrorState”). RW = Read/Write  RO = Read Only

Section: GENERAL

<table>
<thead>
<tr>
<th>Parameter</th>
<th>RW/RO</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Response_Timeout</td>
<td>RW</td>
<td>1-900 Seconds Message response time out.</td>
</tr>
<tr>
<td>Relay_Timeout</td>
<td>RW</td>
<td>1-900 Seconds Message timeout when using VME pass-through.</td>
</tr>
<tr>
<td>Communication_Retry</td>
<td>RW</td>
<td>0-255 Number of times to re-send a message.</td>
</tr>
<tr>
<td>Suspend_Polling</td>
<td>RO</td>
<td>0 or 1 If 1 CLC_DDE status polling will be disabled.</td>
</tr>
<tr>
<td>Display_CLC_Errors</td>
<td>RW</td>
<td>0 or 1 If 1 CLC_DDE will intercept &amp; display CLC Errors.</td>
</tr>
<tr>
<td>Log_Errors</td>
<td>RW</td>
<td>0 or 1 If 1 all server errors will be logged to the error file.</td>
</tr>
<tr>
<td>Modal_Errors</td>
<td>RW</td>
<td>0 or 1 Displayed errors with the system modal attribute.</td>
</tr>
<tr>
<td>Self_Terminate</td>
<td>RW</td>
<td>0 or 1 Close CLC_DDE when last conversation terminates.</td>
</tr>
<tr>
<td>Monitor_List_Size</td>
<td>RW</td>
<td>1-500 # of entries in communication monitor window.</td>
</tr>
<tr>
<td>Editor</td>
<td>RW</td>
<td>256 Characters Name &amp; path of text editor to use to display error log.</td>
</tr>
</tbody>
</table>

Section: SERIAL

<table>
<thead>
<tr>
<th>Parameter</th>
<th>RW/RO</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baudrate</td>
<td>RO</td>
<td>38400..300 Baud rate for serial connection to CLC card.</td>
</tr>
<tr>
<td>Port</td>
<td>RO</td>
<td>1-4 COM port number to use for serial connection.</td>
</tr>
<tr>
<td>Serial_Event</td>
<td>RW</td>
<td>0 or 1 Use serial event option to increase performance.</td>
</tr>
<tr>
<td>RS485_Converter</td>
<td>RW</td>
<td>0 or 1 Activate RS485 adapter code.</td>
</tr>
</tbody>
</table>

Section: VME

<table>
<thead>
<tr>
<th>Parameter</th>
<th>RW/RO</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sustain_Bus</td>
<td>RW</td>
<td>0 or 1 Release every cycle option for XYCOM PC.</td>
</tr>
<tr>
<td>A32_Addressing</td>
<td>RW</td>
<td>0 or 1 Use A32 addressing for XYCOM PC.</td>
</tr>
<tr>
<td>VME_IRQ</td>
<td>RO</td>
<td>0-7 Number of VME IRQ to use ( 0 = disabled ).</td>
</tr>
</tbody>
</table>

Section: AT_MODEM

<table>
<thead>
<tr>
<th>Parameter</th>
<th>RW/RO</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baudrate</td>
<td>RO</td>
<td>9600..300 Baud rate to use to communicate with the modem.</td>
</tr>
<tr>
<td>Port</td>
<td>RO</td>
<td>1-4 COM port number the modem is on.</td>
</tr>
<tr>
<td>Auto_Connect</td>
<td>RW</td>
<td>0 or 1 Initialize &amp; connect on conversation connection.</td>
</tr>
<tr>
<td>Phone</td>
<td>RW</td>
<td>50 Characters Phone number to dial.</td>
</tr>
<tr>
<td>Initialize_Script</td>
<td>RW</td>
<td>100 Characters Script to initialize modem.</td>
</tr>
<tr>
<td>Disconnect_Script</td>
<td>RW</td>
<td>100 Characters Script to disconnect modem.</td>
</tr>
<tr>
<td>Dial_Prefix</td>
<td>RW</td>
<td>50 Characters Script to send to modem before phone number.</td>
</tr>
<tr>
<td>Escape_Sequence</td>
<td>RW</td>
<td>50 Characters Script to send modem to return to command mode.</td>
</tr>
</tbody>
</table>
**Section: PC**

| PC_IRQ          | RO  | 0 or 1 | if 1 use PC interrupt for communications. |

**Section: DDE**

| Status          | RO  | 200 Characters | CLC_DDE’s status request item. |
| Max_Conversations | RO  | 1-3274        | Maximum allowed conversations. |
| Max_Advise_Items | RO  | 1-3500        | Maximum allowed advise items. |

**Section: STATUS**

| ErrorState | RO  | 0 or 1 | If 1 CLC_DDE is issuing an error. |
| ErrorText  | RO  | 256 Characters | Error text message CLC_DDE is displaying. |
| RequestState | RO  | 0 or 1 | If 1 CLC_DDE is actively communicating. |
B. DIRECT ASCII COMMUNICATION

B.1 Overview

The Indramat CLC Motion Control can send and receive drive parameters, system parameters, user programs, and tables through its serial port. By using the text-based protocol described in this section, a wide variety of devices and programs can communicate with the CLC. The protocol also supports ASCII PC ISA/ESIA and VME bus communication.

B.2 CLC Communication Protocol

The ASCII protocol format for the CLC is designed so that all serial transmissions are similar in structure, facilitating simple coding/decoding routines. The protocol is the same for sending or receiving so that data may be easily handled and tracked.

All serial communications use the following standard protocol template. Information is classified using a three level system: command class, subclass, and data identifier.

```
>1 CS s.n 0123...CDEF $cs\r
   |__________|_ Variable Length Data Field
   |__________|_ Two Digit ASCII Checksum
   |__________|_ Number Within Set Data Identifier
   |__________|_ ASCII period (0x2E)
   |__________|_ Set Type Data Identifier
   |__________|_ optional ASCII space character (0x20)
   |__________|_ Command Subclass
   |__________|_ Command Class
   |__________|_ required ASCII space (0x20)
   |__________|_ Network Address (i.e. card position identifier)
   |__________| Start Character (0x3E)
```

Start Character
The beginning of each message is identified by the " > " character (ASCII 62 decimal, 3E hex).

Network Address
The network address can be used to support data transfer across a bus, or communication system, to multiple CLC cards. The network address must be followed by an ASCII space character (0x20). The VME bus version of the CLC uses the hexadecimal characters "0" through "F" to address corresponding VME cards assigned to positions 1 through 16. An ASCII space (" ") character may be used to address the VME card containing the Host Communication serial port.
Data Identifier
The Data Identifier field is a variable length field used to identify the data set being sent or requested. This identifier is in the format "s.n" or "s.n.x", where; "s" = the set identifier, may be a program handle, drive number, task ID, etc; and "n" = the numeric identifier used for items such as parameter numbers, table indexes and register numbers. The third field, called a "Step x" identifier, is used for parameter lists. No white space is allowed between the identifiers and the separating dot operators.

Variable Length Data
This field contains the actual data being sent to or received from the CLC. All subclasses have read and write capability. The same strings can be used for both responses and downloads.

B.2.1 Reading Data from the CLC
To read data, the data portion of the string sent from the Host is empty. For example:

*The Host system requests a Drive Status Message*

```
>1 DP 1.95 \r\n
|_ No data sent, requesting current data
```

*The CLC responds with the current status of the specified drive:*

```
>1 DP 1.95 302 Position Mode Encoder 1 $cs\r\n
|_ Data (status message)
```

B.2.2 Writing Data to the CLC
To write data, send data in the data portion with the same starting protocol. Data received from the CLC can be sent in the same format. The CLC responds with an acknowledgment or an error message. For example:

*The Host sends the CLC the DDS drive Kv Parameter for drive 1:*

```
>1 DP 1.104 1.00$cs\r\n
|_ New data sent
```

*The CLC has successfully accepted the parameter, since no error message was returned:*

```
>1 DP 1.104 $cs\r\n
|_ No message: data stored successfully
```
B.2.3 Communication Errors

If there is a checksum error, a format error, or an error in the data sent to the CLC or the drive, the CLC returns an error string in the data field. The string begins with a "!" character, followed by an error code and a descriptive message. Communication error codes and messages are listed at the end of this section. For example:

>1 DF 1.104 !05 Greater than maximum value $cs\r\n | | _ Error message
| | _ Error Code (decimal)
| _ Error indicator "!

B.2.4 Checksum

A CLC checksum is sent as two ASCII hexadecimal digits preceded by an ASCII '>'. The checksum is optional when requesting data from the CLC. When sending data to the CLC the checksum is required, unless it is disabled in the checksum parameter.

To compute the checksum, add the hexadecimal ASCII values of all of the characters before the '>', including the starting " > " character. Then add the most significant digit of the checksum to the two least significant digits. Negate (' +/-' on a calculator) this value to form the two’s complement. The checksum transmitted to the CLC is the 2 least significant digits of this number. This is the same checksum method used with the Indramat CLM Positioning Control and Station Operator Terminal (SOT). For example:

Checksum on >1 AP 1.1 2

\[\begin{align*}
3E & \text{ hex} \\
31 & \\
20 & \\
41 & \\
50 & \\
20 & \\
31 & \\
2E & \\
31 & \\
20 & \\
+32 & \\
\hline
222 & \text{ (sum)} \\
\hline
22 & \text{ (two least significant digits)} \\
+2 & \text{ (add two)} \\
\hline
24 & \text{ (two's complement) - Checksum = 24 (DC in hexadecimal)}
\end{align*}\]
B.2.5 End of Message

An ASCII carriage return (CR = 13 in decimal or 0d hexadecimal) and linefeed (LF, 10
decimal or 0a hex) combination is used for terminal compatibility. This document uses the
notation '\r'(return) and '\n'(newline), as used with the C programming language,
interchangeably with the CR LF notation. The CLC always sends a CR LF combination, but
will accept either a single LF or a CR LF from the Host device.

B.2.6 Backspaces and White spaces

The ASCII backspace character (8 decimal or hex) erases the previous character from the
CLC's serial buffer except at the start of a message. This is useful for editing strings entered at
a terminal. Also, any whitespace character (tab or space) can be used as a delimiter in strings.
Whitespaces between fields or at the end of a message are discarded by the CLC.

B.2.7 Numeric Data Formats

The CLC sends numeric data in ASCII parameter-specified units and scaling format. The
format of floating point data depends on the data's use and how and where it is stored. Floating
point data of fixed precision (e.g., drive data) uses fixed resolution. The resolution of data
stored on the CLC card (i.e., local or global variables) depends on the storage precision used;
32 or 64-bit.

Floating point data that is too large or small to be printed in decimal format is represented in
scientific notation. Hexadecimal data is sent and received with an '0x' prefix. Binary data is
represented as a 16 digit string of ASCII "1" or "0" characters. For example:

Floating point position data:  0.0100  123.4567  -12.0000  12.3e+16
(resolution = 0.0001 units)

Integer data:  0 1000  -10
Hexadecimal data:  0x12AB  0x1234ABCD
Binary data:  0000111100001111

B.2.8 Format of Data Sent to the CLC

Any resolution can be used for data sent to the CLC. Numbers may be padded with zeros or
spaces at either end as a visual formatting aid when entering data from a terminal. Padding
applies to data identifiers as well as the data field.

The resolution of the data stored on the CLC is the resolution of the data the CLC sends to the
Host on request. Floating point numbers may also be sent in scientific notation.
B.3 Command Classes/ Subclasses

The tables below list the command class identifier and the subclass identifier for each of the available subclasses within the command class.

B.3.1 Parameters

<table>
<thead>
<tr>
<th>Command Class</th>
<th>Command Subclass</th>
</tr>
</thead>
<tbody>
<tr>
<td>A - CLC Axis Parameters</td>
<td>A - Attributes</td>
</tr>
<tr>
<td></td>
<td>B - Block List Parameter</td>
</tr>
<tr>
<td></td>
<td>D - Lists/Tables</td>
</tr>
<tr>
<td></td>
<td>H - Upper Limit</td>
</tr>
<tr>
<td></td>
<td>L - Lower Limit</td>
</tr>
<tr>
<td></td>
<td>P - Parameter Data</td>
</tr>
<tr>
<td></td>
<td>T - Name Text</td>
</tr>
<tr>
<td></td>
<td>U - Units Text</td>
</tr>
<tr>
<td>C - CLC Card Parameters</td>
<td>(Same subclasses as CLC Axis Parameters)</td>
</tr>
<tr>
<td>D - SERCOS Drive Parameters</td>
<td>(Same subclasses as CLC Axis Parameters)</td>
</tr>
<tr>
<td>T - Task Parameters</td>
<td>(Same subclasses as CLC Axis Parameters)</td>
</tr>
</tbody>
</table>

See 4 Drive and CLC Parameters and Subclasses

B.3.2 Variables

<table>
<thead>
<tr>
<th>Command Class</th>
<th>Command Subclass</th>
</tr>
</thead>
<tbody>
<tr>
<td>I - Integer Variables</td>
<td>P - Data</td>
</tr>
<tr>
<td></td>
<td>T - Text Label</td>
</tr>
<tr>
<td>F - Floating Point Variables</td>
<td>(Same subclasses as Integer Variables)</td>
</tr>
<tr>
<td>G - Global Integer Variables</td>
<td>(Same subclasses as Integer Variables)</td>
</tr>
<tr>
<td>H - Global Floating Point Variables</td>
<td>(Same subclasses as Integer Variables)</td>
</tr>
</tbody>
</table>

See 6 User Program Variables
B.3.3  PID

<table>
<thead>
<tr>
<th>Command Class</th>
<th>Command Subclass</th>
</tr>
</thead>
<tbody>
<tr>
<td>M - PID loop</td>
<td>B - Set Point variable, Feedback variable, Output variable, First variable of control block</td>
</tr>
<tr>
<td></td>
<td>E - Calculated set point</td>
</tr>
<tr>
<td></td>
<td>F - Calculated feedback</td>
</tr>
<tr>
<td></td>
<td>G - Calculated output</td>
</tr>
<tr>
<td></td>
<td>J - Loop time</td>
</tr>
<tr>
<td></td>
<td>L - List of valid PID loop numbers</td>
</tr>
<tr>
<td></td>
<td>R - Control register</td>
</tr>
<tr>
<td></td>
<td>S - Status register</td>
</tr>
<tr>
<td></td>
<td>T - Type</td>
</tr>
</tbody>
</table>

See B7 PID

B.3.4  Point Tables

<table>
<thead>
<tr>
<th>Command Class</th>
<th>Command Subclass</th>
</tr>
</thead>
<tbody>
<tr>
<td>X - Absolute Point Table</td>
<td>X - X coordinate</td>
</tr>
<tr>
<td></td>
<td>Y - Y coordinate</td>
</tr>
<tr>
<td></td>
<td>Z - Z coordinate</td>
</tr>
<tr>
<td></td>
<td>B - Blend Radius</td>
</tr>
<tr>
<td></td>
<td>S - % of Max Speed</td>
</tr>
<tr>
<td></td>
<td>A - % of Max Acceleration</td>
</tr>
<tr>
<td></td>
<td>D - % of Max Deceleration</td>
</tr>
<tr>
<td></td>
<td>J - Jerk limiting %</td>
</tr>
<tr>
<td></td>
<td>1 - Event 1</td>
</tr>
<tr>
<td></td>
<td>2 - Event 2</td>
</tr>
<tr>
<td></td>
<td>3 - Event 3</td>
</tr>
<tr>
<td></td>
<td>4 - Event 4</td>
</tr>
<tr>
<td></td>
<td>R - Roll (6-axis robot)</td>
</tr>
<tr>
<td></td>
<td>P - Pitch (6-axis robot)</td>
</tr>
<tr>
<td></td>
<td>W - Yaw (6-axis robot)</td>
</tr>
<tr>
<td></td>
<td>E - Elbow (6-axis robot)</td>
</tr>
<tr>
<td></td>
<td>V - List of all user defined labels</td>
</tr>
<tr>
<td></td>
<td>L - List all of the above in one row</td>
</tr>
</tbody>
</table>

Y - Relative Point Table (Same subclasses as X Absolute Point Table)

See B8 Point Tables
B.3.5 Event Tables

<table>
<thead>
<tr>
<th>Command Class</th>
<th>Command Subclass</th>
</tr>
</thead>
<tbody>
<tr>
<td>E - Event Table</td>
<td>A - Argument</td>
</tr>
<tr>
<td></td>
<td>D - Direction</td>
</tr>
<tr>
<td></td>
<td>F - Function</td>
</tr>
<tr>
<td></td>
<td>L - List all elements in one row</td>
</tr>
<tr>
<td></td>
<td>M - Message</td>
</tr>
<tr>
<td></td>
<td>S - Status</td>
</tr>
<tr>
<td></td>
<td>T - Type</td>
</tr>
</tbody>
</table>

See B9 Event Tables

B.3.6 Program Communication

<table>
<thead>
<tr>
<th>Command Class</th>
<th>Command Subclass</th>
</tr>
</thead>
<tbody>
<tr>
<td>P - Program</td>
<td>A - Activate Program</td>
</tr>
<tr>
<td></td>
<td>D - Upload/Download</td>
</tr>
<tr>
<td></td>
<td>E - Delete Program</td>
</tr>
<tr>
<td></td>
<td>F - Event Function List</td>
</tr>
<tr>
<td></td>
<td>H - List of Program Headers</td>
</tr>
<tr>
<td></td>
<td>I - Data change identification</td>
</tr>
<tr>
<td></td>
<td>L - Max number of rows in sequence list</td>
</tr>
<tr>
<td></td>
<td>N - Name of Program</td>
</tr>
<tr>
<td></td>
<td>Q - Max number of rows in sequence table</td>
</tr>
<tr>
<td></td>
<td>R - Header Record, Start Upload</td>
</tr>
<tr>
<td></td>
<td>S - Number of function slots available</td>
</tr>
<tr>
<td></td>
<td>V - List of Program Variable Labels</td>
</tr>
<tr>
<td></td>
<td>W - Header Record, Start Download</td>
</tr>
<tr>
<td></td>
<td>X - Transfer Tables between Programs</td>
</tr>
</tbody>
</table>

See B10 CLC Program Communication
### B.3.7 Functions

<table>
<thead>
<tr>
<th>Command Class</th>
<th>Command Subclass</th>
</tr>
</thead>
<tbody>
<tr>
<td>S - Function Class</td>
<td>A - Access Type</td>
</tr>
<tr>
<td></td>
<td>H - Maximum Value</td>
</tr>
<tr>
<td></td>
<td>L - Minimum Value</td>
</tr>
<tr>
<td></td>
<td>N - Argument Label</td>
</tr>
<tr>
<td></td>
<td>R - Argument attributes in a row format</td>
</tr>
<tr>
<td></td>
<td>T - Function name</td>
</tr>
<tr>
<td></td>
<td>V - Local variable attributes in row format</td>
</tr>
<tr>
<td></td>
<td>Y - Type of data</td>
</tr>
</tbody>
</table>

*See B11 Function Class (S)*

### B.3.8 I/O Registers

<table>
<thead>
<tr>
<th>Command Class</th>
<th>Command Subclass</th>
</tr>
</thead>
<tbody>
<tr>
<td>R - I/O Registers</td>
<td>B - Current State in Binary</td>
</tr>
<tr>
<td></td>
<td>C - Forcing State Change</td>
</tr>
<tr>
<td></td>
<td>D - Current State in Decimal</td>
</tr>
<tr>
<td></td>
<td>E - Erase all Forcing Masks</td>
</tr>
<tr>
<td></td>
<td>F - Forcing Mask</td>
</tr>
<tr>
<td></td>
<td>S - Binary Forcing State</td>
</tr>
<tr>
<td></td>
<td>X - Current State in Hexadecimal</td>
</tr>
</tbody>
</table>

*See B12 Input/Output Registers*

### B.3.9 Sequencer Data

<table>
<thead>
<tr>
<th>Command Class</th>
<th>Command Subclass</th>
</tr>
</thead>
<tbody>
<tr>
<td>L - Sequencer List</td>
<td>T: Name of Sequence list</td>
</tr>
<tr>
<td></td>
<td>P: Print or Store Sequence tables</td>
</tr>
<tr>
<td>Q - Sequence Table</td>
<td>1: Argument 1</td>
</tr>
<tr>
<td></td>
<td>2: Argument 2</td>
</tr>
<tr>
<td></td>
<td>3: Argument 3</td>
</tr>
<tr>
<td></td>
<td>4: Argument 4</td>
</tr>
<tr>
<td></td>
<td>5: Argument 5</td>
</tr>
<tr>
<td></td>
<td>F - Function number</td>
</tr>
<tr>
<td></td>
<td>P - Print/Store function numbers and arguments</td>
</tr>
<tr>
<td></td>
<td>T - Table Name</td>
</tr>
</tbody>
</table>

*See B13 Sequencer Data*
B.3.10 Zones

<table>
<thead>
<tr>
<th>Command Class</th>
<th>Command Subclass</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z - Zones</td>
<td>S: Status</td>
</tr>
<tr>
<td></td>
<td>A: Upper X Coordinate</td>
</tr>
<tr>
<td></td>
<td>B: Upper Y Coordinate</td>
</tr>
<tr>
<td></td>
<td>C: Upper Z Coordinate</td>
</tr>
<tr>
<td></td>
<td>D: Lower X Coordinate</td>
</tr>
<tr>
<td></td>
<td>E: Lower Y Coordinate</td>
</tr>
<tr>
<td></td>
<td>F: Lower Z Coordinate</td>
</tr>
<tr>
<td></td>
<td>L: List all elements in one row</td>
</tr>
</tbody>
</table>

B.4 Drive and CLC Parameters and Subclasses

The CLC System/Axis, Task, and Drive parameters follow the same general format. The subclasses are data elements (data, name, units, etc.) as specified in SERCOS. CLC parameters include system configuration data that is entered during the configuration of the system, as well as continuously updated system status values and messages that monitor system operation.

CLC card parameters are accessed using parameter set C1.

CLC axis parameters use the parameter sets A1 through A8.

DDS drive parameters are accessed using parameter sets D1 through D8.

CLC task parameters use the parameter sets TA through TD (for user program tasks A - D).

Additional information on the parameter sets can be found in the section on CLC Parameters in this manual, and in the DDS-2 Drive and SERCOS manuals.

For example:

```
>1 CP 1.122 $cs\r\n
|| _number
|| _set: Axis, Task, or Drive Address
|| _subclass: Parameter data, name Text, High or Low limits, Attributes
|| _class: type of parameter
```

B.4.1 Parameter Data Subclass

The P subclass specifies the actual parameter data, sent and received in ASCII format according to its attributes (decimal, hexadecimal, text, etc.).
B.4.2 Name Text Subclass

The T subclass provides the name of the parameter, provided as a text string in the language selected for the CLC. The ability to specify actual text for the parameter name permits the host software to be independent of CLC or Drive parameter updates.

B.4.3 Units Text Subclass

The U subclass returns the system units, "in" (inches) or "mm" (millimeters), as an ASCII text string.

B.4.4 Upper Limit, L: Lower Limit Subclasses

The H and U subclasses return the range of permissible data entry that is set by the DDS drive or CLC for numeric data. Limits are always returned as floating point type data.

B.4.5 Attribute Subclass

The A subclass request a hexadecimal longword (16 bits). Bits in this longword are set for data type and scaling according to the SERCOS specification. The attribute data is available for informational purposes, or may be used to detect if a SERCOS parameter is a command or a status value.

B.4.6 Parameter Lists Subclasses

The D subclass is used to request lists of parameters. Since new versions of DDS drives and the CLC may expand or change the parameter sets, lists of all parameters and all required parameters can be uploaded by the Host program. Parameters such as the DDS oscilloscope function also use variable-length lists. Parameter list formats are described later in this section.

B.4.7 SERCOS Parameter Sets

The SERCOS specifications allow a digital drive to have both Standard and Product Specific Parameter Sets. The Standard parameters are accessed by the parameter number (E.G., 1.95). Product Specific Parameters can be accessed using a 'P' prefix, which adds 32768 to the parameter number. For example, Parameter P-0-0005 can also be accessed as "1.32773" or "1.5P".

Examples:

The Host requests the name, data, and units for Drive 1 parameter 123:

>1 DT 1.123 \r\n ;request drive 1 text name for parameter 123
>1 DP 1.123 \r\n ;request drive 1 parameter data
>1 DU 1.123 \r\n ;request drive 1 units of measurement

The DDS drive responds, through the CLC:

>1 DT 1.123 Feed Constant $cs\r\n ;the name is "Feed Constant"
>1 DP 1.123 6.2832 $cs\r\n ;the parameter value is 6.2832
>1 DU 1.123 mm $cs\r\n ;the measurement units are in mm

B.5 Parameter Lists

Some CLC functions and parameters, and SERCOS parameters, are implemented as variable-length data lists. Lists of parameters are used to determine all the parameters present on the DDS drive or CLC card, and to classify or request parameters by function or type. The DDS-2 oscilloscope function data tables are accessed as parameter lists. Sequence numbers are used to list each parameter in the list, allowing other transmissions to the CLC during the list upload.

List Parameter Command:

>1 xD a.x.n \r\n
|   |   |   | Step: Sequence number (0 to length+1)
|   |   |   | Number: Parameter number
|   |   | Set: Axis, drive, or task number
|   |   | Subclass: Command, List Parameter or Table
|   | Class: A=axis, C= card, D=drive, T=task

B.5.1 Listing a Parameter

To request a parameter list, the Host sends the list parameter command with the sequence number = 0 to the CLC. The CLC responds with the sequence number replace with a count of the number of items in the list. The Host then requests each list item sequentially, beginning with sequence number = 1. The sequence number is then incremented by one and the request repeated until all needed items or the entire list has been received.

The CLC requires parameter list sequence numbers to be incremented sequentially. If an error occurs, the request for the current list item may be immediately repeated, allowing the Host to request missed data and ensuring that the data is sent in the proper order. The CLC will respond with an error if sent an invalid sequence number.

At the end of the upload, the Host must signal the CLC to close the list by sending a sequence number equal to the length of the list + 1.
If required, more than one parameter list can be active at one time. The Host must always close a list when it is finished since each open list uses CLC resources.

Example of Parameter List request:

0) The Host requests a List Parameter:
   >1 DD 1.17.0 \r\n   ;Parameter S-0-17

   The CLC responds with length of the list:
   >1 DD 1.17.0 180 $cs\r\n   ;180 parameters in list

1) The Host requests the first parameter in the list:
   >1 DD 1.17.1 \r\n
   The CLC responds with the first parameter:
   >1 DD 1.17.1 44 $cs\r\n
2) The Host requests the second parameter in the list:
   >1 DD 1.17.2 \r\n
   The CLC responds with the second parameter:
   >1 DD 1.17.2 104 $cs\r\n
   ...

180) The Host continues to sequentially request items in list.

181) The Host closes the list by sending a sequence number = the list length+1

   >1 DD 1.17.181 \r\n
   The CLC acknowledges the end of the list:
   >1 DD 1.17.181 !19 List is finished $cs\r\n
B.5.2 Parameter List Block Transfer

Classes: C, A, D, T  
Subclass: B  
Data Type: List of space-delimited strings with ASCII integers or floats

For faster communications, the CLC can send and receive parameter lists 16 elements at a time. Drive parameter lists allowing block transfer include cam tables, oscilloscope data, and any other non-text parameter list. The ‘B’ subclass works similar to the ‘D’ parameter list subclass, but instead of sending one item at a time, 16 elements are sent.

> u, xB a.s.n \r\n  ||  ||  || Step: Sequence number (0 to length +1)  
  ||  ||  _ Number: Parameter number  
  ||  ||  _ Set: Axis, drive, or task number  
  ||  _ Subclass: Command, List Parameter or Table  
  _ Class: A=axis, C = system, D=drive, T=task

Requesting a Block List Parameter
To request the start of a list, the Host sends sequence number 0 to the CLC. The CLC responds with the number of elements in the list.

The number of steps in the list is equal to ((elements +15)/ 16). The Host requests this number of steps from the list until the list is finished.

The data in the response strings is space delimited. Floating point and decimal values are scaled the same as when they are printed individually.

If the number of elements in the list is not evenly divisible by 16, the CLC will fill the last response string with space-delimited zeros for each remaining element. If the data cannot be printed in less than 220 characters, the error message “!55 List or string is too long” is issued.

The CLC requires that step numbers be incremented by one, but any previous step may be repeated. This allows the host to request any missed data and ensures that the data is sent in the proper order. For example, the sequence (1, 2, 3, 4) is valid, but (1, 2, 3, 5) is not. If an invalid step number is sent, the CLC responds with an error.

At the end of the upload, the Host must close the list by sending a sequence number equal to (length of list + 1). The Host must always close a list when it is finished since each new list uses CLC resources.
Example:

0) Host requests a list parameter using block transfer
   > 1 DB 1.32840.0 \r\n   ;Parameter P-0-72 (cam table 1)
   CLC responds with the number of elements in the list:
   > 1 DB 1.32840.0 1024 $cs\r\n   ;1024 points in cam table = 64 steps

1) Host requests first 16 elements in list:
   > 1 DB 1.32840.1 \r\n   CLC responds with first 16 elements:
   > 1 DB 1.32840.1 0.0 0.0015 0.002 0.01 0.015 --11 more elements....-- \r\n
2) Host requests elements 17-32:
   > 1 DB 1.32840.2 \r\n   CLC responds with next 16 elements
   > 1 DB 1.32840.1 20.0 20.0015 --14 more elements....-- \r\n
3-64) Host continues to request items in list as above.

65) To close the list, host sends sequence number (steps+1)
   > 1 DB 1.32840.65 \r\n   CLC acknowledges end of list:
   > 1 DB 1.32840.65 !19 List is finished $cs\r\n
Sending a Block List Parameter
To start sending a block list, the Host sends sequence number 0 to the CLC, along with the number of elements to be sent. The number of steps in the list is equal to ((elements +15)/ 16). The Host sends this number of steps from the list until the list is finished.

The data in the strings must be space delimited. The host can send the data with any resolution, with or without decimal point.

If the number of elements in the list is not evenly divisible by 16, the host must fill the last string with space-delimited zeros for each remaining element.

If the number of elements in the string is less than 16, the CLC responds with the message “!54 List or String is too short”. If the length of the data portion of the string sent to the CLC (minus protocol header, checksum, and terminator) is greater than 220 characters, the CLC responds with the message “!55 List or string is too long”.

At the end of the download, the Host must close the list by sending a sequence number equal to (length of list + 1). The string for this step must include at least one data element. For simplicity, the host can send 16 space-delimited zeros.
Example:
0) Host starts sending a list parameter using block transfer
   >1 DB 1.32840.0 1024 $cs\r\n   ;Parameter P-0-72 (cam table 1)
   ;1024 points in cam table = 64 steps
   CLC responds with an acknowledgment:
   >1 DB 1.32840.0 $cs\r\n
1) Host sends first 16 elements in list:
   >1 DB 1.32840.1 0.0 0.0015 0.002 0.01 0.015 --11 more elements...-- $cs\r\n   CLC acknowledges:
   >1 DB 1.32840.1 $cs\r\n
2-64) Host continues to send items in list as above.

65) To close the list, host sends sequence number (steps + 1), with string having at least one zero.
   >1 DB 1.32840.65 0.0 $cs\r\n   CLC acknowledges end of list:
   >1 DB 1.32840.65 !19 List is finished $cs\r\n
Parameter List Transfer of CLC Cams
When CLC-based cams are sent to and received from the CLC, both X and Y values are transferred. Therefore, when the block data transfer method is used, only 8 rows of the cam file are transferred per string rather than 16. Therefore, the steps in the list is equal to ((elements + 7)/ 8). The data in the string is transferred as: “X1 Y1 X2 Y2 X3 Y3”, etc. If the CLC returns a count of 1025, the host must ask for 129 strings, plus the closing sequence.

Example:
0) Host requests CLC cam #1
   >1 CB 1.3101.0 \r\n   ;Parameter C-0-3101 (cam table 1)
   >1 CB 1.3101.0 1025 \r\n   ; CLC indicates 1025 points (129 steps)

1-129) Host requests data and CLC sends
   >1 CB 1.3101.1 \r\n   >1 CB 1.3101.1 0.0 0.0015 0.002 0.01 --12 more ...-- $cs\r\n
130) Host closes list
   >1 CB 1.3101.130 \r\n   >1 CB 1.3101.130 !19 List is finished $cs\r\n
B.6 User Program Variables

The CLC maintains a unique set of integer and floating point variables for each user program. An additional set of integer and floating point global variables is not related to a specific program and may be accessed by any program or device on the bus. User variable data can be exchanged between the Host and the CLC using the same format as the floating point and integer parameters. The current value of a variable is obtained and changed using the P subclass.

Format:

>1 IP h.xx
   |   |   | _ number: variable table index number
   |   | _ set: Program handle
   | _ subclass: P=Send/receive Data, T=print text label
   | _ class: I=Integer Variable, F=Float Variable
     G=Global Integer,   H=Global Float

The user program handle provides access to the variables for any CLC resident user program. Use the program handle "0" to access the active program's variables.

B.6.1 'P': Data

Type: Floating Point ("F") or Integer ("I")

Data in a CLC variable table is accessed by supplying the class (I or F), and the numeric index (e.g. 1 for I[1] or 15 for F[15]) of the desired variable. The variable number "0" is used to request a count of the variables used by the selected program.

B.6.2 'T': Label Text

Type: String

The text label for any variable can be obtained by using the T subclass. If no text label is found, an ASCII space (" ") character is returned. Since the program labels are fixed when the program is compiled, labels cannot be changed with this command.

Examples:

The Host requests the number of integer variables used by program 1:
>1 IP 1.0 \r\n
The CLC responds with:
>1 IP 1.0 20\r\n ;20 variables

The Host sends floating point data, 123.456 to Variable F12 for the program with handle 2:
>1 FP 2.12 123.456 $cs\r\n
The CLC acknowledges with:
>1 FP 2.12 $cs\r\n
The Host requests the label name for Variable I20 for the current program:
>1 IP 0.20 \r\n
The CLC returns the name "count":
>1 IP 0.20 count\r\n
B.7 PID

PID information is only available from the active program.

B.7.1 Getting the assigned variables.

The host request the control register of PID loop 2 of active program.
>0 MR 0.2 $cs\r\nThe CLC responds with register number, 0 if not assigned.
>0 MR 0.2 121 $cs\r\n
The host request the status register of PID loop 2 of active program.
>0 MS 0.2 $cs\r\nThe CLC responds with register number, 0 if not assigned.
>0 MS 0.2 122 $cs\r\n
The host request the variables of PID loop 2 of active program.
>0 MB 0.2 $cs\r\nThe CLC responds with set point variable, feedback variable, output variable, and start of control block.
>0 MB 0.2 F70 F71 F72 F143 $cs\r\n
The host request the valid PID loop numbers of active program.
>0 ML 0.0 $cs\r\nThe CLC responds with loop numbers 2,3,7,8,9. Other loops are not used
>0 ML 0.0 2 3 7 8 9 $cs\r\nThe CLC responds with 0. No loops are not used
>0 ML 0.0 0 $cs\r\n
Control Blocks Variables
The control block variables are defined by the type of PID loop. The usage for type 1 is given above.
B.8 Point Tables

Each coordinated motion program stored on the CLC card has a unique set of absolute and relative points used to define the endpoints of the standard geometry segments. Each point contains a number of descriptive elements. All the elements of a point in either the Absolute or Relative Tables may be accessed individually or as a list. The available elements are shown in the list of Command Classes and Subclasses at the beginning of this section, and are described in the section on Programming Elements.

General format:

```
>1 XS h.nn \r\n   |   |   number: Point number
   |   |   set: Program handle
   |   subclass: Speed request
   class: X=ABS, Y=REL
```

The program handle selects which of the user programs is accessed. To request the program points for the active program, use a "0" as the handle.

Point data is accessed by specifying a point number, that is used as a numeric index into the specified point table (E.G., 1 for ABS[1] or 21 for REL[21], the absolute or relative table is specified by the class). To request the total number of points in a table, use a "0" for the point index number.

B.8.1 Point Table Data

The X, Y, Z, R, P, and W coordinate subclasses are in the same format as floating point variables and parameters. The S, A, D, and J subclasses are in "xxx.x" format. Event subclasses 1, 2, 3, 4, and E are in the same format as integer variables. All subclasses have read/write access. The V subclass is a List of all user defined labels.
The following table lists each subclass, the type of units, the data type, and the data size.

<table>
<thead>
<tr>
<th>Subclass</th>
<th>Units</th>
<th>Type</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>X - X-coordinate</td>
<td>distance</td>
<td>Floating</td>
<td>32-bit</td>
</tr>
<tr>
<td>Y - Y-coordinate</td>
<td>distance</td>
<td>Floating</td>
<td>32-bit</td>
</tr>
<tr>
<td>Z - Z-coordinate</td>
<td>distance</td>
<td>Floating</td>
<td>32-bit</td>
</tr>
<tr>
<td>B - Blend Radius</td>
<td>distance</td>
<td>Floating</td>
<td>32-bit</td>
</tr>
<tr>
<td>S - Speed</td>
<td>% of max speed</td>
<td>Integer</td>
<td>8-bit</td>
</tr>
<tr>
<td>A - Acceleration</td>
<td>% of max acceleration</td>
<td>Integer</td>
<td>8-bit</td>
</tr>
<tr>
<td>D - Deceleration</td>
<td>% of max deceleration</td>
<td>Integer</td>
<td>8-bit</td>
</tr>
<tr>
<td>J - Jerk Limit</td>
<td>percent</td>
<td>Integer</td>
<td>8-bit</td>
</tr>
<tr>
<td>1 - Event 1</td>
<td>event number</td>
<td>Integer</td>
<td>16-bit</td>
</tr>
<tr>
<td>2 - Event 2</td>
<td>event number</td>
<td>Integer</td>
<td>16-bit</td>
</tr>
<tr>
<td>3 - Event 3</td>
<td>event number</td>
<td>Integer</td>
<td>16-bit</td>
</tr>
<tr>
<td>4 - Event 4</td>
<td>event number</td>
<td>Integer</td>
<td>16-bit</td>
</tr>
<tr>
<td>R - Roll</td>
<td>degrees</td>
<td>Floating</td>
<td>32-bit</td>
</tr>
<tr>
<td>P - Pitch</td>
<td>degrees</td>
<td>Floating</td>
<td>32-bit</td>
</tr>
<tr>
<td>W - Yaw</td>
<td>degrees</td>
<td>Floating</td>
<td>32-bit</td>
</tr>
<tr>
<td>E - Elbow State</td>
<td>none</td>
<td>Integer</td>
<td>8-bit</td>
</tr>
</tbody>
</table>

Examples:

>1 YJ 0.12 \r\n ;host requests the jerk limit for point REL[12] in the active program.

>1 YJ 0.12 50.0 $cs\r\n ;CLC responds

>1 X1 3.1 2 $cs\r\n ;host selects event 2 for ABS[1]'s first event for program 3.

>1 X1 3.1 $cs\r\n ;CLC acknowledges
B.8.2 Point Table Data, Row Format

The data for a point is obtained in a rectangular format, similar to a row in a spreadsheet program. A regular format allows faster downloading and easier conversion to and from spreadsheets and editor programs.

As returned by the CLC, each of the 12 elements of a point are separated by an ASCII space character, and the entire point table entry is terminated by a newline. Any number of spaces between elements is allowed when uploading to the CLC.

Examples:

The Host requests number of absolute points in program 2.

>1 XL 2.0 \r\n
The CLC responds with 10 points.

>1 XL 2.0 10 10 10 10 10 10 10 10 10 10 10 10\r\n  _ number of points

The Host requests point ABS[1]'s data for the active program:

>1 XL 0.1 \r\n
The CLC Responds:

>1 XL 0.1 1.0 2.0 0.0 0.25 50 10 33 50 1 0 0 0 0.0 0.0 0.0 0
;where X Y Z B S A D J 1 2 3 4 R P W E

The Host sends point REL[2]'s data for program 3:

>1 YL 3.2 1.0 2.0 0.0 0.25 100 60 33 50 1 10 0 0 $cs\r\n
The CLC Acknowledges:
>1 YL 3.2 $cs\r\n
B.8.3 List of All User-defined Labels (V)

Label: integer followed by string of \( \leq 21 \) byte
Set: program handle
Number: index for list
Step: not used

For all classes that include user-defined labels, this subclass lists the labels. To start the list and get the total number of labels, request data with Step = 0. The labels are listed in the format ‘Label_num Label_string’ from the serial port. The type ‘Label’ is used from the executive interface. Label_num is the index in the corresponding table; for example ‘1’ for ABS[1]. Label_string is a string of 21 bytes maximum, including null termination.

B.9 Event Tables

Each program stored on the CLC has an associated event table. Each element of the Event Table may be accessed individually or as a list. The available elements for the Event Table are shown in the list of Command Classes and Subclasses, and described in the Programming Elements section.

General Format:

```
>1 ES h.nn\r\n    |   |   _ number: Event number
    |   |   _ set: Program handle
    |   | _ subclass: Status request
    _ class: Event Table
```

The user program handle specifies which user program is accessed. To request events for the active program, use a "0" as the handle.

Event data is accessed by using an event number as an index into the event table (E.G. 1 for EVT[1]). To request the total number of entries in the event table, use a "0" for the event index number.
B.9.1 Event Table Data

Most data in the event table has associated codes corresponding to events in the teach pendant display.

<table>
<thead>
<tr>
<th>Subclass</th>
<th>Selections</th>
<th>Type</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>S Status</td>
<td>0 = inactive 1 = queued 2 = pending 3 = executing 4 = done</td>
<td>Unsigned Integer</td>
<td>Read-only</td>
</tr>
<tr>
<td>T Type</td>
<td>0 = undefined 1 = timer based 2 = coord. time 3 = coord. % distance 4 = single-axis distance 5 = repeating axis position 6 = task external interrupt input 7 = VME broadcast interrupt 8 = VME short address interrupt 9 = axis feedback capture</td>
<td>Unsigned Integer</td>
<td>Read/write</td>
</tr>
<tr>
<td>D Direction</td>
<td>0 = start of move 1 = end of move</td>
<td>Unsigned Integer</td>
<td>Read/write</td>
</tr>
<tr>
<td>A Argument</td>
<td>A numeric value (in milliseconds if timed event) (a % of the segment distance if coordinated motion) (degrees if repeating axis position event) (contains the probe position read from the drive if feedback capture event is selected)</td>
<td>Floating</td>
<td>Read/write</td>
</tr>
<tr>
<td>F Function</td>
<td>Valid event function mark</td>
<td>String</td>
<td>Read/write</td>
</tr>
<tr>
<td>M Message</td>
<td>text from 0-80 characters</td>
<td>String</td>
<td>Read/write</td>
</tr>
</tbody>
</table>

Examples:

```plaintext
>1 ES 2.0 \r\n ;Host requests number of events in program 2.

>1 ES 2.0 50\r\n ;50 events
>1 ES 1.20 \r\n ;Host req. EVT[20].s for program 1

>1 ES 1.20 1 $cs\r\n ;CLC responds with ACTIVE
```
>1 ET 0.10 3 $cs\r\n ; Host selects active EVT[10] as ; distance

>1 ET 0.10 $cs\r\n ; CLC acknowledges

>1 EF 0.1 Gripper_On $cs\r\n ; Host selects event function

>1 EF 0.1 !32 Label not found $cs\r\n ; Label was not found in ; program

### B.9.2 Event Table Data, Row Format

The data for each event is transferred between the CLC and Host in a regular format that permits easy exchange of data with spreadsheet and text editing programs. Each element of an event is separated by a single space when sent from the CLC. When sending data to the CLC any number of spaces may be used between the elements of an event. The CLC will not return an error if an event’s status element is sent; the element will be ignored.

**Examples:**

The Host requests EVT[1]’s data for program 5:

>1 EL 5.1 \r\n
The CLC responds with:

>1 EL 5.1 1 2 0 90.0 In_Zone In the zone! $cs\r\n       STD A F M

The Host sends data for program 2, EVT[19]:

>1 EL 2.19 0 1 1 1000 Calc_Zone Calculating... $cs\r\n
B.10 CLC Program Communication

Each user program (text or icon) that is developed and successfully compiled on a PC Host results in a downloadable base-code CLC executable file. These executable files may be exchanged between the Host and CLC using the upload/download method described in this section. Programs resident on the CLC can be activated and deleted; and lists of variables and subroutines associated with the program may be accessed using the serial communication interface. The number of programs that may be downloaded to a CLC is limited by the size of the programs and the amount of available non-volatile memory.

B.10.1 Program Header Record

Each user program contains a header record that identifies the program and is used for starting program uploads and downloads. The header contains the program name, size, checksum, and date; and is stored with the program in the CLC’s non-volatile memory.

The size is a decimal count of all bytes in the program file. The checksum is a 4 byte hexadecimal two’s complement sum of all bytes in the file (excluding the checksum). The date is in Month/Day/Year format (E.G., 07/04/93). The program name can include any ASCII characters and may be of any length. The detailed CLC program file format is described in an appendix.

B.10.2 Download Block Size

CLC program file uploads and downloads are performed as a sequence of fixed-length blocks. The block size is selected using Parameter C1.50 (Download Block Size) and defaults to 16 bytes. The Download Block Size parameter may be changed to 1 to 512 bytes by the Host computer to optimize transmission time or for terminal compatibility.

B.10.3 Initializing a Download (PW)

To start a download from the Host to the CLC, the Host sends a Header Record with the "PW" command. If the available CLC memory is insufficient to contain the program, the CLC returns an error message. If enough memory is available, the CLC responds with a numeric program handle. The program handle returned by the CLC must be used to identify the program with subsequent commands. The CLC will always respond with a new handle, allowing more than one program with the same name.
Example:

The Host starts a download:

```
<----------Header Record--------->
>1 PW 0.0 1592 ABCD1234 12/25/92 Program_1 $cs\r\n    |    |        |        | Program Name (variable length)
    |    |        | Date (mm/dd/yy)
    |    | Program checksum (hex)
    | Program size in bytes (decimal)
--- Command: Start Download
```

The CLC responds with a valid program handle:

```
>1 PW 0.0 1 $cs\r\n--- Program handle returned by CLC
```

**B.10.4 Executing a Download (PD)**

After the Host has sent the program header record and obtained a program handle, the Host must send the entire program to the CLC, before the program may be activated. The Host uses a sequential number to identify each block. Block by block transfer permits other communication with the CLC to take place between blocks.

If a communication error is received when sending a program block, the current block may be repeated. The CLC download program requires program block sequence numbers to be incremented by one, or an error message will be returned.

After all program blocks have been downloaded, the Host completes the download by sending a final block with the data area containing at least one non-space character. The final block is identified by a sequence number equal to the last program block sequence number + 1. The checksum is verified and the CLC responds with an acknowledgment or an error message.

**NOTE:** *The PD command must not be executed without a preceding PW or PR command to initialize the transfer.*
Example:

The Host sends a block (using the default block size = 16 bytes):

```
>1 PD 1.1 0123456789ABCDEF0123456789ABCDEF$cs\r\n   _ 32 hex digits of data (16 bytes)
   _ Sequence number (from 1 to (size/block size))
   _ Program handle previously obtained by PW or PR command
   _ Command: Download
```

The CLC responds during download (no error):

```
>1 PD 1.1 $cs\r\n```

The Host sends the final block after 100 program blocks have been sent:

```
>1 PD 1.101 0 $cs\r\n   _ dummy block not stored by CLC
   _ (size of program/block size) +1
```

The CLC responds after verifying the checksum:

```
>1 PD 1.101 !19 List is finished $cs\r\n```

B.10.5 Initializing an Upload (PR)

To request an upload from the CLC, the Host sends a program handle with the "PR" command. If the program handle does not match a program in the CLC, the CLC responds with an error. The CLC responds with the corresponding program header. (The "PH" command can be used to obtain a list all CLC resident programs and handles.)

The Host requests an upload:

```
>1 PR 0.0 1 $cs\r\n   _ Program handle
   _ Command: Start Upload
```

The CLC responds with the program's header:

```
>1 PR 0.0 1 1592 ABCD1234 12/25/92 Program_1 $cs\r\n   _ Program handle
```
B.10.6 Executing an Upload (PD)

The PD command is also used during uploads, but the Host sends only the sequence number. The CLC responds with the data stored in each block. The process repeats sequentially, block by block, in the same manner as the program download (see Executing a Download, above).

The Host requests a block:

```
>1 PD 1.1 $cs\r\n   |   |_ program block sequence number
   |   |_ program handle
   |  Command: upload
```

The CLC responds with data for the requested program and block:

```
>1 PD 1.1 0123456789ABCDEF0123456789ABCDEF$cs\r\n```

The Host sends the dummy block request, after the last program block has been received:

```
>1 PD 1.101 $cs\r\n   |_ (size of program/block size) +1
```

The CLC responds after checksum verification:

```
>1 PD 1.101 !19 List is finished$cs\r\n```

B.10.7 Erasing (Deleting) a Program (PE)

This command erases the program identified with the specified program handle. The CLC memory previously required by the program becomes available to other programs; and the header and handle are removed from the CLC's list of resident programs.

Notes:

1) A currently active program may not be erased, unless the CLC has been placed in Parameter Mode.

2) The Erase Program command format specifies "0" in the positions normally occupied by the program handle and block sequence number, followed by the numeric program handle of the program to delete.
The Host sends an Erase Program Command:

```
>1 PE 0.0 1 $cs\r\n   |_ Erase Program with handle #1
```

The CLC acknowledges with:

```
>1 PE 0.0 $cs\r\n```

**B.10.8 List CLC Resident Programs (PH)**

The "PH" command provides a list of the programs resident on the CLC. The list contains the headers of all programs and their corresponding program handles. An upload sequence of commands are used to obtain the list.

The Host requests the start of the program list:

```
>1 PH 0.0 $cs\r\n   |   | sequence 0= start of list
   |   | Command: list program headers
```

The CLC responds with the number of programs on the card:

```
>1 PH 0.0 4 $cs\r\n   |_ number of resident programs
```

Host Requests a file record from the list:

```
>1 PH 0.1 $cs\r\n   |_ Sequence number for record 1
```

**CLC Responds:**

```
>1 PH 0.1 1 1592 ABCD1234 12-25-1992 Program_1 $cs\r\n   |   | <-------- Program header -------->
   |   | Program handle for following program header
   |   | Sequence number for program header list
```
B.10.9 Activating a Program (PA)

The Host can activate a program by sending its program handle with the "PA" command. If the handle or the program is invalid, the CLC returns an error message.

NOTE: A program can not be activated by the Host if the CLC is currently running any user tasks.

The Host sends an activate program 2 command:

```
>1 PA 0.0 2$cs\r\n
 _ Activate program with handle = 2
```

The CLC acknowledges with:

```
>1 PA 0.0 $cs\r
```

B.10.10 Request Currently Active Program (PA)

If the CLC send a "PA" command without specifying a program, the CLC returns the handle of the currently active program. Other information about the program is available through task parameter requests.

The Host sends:

```
>1 PA 0.0 \r\n
```

The CLC returns:

```
>1 PA 0.0 2 $cs\r\n
 _ Program 2 is active
```

B.10.11 Request Name of Program (PN)

The "PN" command may be used to obtain the ASCII text name of a specified program handle from the CLC. If a handle of 0 is requested, the name of the currently active program is sent. If an invalid handle is requested, the CLC returns an error message.

The Host requests the active program:

```
>1 PN 0.0 \r\n
 _ Request active program's name
```
The CLC responds with:

>1 PN 0.0 prog_one\r\n
The Host requests the name of program 3:

>1 PN 3.0 \r\n     |__ Request program 3's name

The CLC returns:

>1 PN 3.0 prog_3\r\n
B.10.12 List of Event Function Marks (PF)

Text marks for CLC resident event functions can be listed by using a list sequence. Subroutine marks are listed in alphabetical order, as stored in the CLC program file. The program handle is used to identify which program's events are required.

Request Format:

>1 PF h.s
     |  |__ sequence number
     |__ program handle

Response Format:

>1 PF h.s Sub_Mark
         |__ subroutine mark (or function name)

Example:

The Host requests the function mark list for the active program:

>1 PF 0.0 \r\n
The CLC responds with the count of marks for the active program:

>1 PF 0.0 2 $cs\r\n         |__ 2 marks are defined

The Host requests the first mark:

>1 PF 0.1 \r\n
The CLC returns:

>1 PF 0.1 Close_Gripper $cs\n
The Host requests the next mark (2):

>1 PF 0.2 \n
The CLC responds:

>1 PF 0.2 Open_Gripper $cs\n
The Host closes the list:

>1 PF 0.3 \n
| _ next sequence number after count of defined marks

The CLC acknowledges and closes the list:

>1 PF 0.3 !19 List is finished $cs\n
B.10.13 List Variable Labels (PV)

Text labels for variables are stored with each program resident in the CLC, and can be listed using a list sequence. The variable numbers and text names are listed in alphabetical order by text name. The variable number consists of an "I" or "F" ASCII character, followed by a 1-to-3 digit number. The variable number and its label are separated with a space.

Request Format:

>1 PV h.s
   | _ sequence number
   | _ program handle

Response Format:

>1 PV h.s Ixx Label
   | _ variable label
   | _ variable id number (index number)
   | _ variable type identifier (F = floating point, I = integer)
Example:

The Host requests the variables label list for program 1:

>1 PV 1.0 \r\n
The CLC responds with the count of named variables for program 1:

>1 PV 1.0 2 $cs\r\n
|__ 2 labels are defined

Host requests the first label:

>1 PV 1.1 \r\n
The CLC responds with:

>1 PV 1.1 F11 axis1_velocity $cs\r\n
The Host requests the next label

>1 PV 1.2 \r\n
The CLC responds with:

>1 PV 1.2 I1 part_count $cs\r\n
Host closes the list:

>1 PV 1.3 \r\n
The CLC closes the list and acknowledges:

>1 PV 1.3 !19 List is finished $cs\r\n
B.10.14 Transfer Tables Between Programs (PX)

The CLC stores a unique set of tables for each user program. When a new program is downloaded, the new programs tables are set to default or new values. The "PX" command allows the Host to copy a working program's tables to another program. The ability to copy a set of working, tested tables to another program under test can be exceedingly useful when debugging a program, or entering/modifying points using the teach pendant.

The transfer command copies all the tables associated with the CLC resident source program to
the resident destination program. The tables include: the float and integer variables, the absolute and relative point tables, and the event table.

Tables are copied on a one-for-one, table-to-table basis; with the source table overwriting the destination table. If the source and destination tables are of different sizes, the table will be copied so that the destination table size does not increase. If the source table size is less than the destination, the remainder of the destination table will keep the same values.

The event functions in the source program are checked against those that exist in the destination program and new function offsets are stored. If an event function does not exist in the destination program it is set to "NONE."

Format:

```plaintext
>1 PX s.0 d \r\n   |   |__ program handle of destination program
   |__ program handle of source program
```

Example:

The Host requests that program 3 load a copy of program 1's tables:

```plaintext
>1 PX 1.0 3 \r\n```

The CLC responds with:

```plaintext
>1 PX 1.0 \r\n```

Example:

The Host requests that program 1 load a copy of the active program's tables:

```plaintext
>1 PX 0.0 1 \r\n```

The CLC responds with:

```plaintext
>1 PX 0.0 \r\n```
B.10.15 Sequencer/Subroutine Related Subclasses

Subclasses:
L  Max number of rows in sequence list    integer, read only
Q  Max number of rows in sequence table  integer, read only
S  Number of function slots available    integer, read only
I  Data change identification            integer, read only

Set:          program handle
Number:       for S subclass: 1=total number, 2=current available
              for Q, L subclasses, always set to 0
              for I subclass, 1=sequencer system
Step:         not used
Exec functions:  get_Program(), put_Program()

These classes display the limits on sequence lists and tables that were set up at compile time.

The L subclass displays the maximum number of rows allowed in a sequence list.

The Q subclass displays the maximum number of rows allowed in a sequence table. This
number is internal to the CLC and is set to 100 in the current version.

The S subclass displays information about the memory used for sequence tables. The functions
entered into a sequence table are stored in memory as an array of function slots that is used by
all sequences, which optimizes memory usage and editing. If ‘number’ is set to 1, the total
number of function slots is returned. If ‘number’ is set to 2, the number of unused slots for
functions remaining is returned.

The I subclass provides a way for a user interface to detect if data has been changed by another
port or the user program. Each time data is stored, a counter is incremented. To check the
sequencer system, set ‘number’ to 1. Other numbers are reserved for future functions.
B.11 Function Class (S)

Subclasses:
- **T** Function name
- **A** Access type
- **N** Argument label
- **Y** Type of data
- **L** Minimum value
- **H** Maximum value
- **R** Argument attributes in row format
- **V** Local variable attributes in row format

**Set:** program handle
**Number:** function number (0 = total number of functions)
**Step:** argument number 1-5 (0= number of args for this function)
**Exec functions:** get Function(), put Function()

All subroutines, tasks, and functions in a program have attributes that can be read using this class. These are used for displaying names and validating data. All attributes are set in the user program and cannot be changed on-line. The function number is used in all references to this function in the sequence table.

The **T** subclass returns the function name as a null-terminated string of 20 characters or less.
The **A** subclass displays (2) if the function can be used in a sequence table, (1) if it is hidden from the user.

The **N** subclass displays an argument name as a null-terminated string of 20 characters or less.
The data type **Y** is an ASCII character corresponding to the data class. This class can be used to access label lists for points and registers.

The minimum and maximum values are returned in the **L** and **H** subclasses. The user interface can read these values to determine if data entry is valid.

The **R** subclass provides the serial port interface with subclasses **N, Y, L, and H** for function arguments in a space delimited string.
For example: “>0 SR 0.1.1 test_var F 0.0 10000.0 $cs\r\n”.

The **V** subclass provides the same information as the **R** subclass, but with local variables. Since there are no argument ranges for local variables, zeros are printed in the last two fields. The number of local variables in a task is obtained with the command “>n SV h.f.0”, where n=unit, h=handle, and f=function.
B.12 Input/Output Registers

The Host system may read the CLC's input and output registers at any time; including control, status, and programmable registers. The CLC's axis, system, and task status registers are normally read-only, and are only changed by the CLC I/O mapper executive task or the register forcing commands (see below). Setting I/O registers directly (using the RB, RX and RD commands) has the lowest priority of all I/O access methods.

Directly accessing I/O registers should be done with caution. The CLC is a multitasking system, and as such the potential for I/O contention always exists between user tasks, the Host communication, the I/O Mapper, and the I/O subsystem. It is the programmer's responsibility to anticipate contention problems and synchronize accesses to data between asynchronous CLC tasks when necessary.

The forcing commands (RM, RF, RC and RS) are provided primarily for debugging purposes. Forcing commands should be used with extreme caution since they can be used to override the state of system control registers, and have higher priority than the CLC's I/O mapper or Host direct access commands.

The requirement for a checksum may be disabled by parameter. This practice is not suggested. It results in no communication error checking of data sent to the CLC that may effect safe operation of the system.

B.12.1 I/O Register Access (RB), (RX), (RD)

Input registers are accessed using "R(data type)" commands and a register specifying index number within the range 1 to 200. The current contents of the register may be read as a 16-bit binary number (command "RB"), a 4-digit hex number (command "RX"), or a decimal integer number (command "RD").

Example:

```
>1 Rt 0.nnn $cs\r\n
| | register number
| set ID, always 0 for I/O registers
| subclass: type or format (B=binary, D=decimal, X=hexadecimal)
```
I/O Register Read

Example:

Host requests the contents of register 1 in binary:

>1 RB 0.1 \r\n
The CLC responds with:

>1 RB 0.1 0001001000110010 $cs\r\n    |    | least significant bit
    |    | most significant bit

The checksum is optional when reading data from the CLC.

Sending a "0" as the register index number returns the number of registers in the current system.

I/O Register Write

The Host may send a value to a CLC I/O register in hexadecimal ("RX"), binary ("RB"), or decimal ("RD") using the same format as an I/O read with the addition of a data field and checksum.

Example:

>1 RX 0.121 0x0040 $cs\r\n    | | 16 bit hex word to write
    | | I/O register number 121
    | always 0 for I/O registers
    | read/write to register in hex

B.12.2 Set Current I/O State with Mask (RM)

The RB, RD and RX commands affect every bit of the destination I/O register, the new data word replaces the old word. RM allows you to specify a mask in addition to data bits, limiting the I/O register bits that are changed.

The most significant 16 bits in this 32-bit word provide the mask selecting the bits that may be changed. A "1" enables change, "0" masks any change. The least significant 16 bits changes the state of the I/O register bits. If RM is used to read the register, the CLC returns the state of all bits. See the notes above for changing I/O register bits.
Format:

```plaintext
>1 RM 0.2 0x00600040
   |   |   |   16 bit word of new bit states
   |___|   16 bit mask of bits to change
```

Example:

```plaintext
>1 RM 0.2 0x00600040 ;bit 6 is turned on, bit 7 off
```

RM is a single use, independent equivalent of setting a mask with an RF command, then setting the actual I/O bit states with an RC or RS command. Since RM contains its own mask it does not affect the forcing mask set with RF. See the RF, RC and RS commands below.

**B.12.3 I/O Forcing Selection (RF)**

The forcing selection (RF) and forcing state (RC and RS) commands allow the Host to selectively force the state of individual bits in the I/O registers. Forcing commands take priority over the CLC I/O mapper and I/O devices.

The forcing remains in effect until the mask for each forced bit is cleared, or until there is a timeout error on the serial port. When the forcing state changes for bits in a CLC control register, all edge detection is reset.

If a bit in the 16-bit forcing mask is set to 0, the corresponding bit in the I/O register is controlled by the I/O Mapper and physical I/O. If the forcing mask bit is set to 1, the I/O register bit is forced by the Host "RC" or "RS" command.

The data format of the "RF" 16-bit forcing mask word is always binary.

Example:

```plaintext
>1 RF 0.2 00000000001001000
   |___|   bits 4 and 7 are forced bits and are
controlled by the Host all other bits
are controlled by the physical I/O and
the CLC I/O Mapper
```

**B.12.4 I/O Forcing State Change (RC)**

The most significant 16 bits in this 32-bit word select which bits in the I/O register may be affected, and the least significant 16 bits changes the states of those bits. If it is read, it returns the state of all bits.
The data format of the "RC" state change word is always a 32-bit hexadecimal longword.

Format:

>1 RC 0.2 0x00600040
   |   |___| _ 16 bit word of new bit states
   |___| _ 16 bit mask of bits to change

Example:

>1 RC 0.2 0x00600040
   |   _ bit 6 on, bit 7 off
   | _ allow changes to bits 6 and 7

B.12.5 I/O Binary Forcing State (RS)

The "RS" command is used to read and write the state of the forcing bits for the selected register. If bits are to be affected, the desired bits in the I/O register must have had forcing enabled by a forcing mask set with the "RF" command.

The data format of the "RS" 16-bit forcing state word is always binary.

Example:

>1 RS 0.2 0000000100000001
   |_________| _ bits 1 and 9 are turned on, all other
   |         | _ bits turned off if the bits have
   |         | _ forcing enabled by an "RF" command

B.12.6 Erase All Forcing Masks (RE)

This command sets all forcing masks and states to zero and returns the I/O system to normal control. The command only takes effect at the time that it is sent.

Caution should be used when using this command. The I/O registers are directly affected and clearing the mask(s) may cause immediate unwanted motion in the system.

The data format of the "RE" command is ASCII integer.

Example:

>1 RE 0.1 1
   |   _ set to 1 to erase forcing masks
   | _ always '0.1'
B.13 Sequencer Data

The CLC serial protocol includes the following classes and subclasses to handle the sequencer and functions with arguments.

B.13.1 Sequence List Class (L)

<table>
<thead>
<tr>
<th>Subclasses:</th>
<th>T</th>
<th>Name of Sequence List</th>
<th>string, (&lt; = 21) byte (read/write)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>P</td>
<td>Print/Store Sequence Tables (rows of list)</td>
<td>integer sequence table number</td>
</tr>
</tbody>
</table>

Set: program handle
Number: Sequence List Number (0 returns total number of lists)
Step: sequence table row in this list (0 returns number of rows)
Exec functions: get_SequenceList(), put_SequenceList()

The T subclass reads or writes the sequence list name as a null-terminated string of 20 characters or less.

The P subclass prints or stores a sequence list as a variable-length list. The tables in the list are identified by the ‘number’ in the ‘Q’ data class.

To store a new sequence list, the host sets the ‘step’ to 0 and sends the size of the list. Setting the size of the list to 0 erases all entries in the list.

To change an existing row in the sequence list, ‘step’ must be set to the row number.

If a sequence list is edited while it is running, the communication error ‘Cannot edit sequence table while running’ is issued.

Sending a new sequence list:

\[
\begin{align*}
> 0 \ & \text{LP} \ 0.1.0 \ 5 & \text{;Host tells CLC to store list 1 with 5 function tables} \\
> 0 \ & \text{LP} \ 0.1.0 & \text{;CLC acknowledges} \\
> 0 \ & \text{LP} \ 0.1.1 \ 2 & \text{;Host selects Table number 2 as first table} \\
> 0 \ & \text{LP} \ 0.1.1 & \text{;CLC acknowledges} \\
> 0 \ & \text{LP} \ 0.1.2 \ 1 & \text{;Host selects Table number 1 as second table} \\
> 0 \ & \text{LP} \ 0.1.2 & \text{;CLC acknowledges} \\
> 0 \ & \text{LP} \ 0.1.6 \ 0 & \text{;Host closes the list} \\
> 0 \ & \text{LP} \ 0.1.6 \ !1\!9 & \text{List is finished; CLC acknowledges, table now ready to execute}
\end{align*}
\]
Changing one row of sequence list:
>0 LP 0.1.2 2 ;Host changes second row in list to Table number 1
>0 LP 0.1.2 ;CLC acknowledges

B.13.2 Sequence Table Class (Q)

<table>
<thead>
<tr>
<th>Subclasses</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>Table Name</td>
</tr>
<tr>
<td>P</td>
<td>Print/Store function numbers (rows of table)</td>
</tr>
<tr>
<td>F</td>
<td>Function numbers</td>
</tr>
<tr>
<td>1</td>
<td>Argument 1</td>
</tr>
<tr>
<td>2</td>
<td>Argument 2</td>
</tr>
<tr>
<td>3</td>
<td>Argument 3</td>
</tr>
<tr>
<td>4</td>
<td>Argument 4</td>
</tr>
<tr>
<td>5</td>
<td>Argument 5</td>
</tr>
</tbody>
</table>

Set: program handle
Number: table number (0 returns total number of tables)
Step: function row in this table (0 returns number of rows in this table)
Exec functions: get_SequenceTable(), put_SequenceTable()

The T subclass reads and writes the sequence table name as a null-terminated string of 20 characters or less.

The P subclass prints or stores a sequence table as a variable-length list. The functions in the list are identified by the ‘number’ in the ‘S’ data class.

To store a new sequence table, the host sets the ‘step’ to 0 and sends the size of the table. Setting the size of the table to 0 erases all entries in the list.

To change an existing row in the sequence table, ‘step’ must be set to the row number.

Through the serial protocol the data is a space-delimited string in the format (function number, arg1, .. argn). Arguments are printed as floating point values. Arguments that don’t exist are printed as ‘0’. The CLC will ignore extraneous arguments when data is sent to it.

From the executive interface, the data is type Special with the structure FUNC_DATA_t. All elements of a function are passed and returned using this structure (see end of this document).

The function name can be requested with the “ST” protocol command. Argument names, types, and limits can be requested with the ‘S’ data class.
If a sequence table is edited while it is running, the communication error ‘Cannot edit sequence table while running’ is issued.

Subclasses F, 1, 2, 3, 4, and 5 can be used to individually access elements of each row of the table. The individual arguments can be changed while the sequence table is running.

Getting a sequence table from the CLC:
(refer to example for sequence list for sending data)

>0 QP 0.2.0 ;Host asks for number of functions in table 2
>0 QP 0.2.0 3 ;CLC responds
>0 QP 0.2.1 ;Host asks for first function
>0 QP 0.2.1 2 12.34 100 25 0 0 ;function number 2
>0 QP 0.2.2 ;Host asks for second function
>0 QP 0.2.2 5 100 0 0 0 0 ;function number 5
>0 QP 0.2.3 ;Host asks for second function
>0 QP 0.2.3 1 12.5 30.2 0 0 0 ;function number 1
>0 QP 0.2.4 ;Host closes the list
>0 QP 0.2.4 !19 List is finished ;CLC acknowledges
B.14 Communication Error Codes and Messages

!01 SERCOS Error Code#xxxx (xxxx=Error code)
This is the code set in the data status word of the DDS-2 drive if SERCOS communication is invalid. Call Indramat Service if this error occurs.

!02 Invalid Parameter Number
The requested or sent parameter does not exist on the CLC or the drive, or the format of the parameter is incorrect.

!03 Data is Read Only
The data in this parameter may not be modified.

!04 Write Protected in this mode/phase
The data in this parameter can not be written in this mode or communication phase. Switch into parameter mode (phase 2) to enter the parameter.

!05 Greater than maximum value
The parameter exceeds the maximum allowed value.

!06 Less than minimum value
The parameter is less than the minimum allowed value.

!07 Data is Invalid
Parameter data is invalid, or the format of the parameter is invalid. See the DDS or CLC Parameter Descriptions.

!08 Drive was not found
The requested drive was not found on the SERCOS ring.

!09 Drive not ready for communication
The requested drive or the SERCOS ring has not been initialized.

!10 Drive is not responding
The drive did not respond to a service channel request. Check system diagnostics for the state of the SERCOS ring.
!11 Service channel is not open.
When switching between initialization phases, data from the drive is momentarily invalid, and this message is sent instead of the requested data.

!12 Invalid Command Class
A serial port command is invalid or not supported at this time.

!13 Checksum Error: xx (xx = checksum that CLC calculated)
The CLC detected an invalid or missing checksum in data that was sent to it. As a debugging aid, the checksum that the CLC calculated on the incoming data is also sent with this message.

!14 Invalid Command Subclass
A serial port command option is invalid or not supported.

!15 Invalid Parameter Set
The parameter set number (task or axis) is invalid.

!16 List already in progress
An attempt has been made to start a parameter or program list that is already in progress.

!17 Invalid Sequence Number
The sequence number of a parameter or program list is invalid or has been sent out of order.

!18 List has not started
A parameter or program list has not been initiated (i.e. sequence number was sent before list was started).

!19 List is finished
This is an acknowledgment that a parameter or program list is complete. It does not indicate an error.

!20 Parameter is a List
This parameter is a variable-length list, and its data cannot be displayed as a normal parameter.

!21 Parameter is not a List
Only Variable-Length List parameters can use the Parameter List sequence.

!22 Invalid Variable Number
The variable mnemonic was not 'T' or 'F', or the variable number is greater than the maximum number of variables allocated.
!23 Insufficient program space
This message is sent after the CLC receives a "PW" program header if not enough contiguous
memory is left on the CLC to store the program. Other programs may need to be deleted or their
order rearranged. Check system parameters C1.91, C1.92 and C1.93 for CLC memory status.

!24 Maximum number of files exceeded
The CLC allows up to 10 programs resident in the CLC. This error message is sent when the CLC
receives a "PW" program header and there are already 10 programs stored on the CLC. One of the
CLC resident program files must be deleted to make room to download the program.

!25 Invalid program header
The format of the program header sent to the CLC is invalid, or this command is not available for
reading or writing.

!26 Checksum Error in Program
This message is sent at the end of a download if the checksum of the data does not match the
checksums sent in the program or program header.

!27 Invalid Program Handle
The format of the handle is incorrect, or this command is not available for reading or writing.

!28 Function not Implemented
The function is not implemented in this version of the CLC.

!29 File not Found
A program corresponding to the requested program handle was not found (e.g., the program is not
resident in the CLC).

!30 Invalid I/O Register
The I/O register mnemonic is invalid or a register number greater than the maximum number of
registers was sent.

!31 Invalid Table Index
The ABS, REL, or EVT table name was incorrect, or the index number was greater than the
maximum number of points or events.

!32 Communication Error 32
This error is not used by the CLC at this time.

!33 Invalid Data Format
The format of the data received by the CLC is invalid (e.g., non-digits are sent in a decimal
number).
!34 Active program can't be deleted
The active program cannot be deleted at any time.

!35 Parameter mode is required
The action requested can only be performed in Parameter Mode.

!36 Invalid Event Number
The event number selected in the ABS or REL point table is out of the range of the total number of events.

!37 Invalid Event Function
The function name selected in the event table does not exist on the CLC card or is not defined as an event function.

!38 Program file version mismatch
The version of the file system on the card does not match that of the downloaded file. Upgrade to the latest versions of the Visual Motion compiler and CLC executive.

!39 Can't activate while program running
A new program cannot be activated unless all user tasks are stopped.

!40 No programs are active
No programs are active on the CLC card. Download a program to the card.

!41 System Error: pSOS #XXXX
This is an internal CLC system error. Call Indramat Service for assistance.

!42 Mapper String DD: invalid operator
An invalid boolean operator was found in I/O Mapper String "DD" when it was sent to the CLC.

!43 Mapper String DD: too many operations
Sending the string "DD" exceeded the maximum number of boolean operations allowed by the CLC I/O mapper.

!44 Mapper String DD: invalid register
A register in Mapper String "DD" exceeds the maximum number of registers or is 0.

!45 Mapper String DD: invalid bit or mask
The bit number or mask sent in string "DD" exceeds 16 bits.
!46 Mapper String DD: register is read-only
An assignment to a read-only register or bit was made in I/O mapper string "DD" (e.g., attempting to write to a CLC status register).

!47 Invalid Unit Number
The unit number (second character in string) is not a number between '1' and 'F' or an ASCII space character.

!48 VME Bus Error
A VME bus error occurred while communicating to another card in pass-through mode through the serial port.

!49 VME Communication Handshake Error (D)
The card addressed by the unit number in pass-through mode does not exist or its parameters are not configured properly. Change the unit number to correspond to a card in the rack or set it to a space. (No longer issued on CLC-D.)

!50 Invalid Download Block
The block sent during a program download is incorrect in length or is not in hexadecimal format.

!51 Unit D: Invalid VME Base Address Page
The VME base address page parameter is set to an invalid address for the indicated VME unit number.

!52 Axis Disabled
The parameter set for the requested axis does not exist. Either this axis is disabled or the CLC does not support this number of axes.

!53 Waiting for service channel
When switching between drive initialization phases, data from the drive is momentarily invalid. This message is sent instead of the requested data. This message will also be issued whenever a service channel transaction cannot be completed. Continue to retry the message until a valid response is returned.

!54 List or String is too short
The text string or parameter list is smaller than the minimum length allowed by the CLC or the drive, or the size of a value does not match the attributes sent from the drive.

!55 List or String is too long
The text string or parameter list exceeds the maximum length allowed by the CLC or the drive, or the size of a value does not match the attributes sent from the drive.
!56 PC Communication Handshake Error
The CLC/P is not responding to an ASCII message. Check the address configuration on both the PC (config.sys and system.ini) and the CLC/P (address jumper switches).

!57 Mapper String D: string space is full
The CLC’s memory that was allocated for I-O mapper strings (8KBytes) has been exhausted. Optimize the mapping program so that it fits into memory.

!58 Cannot store cam: already active for axis D
Cam data cannot be changed unless no axes are currently using it. Deactivate the cam for axis ‘D’, then send the cam again.

!59 SERCOS handshake/busy timeout
This is an internal error generated by the SERCOS ASIC. Change modes or reset the card. If it happens again, call Indramat Service.

!60 Executable program is too large (ddK)
The executable portion of the user program downloaded to the CLC exceeds the maximum limit, which is indicated in the message (‘dd’) in kilobytes. Optimize the program and download it again, or update the firmware to a version that has a larger program limit.

!61 System Memory Allocation Error
The dynamic memory space on the CLC has been exhausted. Call Indramat Service for assistance.

!62 Cam X data is < 0 or greater than 360
All values in the x-column (right hand column) of the cam file sent to the CLC must be between zero and the modulo value of the master.

!63 X-Column does not start at 0 or end at 360
In the cam file sent to the CLC, the first point must be zero and the last point must be the modulo value of the master. Check the beginning and end of the cam file.

!64 Not supported in user prog file version 1.1
The requested feature is not present in the file version of the user program from which the data was requested or sent. To use this feature, a compiler upgrade is necessary.

!65 Sequencer: invalid sequence (D)
The sequence number (D) is zero or is greater than the allocated maximum number of sequencers for this program.
!66 Sequencer: invalid step (D)
The sequencer step number (D) is zero or is greater than the allocated maximum number of steps for this program.

!67 Invalid function number (D)
The function number (D) selected for a sequencer step is invalid or refers to a function that does not exist on the card.

!68 Function D not accessible in a step
The function referred to with the number (D) cannot be entered in a sequencer step. It needs to be declared accessible by the sequencer in the user program.

!69 Too many functions are used (D)
The total number of functions used by all steps exceeds the number (D) allocated for the program in the data sizing instruction, or the number of functions used in a step exceeds the number of functions remaining. Reduce the number of functions used or allocate more function slots in the data sizing instruction.

!70 Maximum steps per sequence exceeded (D)
The number of steps in a sequence exceeds the number (D) allocated for the program in the data sizing instruction.

!71 Maximum functions per step exceeded (D)
Up to (D) functions can be used in one sequencer step. This is a CLC system limit, which in version GPS-02.00 is 100.

!72 Program does not include a PLS
PLS data was requested from a program that does not support the Programmable Limit Switch function or does not have any PLS’s configured.

!73 Invalid ABS or REL point index (D)
Point D is zero or is greater than the allocated maximum number of points for the selected point table

!74 Error in command execution
A procedure command set in the CLC or drive parameter has not been successfully completed.

!75 Comm. port buffer overflow
The serial port receive buffer has overflowed. In current versions of the CLC, this buffer is 512 bytes. To avoid this error, the host must communicate in half duplex or use XON-XOFF handshaking correctly.
!76 Invalid Block
This message is reserved for the TRANS01-D version of the CLC. See the documentation for this version.

!77 Can't save sequencer while it is running
Sequencer data can only be save while the program is not running, or while no user tasks are running a sequencer.

!78 Service channel in use
The SERCOS service channel is being used by a user program task or by a CLC internal process, and has suspended the transmission of a list or text string. See the description of parameter C-0-0010, bit 12.

!79 PID block number does not exist
This error is issued when the selected PID block is not initialized in the user program.
C1. ELS CONFIGURATION

C1.1 Electronic Line Shafting (ELS) Overview

Electronic Line Shafting or ELS is used to synchronize one or more slave axes to a master axis. An ELS master can be a real or virtual axis. A real master can be another axis in the system, or an external feedback device such as an encoder. A virtual master is a command generated by the CLC. Each slave axis can use either velocity, phase or cam synchronization. An ELS also includes the capability to jog each axis synchronously or independently, and to adjust phase offset and velocity while the program is running.

Since the master position is transmitted using SERCOS, all slave axes act upon the new master position at the same time thus insuring tight synchronization between all axes whether there are 2 or 25 slave axes.

The ELS Axis Configuration icon in VisualMotion is used to define the master and slaves in an ELS system but it doesn’t configure all the parameters that are necessary for an ELS application. Some parameters must still be configured through the Parameters Overview list or through parameter transfers in the user program.

C1.2 ELS Masters

The ELS master position and velocity, for all master types, can be monitored from System Parameters C-0-0157 and C-0-0158. System Register 30 is a status register for the ELS master. This register applies to all master types, but is used primarily for the virtual master. Status bits include: Master Type - Virtual/Real; Virtual Master enable; Virtual Master at programmed speed; Master Direction - Positive/Negative; and Master Stopped.

C1.2.1 Real Masters

If another axis in the system is the real ELS master, then the master is defined as Follow Axis Feedback. The master axis must be defined separately using the single axis configuration icon as single axis, rotary mode. The single axis gear ratio and zero velocity window are used here to configure the master.

If a master encoder is to be used, then the master is defined as either a Real Master through SERCOS or Daisy-Chained Real Master. If only one drive contains a DFF master encoder card, then it is a real master through SERCOS. If all slave axes contain DFF cards, then it is a daisy chained real master.

NOTE: GPS-5 and DIAx03 drives don’t support Daisy-Chained Masters.
System Parameters C-0-0159 (Real Master Zero Velocity Window), C-0-0161, C-0-0162 (Real Master Ratio Input & Output) and C-0-0163 (Real Master Filter Time Constant) should be configured through the Parameters Overview screen for these master types.

The zero velocity window determines when the master is stopped for diagnostic purposes or for a mode change. The input and output ratio are used as a “gear ratio” for the master encoder – e.g. it takes 4 revolutions of the master encoder to complete one “cycle” of the master. The displayed master position here is always rotary with a modulo of 360, regardless of the real master ratio. The filter time constant is used to filter out master encoder oscillations in the system caused by external disturbances.

C1.2.2 Virtual Masters

The Virtual Master is an axis internal to the CLC (Axis number 0). It is controlled using the acceleration, velocity, stop and go icons in the user program. Its units are degrees, RPM, and rads/sec^2 for position, velocity and acceleration, respectively. To “home” the virtual master, transfer a value of “0” to C-0-157 using a parameter transfer in the user program. When an E-Stop is issued by the CLC (reg.1 bit 3 =0), the virtual master decelerates to a stop using parameter C-0-156 –Virtual Master E-Stop Deceleration.

C1.2.3 Second Master

A second master can also be configured for ELS systems and the CLC can switch between these masters on the fly. The second master is configured as Master 2 and can be a virtual master, real master encoder, or follow axis feedback. Currently, there are no icons in VisualMotion to configure Master 2 so it must be done manually through the parameters “Overview” or through parameter init statements in the user program.

C1.3 ELS Slaves

After the master has been configured, the slave(s) must be setup. The following master/slave synchronization options are available:

Phase Synchronization
Velocity Synchronization
CAM Synchronization

Each one of these ELS modes can also be used with Dynamic synchronization

When a slave is not in synchronization with the master, it is in secondary mode. Secondary mode can be either single axis -- issue position commands to the axis -- or velocity – axis moves at a constant velocity. See Slave Secondary Mode (Non-Dynamic Synchronization).
C1.3.1 Phase Synchronization

Phase synchronization maintains the same relative position among axes, but adjusts the lead or lag of the slaves to the master in terms of degrees. It is used when the positions of axes are most critical. A slave configured as **ELS Phase** follows the position of the master linearly at the ratio specified in the *Turns* setup within the **ELS Axis Configuration icon** in VisualMotion.

ELS samples an absolute (one turn or multi-turn) master position, transmits that position to the slave axes and the slave axes execute the equation:

\[
\phi_{slave} = \phi_{master} \times (K_{slave} / K_{master}) + \text{phase\_offset}
\]

\[
\text{Slave position} = \text{sampled master position} \times (\text{slave turns/ master turns}) + \text{phase\_offset}
\]

If the turns ratio (Ks/Km) is \(1/4\), then the master must make 4 revolutions for the slave to complete 1 revolution. The position units for a phase slave are degrees and the slave modulo is always 360 * Ks/Km. To insure that there is no round off error in the turns ratio, the ratio must be entered using integer values.

One of the features of a phase slave is that it can be at a position offset from the master. This “phase offset” is by default automatically initialized when the slave is commanded to synch to the master – e.g. if the turns ratio is 1/1, the master is at position 100 and the slave is at position 120, the phase offset is initialized to 20.

The phase of the slave can be advanced (or retarded) while the system is in motion. The **ELS adjust icon** is used to move the slave to a commanded offset. When the phase adjust command is issued the axis performs a move on top of the current motion to the commanded offset position. An “Absolute” offset will move the axis so that the phase difference between the master and slave is equal to this value – e.g. if the turns ratio is 1/1, the master is at 100 and the slave is at 70, and an absolute phase offset of 20 is commanded, then the slave moves 50 degrees to 120.

An “Incremental” offset will add the specified number of degrees to the current phase difference – e.g. if the turns ratio is 1/1, the master is at 100 and the slave is at 70, and an incremental phase offset of 20 is commanded, then the slave moves 20 degrees to 90.
Phase Synchronization Features

With Diax03 drives, the phase adjust move can be controlled by either the CLC or by the drive. Axis parameter A-0-0164 (ELS Options) bit 1 determines if the phase offset will be controlled by the CLC (=0) or by the drive (=1).

If the phase offset is controlled by the drive it can use a drive filter or dynamic synchronization. If a drive filter is used, then the user specifies only the time in which to make the phase adjust. The drive uses parameter P-0-0060 (ELS Phase Adjust Time Constant) as the time constant for a low pass filter to calculate a jerk limited move profile for the phase offset.

Absolute Synchronization

The Diax03 drives can also be configured so that the phase offset is not automatically initialized at the time of synchronization. By setting bit 6 of Axis Parameter A-0-0164 to 1, the CLC initializes the phase adjust to the value in Axis Parameter A-0-0151 (Programmed Phase Offset). If Axis Parameter A-0-0151 is set to 40 in the above example, the slave would perform a 20 degree phase offset to position 140 when commanded to sync to the master.

Axis Parameter A-0-0164 can be set only in Parameter Mode (Phase 2). To set this parameter in the user program, use the “ParamInit” icon. Axis Parameter A-0-0151 can be set at any time with either a Parameter Transfer icon or with the ELS Adjust icon.

<table>
<thead>
<tr>
<th>Axis Parameter A-0-0164- bit 1</th>
<th>CLC Controlled Phase Offset</th>
<th>Drive Controlled With Filter</th>
<th>Dynamic Synchronization</th>
</tr>
</thead>
<tbody>
<tr>
<td>ELS Phase Adjust Time Constant</td>
<td>0 A-0-0155</td>
<td>1 P-0-0060</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>A-0-0153 used only when A-0-0155 = 0</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>ELS Phase Adjust Velocity</td>
<td>P-0-0034</td>
<td>P-0-0034</td>
<td>P-0-0143</td>
</tr>
<tr>
<td></td>
<td>A-0-0004 bit 10 = 1</td>
<td>N/A</td>
<td>P-0-0034</td>
</tr>
</tbody>
</table>

Relative Synchronization

If A-0-0164 bit 6 = 0, the CLC sets the phase offset to the difference between the master and slave position and no move is made.
C1.3.2 Velocity Synchronization

**Velocity synchronization** relates slave axes to a master in terms of rotational rate. It is used when axis velocities are most critical, as in paper processing operations in which two or more motors act on a single piece of fragile material.

A slave configured as **ELS Velocity** follows the velocity of the master at the ratio specified in the Turns setup. In this mode, the master velocity is calculated from successive master position samples and the slave is commanded to that velocity using the equation:

\[
V_{\text{slave}} = V_{\text{master}} \times (1 + \text{fine\_adjust}) \times (K_{\text{slave}} / K_{\text{master}})
\]

The “fine adjust” value in this case is an additive velocity expressed in percent of current speed. This can be used to fine adjust the following ratio. The limit on the fine adjust ratio is \( \pm 300\% \).

See Parameters P-0-0083, A-0-0150 and A-0-0159 to change the fine adjust value.

C1.3.3 CAM Synchronization

**Cam synchronization** is used when custom position, velocity or acceleration profiles are needed at a slave axis. These special profiles are developed at the slave by sending position commands every SERCOS cycle.

A cam is an (x, y) table of positions that relate a master axis to a slave. Cams can be stored on the CLC or on the digital drive. CLC cams have more adjustment options and can work with any SERCOS drive. Drive cams are more efficient and can be applied to more axes. The same programming commands and utilities are used for both drive-resident and CLC-resident cams.

The velocity profiles are generated from a cam (x,y) table that contains angular position data for the master axis and the corresponding cam position. A normalized cam commands 360 degrees of rotation for every 360 degrees of the master. It contains 1024 points where 0.0 \( \leq x \leq 360^\circ \) and 0.0 \( \leq y \leq 360^\circ \).

It is possible that the position of the master for a given SERCOS cycle does not match exactly with one of the x points from the table. In this case a linear interpolation between the two table points estimates the unknown position.
C1.3.4 Dynamic synchronization

Dynamic synchronization is available in DSM2.3-ELS-04VRS and GPS-05VRS firmware. In ELS phase and drive cam modes, each slave axis can independently ramp up to synchronization with a running master and lock on to its position. The drive controls the ramp and the phase offset profile.

For all ELS modes:
The ramp starts when the CLC commands the drive to switch into synchronization mode. The slave drive then accelerates at a parameter-determined rate to match the synchronous speed.

For phase synchronization and electronic cam modes:
At the end of the ramp, a phase offset will be performed automatically before the axis is synchronized. This compensates for any inaccuracy caused by the ramp being started too late or the velocity changing during the ramp.
While the slave axis is synchronized, phase offset can be adjusted at any time with a profile that has adjustable velocity and acceleration (like a single-axis move).

See the following Parameters for more information:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A-0-0151</td>
<td>Programmed Phase Offset</td>
</tr>
<tr>
<td>S-0-0228</td>
<td>Position Synchronization Window</td>
</tr>
<tr>
<td>P-0-0142</td>
<td>Synchronization Acceleration</td>
</tr>
<tr>
<td>P-0-0154</td>
<td>Synchronization Direction</td>
</tr>
</tbody>
</table>

Operation of Dynamic Synchronization

Rampup to synchronous mode
Set parameter P-0-0142 and P-0-0143. Since these parameters are stored through the service channel, this should be done only at the start of the program or when the values need to be changed.

Set the phase offset position with the ELS adjust user program command. At the end of the ramp, a phase offset move to this position will be performed.

To start ramping to the synchronous velocity, execute the ELS mode change user program command with the ‘Sync to Master’ option. The drive will start the rampup-lockon procedure, based on the phase offset and the synchronization acceleration.
Rampdown to single-axis mode
Execute the ELS mode change command with the ‘Single Axis’ option. To immediately stop the axis, execute the Axis Stop command. To stop to a position, execute the Axis Move command with the absolute position desired.

NOTE: If the axis stop command does not follow the axis mode change command, the axis will move to the last stored target position. Therefore, it is necessary to add a stop command or a new move command after switching into single-axis mode.

Synchronization Status
Axis status register bit 4 ‘Phase Adjusted’ indicates the phase offset status. If the difference between the drive’s current position and its synchronous position (output of ELS or cam equation) is within the synchronization window (S-0-0228), the bit is set to (1). Otherwise, it is (0).

C1.3.5 Slave Secondary Mode (Non-Dynamic Synchronization)
When a program begins execution, either at the beginning of a cycle or after an error, all slaves are in their secondary mode. They will not start following the master until a synchronize command is issued through the “ELS Mode Change” icon. When this command is issued, the slaves establish a relative synchronization to the master. If the master is moving when the synch command is issued, the slaves do not ramp up to the master speed (this feature, called Dynamic Synchronization, is available with version 4 of the Diax03 drive firmware).

The master should either be at standstill when the command is issued or the drives should be in velocity mode with a velocity equal to that of the master. When switching from synch mode to secondary mode, the axis does not ramp to a stop either. If switching to single axis positioning secondary mode, then the velocity is immediately commanded to 0. If switching from synch to velocity secondary mode, then the drive axis will continue moving at constant velocity equal to the last sampled master velocity. To stop the axis, either the “Stop” icon must be issued or a velocity of 0 commanded through the “Velocity” icon.
C1.3.6 Summary of ELS Parameters and Registers

**Master**

**Setup/Control**
- Zero Velocity Window C-0-0159
- Real Master Ratio Input C-0-0160
- Real Master Ratio Output C-0-0161
- Real Master Filter Time Constant C-0-0163

**Status**
- Position of master (fmaster) - degrees C-0-0157
- Current Velocity (Vmaster) - rpm C-0-0158
- Virtual Master at Programmed Speed Reg. 30 bit 7
- Master Direction: Positive/Negative Reg. 30 bit 9
- ELS Master Stopped Reg. 30 bit 10

**Slave**

**Setup/Control**
- Phase Offset Controlled by CLC/Drive A-x-0164, bit 1
- Phase Adjust Parameters see previous table

**Status**
- Position of slave (fslave) - degrees S-0-0047
- Velocity of slave (Vslave) - rpm S-0-0036
- Phase Adjusted Reg. 31 - 38 bit 4
- ELS Enabled Reg. 31 - 38 bit 5
- Secondary Mode Enabled Reg. 31 - 38 bit 6
- Phase Difference P-x-0034
C1.4 Cam Applications

A cam is an (x, y) table of positions that relate a master axis to a slave. ELS (Electronic Line Shaft) systems make use of cams when special position, velocity or acceleration profiles are needed at a slave axis. These special profiles are developed at the slave by sending position commands every SERCOS cycle.

C1.4.1 Overview

Cams can be stored on the CLC or on a digital drive (Indramat DDS 2.2 or a similar DIAx03 drive). CLC cams have more adjustment options and can work with any SERCOS drive. Drive cams are more efficient and can be applied to more axes. The same programming commands and utilities are used for both drive-resident and CLC-resident cams.

A normalized cam commands 360 degrees of rotation for every 360 degrees of the master. It contains 1025 points, 1024 intervals, where

$$0.0 \leq x \leq 360^\circ \text{ and } 0.0 \leq y \leq 360^\circ.$$

It is possible that the position of the master for a given SERCOS cycle does not match exactly with one of the x points from the table. In this case a linear interpolation between the two table points estimates the unknown position.

A master axis may turn in a positive or negative direction without loss of cam function.

C1.4.2 Cam Commands

The following commands are used to initialize and adjust cams. Refer to Chapter 6, Icon Programming for more information.
C1.5 Building Cam Tables

There are two ways to build a cam within VisualMotion:

- The Cam Build Icon (See Chapter 6. Icon Programming).
- The CAM Building Utility (See Chapter 4. VisualMotion Menu Commands). Refer to the Cam Builder Selection under the Tools Menu.

A spreadsheet can also be used to build a Cam Table. In the spreadsheet method, the equations that define the cam can be developed and executed to generate the xy table of positions. Although the xy table need not be normalized, it is a good idea to use the normalized form. The cam shaping parameters are easier to use when the cam table is normalized. See Section Cam Equation.

Cams enter the CLC as text files in CSV format, the kind most spreadsheets generate.

If the CSV file contains less than 1025 points, an algorithm within the CLC fills in the missing points. As a rule, a CSV cam file should contain at least 200 points, anything less than that does not sufficiently define the cam and unexpected results may occur.
C1.5.1 PCAM Cam Build Type

The PCAM build type accepts input in the form of a table of target positions. The utility generates smooth table values between the targets, each target being viewed as a separate accelerate/decelerate type motion. Thus, a start/stop type motion is embedded in the output cam table for each target specified in the input file.

It is a good idea to limit the number of targets to less than 10 and to keep sufficient distance between targets to allow for a fully formed motion profile between them. A rule of thumb is to have at least 20 degrees between targets.

Example:

With this position table the PCAM build type will generate the following cam table:

Since all transitions in position (such as above - 0 to 10 degrees) use a start/stop type profile, the number of transitions must be limited and must not be too close together. Otherwise, the CAM motion will become jerky.
C1.5.2 VCAM Cam Build Type

The VCAM build type accepts a velocity profile as input and outputs a normalized profile. The following examples use Time - Velocity instead of Degrees - Velocity for the input data. Both units can be used to build output data.

Example 1:

With the velocity profile defined as above the VCAM build type will generate the following velocity profile curve:

The total distance traveled by the slave is 360 degrees since the profile is normalized. In CAMs that cycle back to a zero position, it is important that the total area under the velocity curve defined by the input file be zero, otherwise the CAM will gain or lose a position offset every cycle (as illustrated in the PCAM example).
Example 2 - Velocity Profile with Dwell:

In this next example the velocity profile of the cam contains a 25% of cycle dwell.

The resulting cam velocity profile follows:

By adding more lines of input data the velocity profile can be shaped to almost any degree.

The velocity profile of the cam will match the original input file when the master cycle time is 0.80 second.

As the master cycle time increases the cam profile will stretch over time but the total position commanded to the slave remains 360 degrees every cycle.

Likewise, as the master cycle time decreases the cam profile will shrink but the total position commanded to the slave remains 360 degrees every cycle.
C1.5.3 ACAM Cam Build Type

ACAM build type accepts an acceleration profile as input and outputs a normalized profile. The following examples use Time - Acceleration instead of Degrees - Acceleration for the input data. Both units can be used to build output data.

Example:

The distance traveled by the slave is 360 degrees since the cam is normalized.

It is important that the total area under the acceleration curve defined by the input file be zero, otherwise the cam will gain or lose speed every cycle and thus be unstable. The acceleration profile of the cam will match the original input file when the master cycle time is 1 second.

As the master cycle time increases the cam profile will stretch over time but the total position commanded to the slave remains 360 degrees every cycle. Likewise, as the master cycle time decreases the cam profile will shrink but the total position commanded to the slave remains 360 degrees every cycle.
C1.5.4 SCAM Cam Build Type

The SCAM build type accepts input in the form of target positions and outputs a cam by fitting these position to third order spline polynomials.

Use of this type of cam builder is more complex than the others due to the interplay between the target positions used.

It is possible to control cam shape and acceleration properties to almost any degree with this utility, but skill is required in choosing the proper target positions.
C1.6 Cam Equation

Cams execute every SERCOS cycle and are governed by the following equation:

\[ \text{Scmd} = H \times \text{CAM} \times [(M/N) \times \text{Mcmd} + \text{Mph}] + L \times \text{Mcmd} + \text{Sph} \]

Where,

- \( \text{Mcmd} \): Is the position command of the master (degrees)  
  - C-0-0156
- \( \text{M} \): Multiplying scaling term of the master position  
  - A-0-0032
- \( \text{N} \): Divisional scaling term of the master position  
  - A-0-0031
- \( \text{Mph} \): Master Cam phase adjust (degrees)  
  - A-0-0151
- \( \text{Sph} \): Slave Cam phase adjust (degrees)  
  - C-0-3101…3108
- \( \text{CAM} \): Normalized 1025 cam table  
  - A-0-0033
- \( H \): Cam scaling term  
  - A-0-0035
- \( L \): Linear scaling term of master position  
  - A-0-0162
- \( \text{Scmd} \): Commanded position to slave  
  - A-0-0163

The effect a parameter has on motion can be seen by setting all other parameters to zero or one.

**C1.6.1 Consider** \( H = 0 \) **and** \( L \neq 0, \text{Sph} = 0 \).

\( \text{Scmd} = L \times \text{Mcmd} \).

\( L \) has the effect of introducing the position command of the master (\( \text{Mcmd} \)) into the cam equation, but in a linear way.

Setting \( L = 1/2 \) means that the slave axis makes one revolution to every two revolutions of the master, or the slave has 1/2 the speed of the master.

With \( L = 1 \) the slave and master move the same distance at the same speed. Thus, \( L \) effects the speed of the slave in a linear way.
C1.6.2 Consider $H \neq 0$, $L = 0$, $M = N = 1$ and $Mph = Sph = 0$.

$Scmd = H*CAM[Mcmd]$.  

**What happens if $H = 1$?** The slave axis is completely governed by the cam table. For each cycle of the master, the slave makes one cycle, or for each revolution of the master axis, the slave axis makes one revolution. The difference here is that the relationship between the slave and master is not linear.

Consider the acceleration profile defined by the ACAM utility example *See Section ACAM Cam Build* Type. If the master axis is running at a constant velocity, what kind of motion is observed at the slave?

![Master Axis diagram](image)

At the start of the cycle (master at zero degrees) the slave is at rest (stopped). When the master reaches 36 degrees the slave passes through peak acceleration and is speeding up rapidly. At 72 degrees the slave is at constant speed. The slave remains at constant speed until the master is at 288 degrees, here the slave starts to decelerate. Peak deceleration is achieved at 324 degrees, and by end of cycle the slave has stopped again.

![Slave Axis diagram](image)
What distance has the slave traveled? Since the cam table is normalized we know that the y side (output side) of the table varies from 0 to 360 degree so the slave has traveled one revolution.

To summarize, the master axis rotates at constant velocity while the slave rotates at the same rate but does so by accelerating, traveling at constant speed, then decelerating to a stop. Thus the slave constantly starts and stops for every cycle of the master. This is perfect motion for indexing something into position once every cycle.

**What happens if H = 2?** The only thing that changes is the distance traveled by the slave. Instead of the slave making one revolution each cycle, as the master does, it makes two revolutions. So, H = 2 doubles the distance traveled by the slave. On the other hand, H=1/2 cuts the slave distance in half.
C1.6.3 Consider \( H = L = 1, \ M = N = 1, \) and \( Mph = Sph = 0? \)

Now the slave makes two revolutions each time the master makes one, but instead of starting and stopping, the slave rotates continually. In fact, the slowest the slave axis moves is at whatever the rate the master is.

![Graph showing slave and master velocities](image)

In other words, at the start and end of every cycle, the slave is rotating at the same speed as the master, but then speeds up and slows down to gain another whole revolution every cycle. What happens is the velocity profile of the cam has been added to the line speed of the master.
C1.6.4 Consider L = 0, H = 1, M = 2, N = 1 and Mph = Sp = 0

As the master makes one revolution (360 degrees) the input to the cam table is doubled since (M/N)*Mcmd = 720 degrees, and so the cam table is traversed twice for every cycle of the master. The effect is that the slave advances two revolutions to each one of the master.

The effect M = 2 and H = 1 has is not the same as that of M = 1 and H = 2. The distance traveled by the slave is the same in both cases but the profiles used to travel that distance are vastly different.

With M = 2 and H = 1, the slave starts and stops twice for each master cycle, however, for M = 1 and H = 2 the slave starts and stops only once. M = 2 has the effect of putting out the cam profile twice per each master cycle, whereas H = 2 has the effect of doubling the distance traveled during one cam cycle.

In essence, the effect M and N has is to define the number of cam cycles per master cycle.
C1.6.5 Consider L = 0, H = M = N = 1 and Mph = 90, Sph = 90 degrees.

Finally, the phase adjust term needs to be considered. The effect here can be seen by stepping the slave through one cycle.

At the top of the cycle (master at 0°) the slave is 90 degree into the cam table. At that point, the slave is at constant speed and 18 degrees out of the acceleration stage. In other words, the cam profile is leading the master cycle by 90 degrees.

The effect Mph has is to slide the cam profile right (-phase) or left (+phase) relative to a master cycle.
The effect Sph has is to slide the cam profile right (-phase) or left (+phase) relative to a slave.

**C1.7 CLC Cam Alignment**

It is possible to align a slave axis to the output position of the cam by using the align status, the current slave position, and single-axis positioning. Drives that do not support single-axis mode (i.e. the RAC) cannot use the alignment function.

The following procedure can be used in the CLC user program before switching an axis into synchronization mode:

1. Check the Aligned Status bit (axis status bit 8) to determine if a move needs to be performed.
2. Switch the axis to single axis mode using the ELS/MODE instruction.
3. Set the programmed velocity to the axis phase adjust velocity (A-0-0153).
4. Set the programmed acceleration to the axis phase adjust acceleration (A-0-0154).
5. Get the slave position corresponding to the current master position according to the cam table equation from parameter A-0-0163.
6. Make a single-axis absolute move to the position in A-0-0163 using the AXIS/MOVE command. The drive currently supports only the shortest path mode. If the same direction is desired, do two successive moves.
7. Enable motion using the AXIS/START instruction (GO icon).
8. Wait for the move to be completed by using the AXIS/WAIT instruction or by testing the in-position or alignment status of each drive being aligned.
9. Switch the axis to cam mode using the ELS/MODE instruction.
C1.7.1 Aligned Status Bit (Axis Status Register bit 8)

This bit provides the status of cam alignment. It is set to (1) if the axis is aligned to the cam, and (0) if it is not aligned.

The following conditions set this bit to (0):
1. The axis is not configured in the program to be a cam axis.
2. A valid cam is not active for this axis.
3. The absolute value of (position of the axis - slave position from cam equation) is greater than the in-position window (drive parameter S-0-0057).

The following conditions set this bit to (1):
1. The axis is synchronized to the master
2. The absolute value of (position of the axis - slave position from cam equation) is less than or equal to the in-position window (drive parameter S-0-0057).
C1.8 Drive Cams

Indramat digital drives include the ability to store and execute cams with up to 1024 points. The positions are defined with a normalized table that is based on the ELS master position. The slave axis units and scaling are defined by the shaft distance (H). It is possible for the slave axis to have either rotary or linear units.

C1.8.1 Drive Cam Equation

The electronic cam mode on the digital drives can be described by the following equations:

a) When switching to synchronization mode, the following equation is immediately applied in the drive:
   \[ Scmd = (H \times \text{Cam}(Mcmd - Mph)) + Sph \]

b) As long as synchronization mode is active, the drive updates the position difference:
   \[ \Delta Scmd = (H \times \Delta \text{Cam}(Mcmd - Mph)) + \Delta Sph \]

\[
\begin{array}{lll}
\text{Mcmd} & \text{Position command of the master (degrees)} & \text{C-0-0157} \\
\text{Mph} & \text{Master phase adjust (degrees)} & \text{P-0-0061} \\
\text{Cam} & \text{Normalized 1024 point cam table} & \text{P-0-0072, P-0-0092} \\
\text{H} & \text{Cam shaft distance (deg, mm, or in.)} & \text{P-0-0093} \\
\text{Sph} & \text{Slave phase adjust (deg, mm, or in.)} & \text{A-0-0151, P-0-0034} \\
\text{Scmd} & \text{Commanded position to slave} & \text{S-0-0047} \\
\end{array}
\]

The active cam is switched after the parameter stored in P-0-0094. A status of the active cam is in P-0-0089.

C1.8.2 Axis Configuration

Drives that use the DDS cams are configured as ELS axes with the Cam Synchronization mode. This is a fourth ELS mode in addition to velocity synchronization, phase synchronization, and CLC cam mode. Note that the drives are set up at initialization time to use either CLC cams or drive-resident cams. An axis may not switch between both types of cams or switch to cams not stored on that drive.
C1.8.3 Drive Cam Table Parameters

For drive-resident cams, the drive parameters P-0-0072 and P-0-0092 are used. The cams on
the drive always contain 1024 points. These points are distributed in equal distances over a
range of 360 degrees, resulting in a point for every .35 degrees of the master. The table
values have a permissible range of -200% to +200%.

For example:

<table>
<thead>
<tr>
<th>Point Number</th>
<th>Value (Percent)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.000000</td>
</tr>
<tr>
<td>2</td>
<td>1.100000</td>
</tr>
<tr>
<td>3</td>
<td>2.300000</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>1023</td>
<td>0.400000</td>
</tr>
<tr>
<td>1024</td>
<td>0.000000</td>
</tr>
</tbody>
</table>

The table is made up of points weighted in percent of the H factor (or shaft distance). There is
a linear interpolation between the points dependent upon master axis position. The position
command is determined by multiplying the table values by the H factor, which can be changed
while the cam is running.
C1.8.4 Drive Cam Alignment

It is possible to align a drive-based cam slave axis to the output position of the cam by using the align status, the current slave position, and single-axis positioning.

The following procedures can be used in the CLC user program before switching a cam axis into synchronization mode. *As an option to this, Dynamic synchronization can also be used.*

**If modulo positioning is enabled:**

1. Check the Phase Adjusted bit (axis status bit 4) to determine if a move needs to be performed.
2. Switch the axis to single axis mode using the ELS/MODE instruction.
3. Set the programmed velocity. A good value to use is the homing velocity (S-0-0041).
4. Set the programmed acceleration. A good value to use is the homing acceleration (S-0-0042).
5. If shortest path motion is needed, set drive parameter P-0-0013 to 0. For motion in the positive direction only, set this parameter to a 1.
6. Parameter P-0-0034 (Actual value of additive pos. cmd.) contains the difference between the current axis position and the position at the output of the cam generator corresponding to the current master position. Read this parameter into a variable.
7. Calculate the distance for an incremental move to: distance = S-0-0103 - ((P-0-0034 - slave_offset) % S-0-0103), where slave_offset is the slave phase offset to be established at synchronization.
8. Start a single-axis incremental move of distance in the equation above using the AXIS/MOVE command.
9. Enable motion using the AXIS/START instruction (GO icon).
10. Wait for the move to be completed by using the AXIS/WAIT instruction or by testing the in-position status of each drive being aligned.
11. Switch the axis to cam mode using the ELS/MODE instruction.
If absolute positioning is enabled:

1. Check the Phase Adjusted bit (axis status bit 4) to determine if a move needs to be performed.
2. Switch the axis to single axis mode using the ELS/MODE instruction.
3. Set the programmed velocity. A good value to use is the homing velocity (S-0-0041).
4. Set the programmed acceleration. A good value to use is the homing acceleration (S-0-0042).
5. Parameter P-0-0034 (Actual value of additive pos. cmd.) contains the difference between the current axis position and the position at the output of the cam generator corresponding to the current master position. Read this parameter into a variable.
6. Calculate the distance for an incremental move to: distance = -(P-0-0034 - slave_offset), where slave_offset is the slave phase offset to be established at synchronization.
7. Start a single-axis incremental move of distance in the equation above using the AXIS/MOVE command.
8. Enable motion using the AXIS/START instruction (GO icon).
9. Wait for the move to be completed by using the AXIS/WAIT instruction or by testing the in-position status of each drive being aligned.
10. Switch the axis to cam mode using the ELS/MODE instruction.

C1.8.5 Phase Adjusted Status (Axis Status Register bit 4)

This bit provides the status of cam alignment, as well as the status of a phase adjust. It is set to (1) if the axis is aligned to the cam, and (0) if it is not aligned. It is set from the cyclic value of drive parameter S-0-0182, bit 8. Note that the ‘phase adjusted’ bit is set before cam synchronization only if phase offset is performed on the drive, not the CLC.

The following condition sets this bit to (0):
The absolute value of (position of the axis - slave position from cam equation) is greater than the position synchronization window (drive parameter S-0-0228).

The following condition sets this bit to (1):
The absolute value of (position of the axis - slave position from cam equation) is less than or equal to the position synchronization window (drive parameter S-0-0228).
C1.8.6 Drive Cam Configuration Notes

1. Cam shaft distance (H) is controlled through the service channel by default. In the cam/engage command, it is only sent when it changes. To put cam shaft distance in cyclic data, set parameter A-0-0180 or 0181 to (32768+93). The cam/engage command will then send the cyclic value. Note that the drive interpolates the H factor so that there are no discontinuities in the profile.

2. The slave phase offset for drive based cams is the same parameter used in ELS (A-0-0151 and S-0-0048). By default, this parameter is in the cyclic data.

3. The master phase offset for drive-based cams is sent through the service channel (P-0-0061) by default. Like the H factor, cyclic data can be used by setting A-0-0180 or A-0-0181 to (32768+61). Note that unlike the slave phase offset or the H factor, there is no filter on this value, on either the drive or the CLC.

4. For all cam axes, the positioning is set to rotary in the user program by default. To run a cam with a linear axis, add a parameter init. of bit 2 in A-0-0004 to 0 after the els_init command.

5. A-0-0031 to 0035 are not used for drive-based cam axes. These are battery-backed parameters on the CLC that are used only for CLC-based cams.

6. A-0-0161 to 0163 are not used drive-based cams. The cyclic slave phase offset parameters (A-0-0151 to A-0-0157) are inconsistent with CLC cams but are consistent with ELS slave axes.

7. If parameter A-0-0164, bit 6 is set to (1), the phase offset can be initialized to any value before the drive is switched into synchronization mode. This allows the user to specify a phase offset according to the range of motion. If the bit is set to (0), the CLC automatically establishes relative synchronization by setting the phase offset to the value in P-0-0034, as it does in ELS phase synchronous mode.

8. Parameter A-0-0163 is used for CLC-based cams only. Its equivalent for drive-based cams is: slave_position = S-0-0051 - P-0-0034.

9. Using the cam/build instruction, it can take up to 6 seconds to calculate the cam and send it to the drive.
C1.9 Differences between CLC and Drive Cams

In most cases, drive-resident cams should be used if the drive software supports them. There are some features not implemented in the drive at this time that may require the use of cams stored on the CLC.

<table>
<thead>
<tr>
<th></th>
<th>CLC Resident</th>
<th>Drive Resident</th>
</tr>
</thead>
<tbody>
<tr>
<td>Drive Versions Supported</td>
<td>All SERCOS drives</td>
<td>Indramat DDS2.2 and similar drives (DIAX03)</td>
</tr>
<tr>
<td>Number of cam axes allowed</td>
<td>4</td>
<td>maximum allowed in system</td>
</tr>
<tr>
<td>Number of cams allowed</td>
<td>8</td>
<td>2 * number of drives in system</td>
</tr>
<tr>
<td>CLC resources required</td>
<td>large amount</td>
<td>minimal</td>
</tr>
<tr>
<td>Cam Equation</td>
<td>Scmd = H*CAM[(M/N)<em>Mcmb + Mph] + L</em>Mcmb + Sph</td>
<td>Scmd = H*CAM[Mcmb] / 100 + Sph</td>
</tr>
<tr>
<td>Cam Factors</td>
<td>M, N, H, and L</td>
<td>H</td>
</tr>
<tr>
<td>Table Format</td>
<td>2 column: x= master, y=slave</td>
<td>1 column of slave values</td>
</tr>
<tr>
<td>Master Intervals in Table</td>
<td>variable through x-column</td>
<td>fixed at 0.35 degrees</td>
</tr>
<tr>
<td>Number of Points in Table</td>
<td>200-1024</td>
<td>always 1024</td>
</tr>
<tr>
<td>Slave Units</td>
<td>Position Units</td>
<td>Percent of H factor (100.0 is 1 unit)</td>
</tr>
<tr>
<td>Table Switching Point</td>
<td>0 degrees</td>
<td>selected by parameter</td>
</tr>
<tr>
<td>Alignment before Sync.</td>
<td>Absolute move to A-0-0163</td>
<td>Incremental move of -(P-0-0034 - initial offset)</td>
</tr>
<tr>
<td>Interpolation update rate</td>
<td>2ms with 2 cams, 4ms if &gt; 2 cams</td>
<td>500 microseconds</td>
</tr>
</tbody>
</table>
C1.10 Application (Rotary Cutoff)

In this application a cam profile is designed that drives a two bladed knife used to seal and cut pre-wrapped candy bars on a conveyor belt.

Consider the following drawing:

![Rotating Knife Application](image)

In this application the ELS line master is driving motor r2 directly and motor r1 is the cam axis.

A cam table with a dwell is need so that during the dwell the knife will match line speed to allow for proper sealing and cutting. The placement of the dwell within the cam will effect the master phase adjust, Mph, needed for centering the cam to the product cycle.

Define Ry to be the number of master revolutions/product cycle.

Then,

\[ Ry = \frac{y}{(2\pi r_2)} \]

To set the cam cycle to the product cycle use M and N.
One cycle of the normalized cam for every product implies:

$$\text{Ry} \cdot (M/N) = 1 \text{ revolution}$$

so,

$$N = \text{Ry}$$
$$M = 1$$

For the moment, assume H = 0.

L is set to match distance and speed of the knife during a cycle to that of the product.

The equation for L is as follows:

$$\left(2\pi \cdot r_1\right) \cdot (\text{Ry}) \cdot L = y$$
$$L = \frac{r_2}{r_1}$$

The distance factor $$\left(2\pi \cdot r_1\right) \cdot (\text{Ry})$$ is the total distance traveled by the knife in one product cycle. L is used to scale this distance to y, the product length.

Define B to be the number of knife blades, thus B=2, and d to be the distance between knife blade(s).

If y equals d then the distance traveled by the knife in one product cycle is the same as the product length. In this case no cam is need and H = 0.

If y is not equal d then the knife never hits the product at the right time in the cycle. This is where H comes in and a cam is need.

H is used to scale the cam to make up for any length difference between y and d.

The distance between knife blades is defined by:

$$d = \left(2\pi \cdot r_1\right) / B$$

The distance adjustment needed each cycle is given by:

$$d_{\text{adj}} = d - y$$
The cam is normalized so it makes one cycle per product cycle, therefore:

\[ B^*d^*H = d_{\text{adj}} \]
\[ H = (d - y)/(B^*d) \]
\[ H = (1 - y/d)/B \]

Note that \( B^*d = 2^*\pi^*r1 \).

*H may be zero.* When \( H \) is zero the cam profile is not needed and this application can be done without a cam.

*H may be negative.* This happens when the distance between the blades is smaller than the product length, so that the knife will slow down during the cam cycle then speed up and be at line speed during the dwell. The distance adjustment attributed to the cam is exactly \( d-y \).

*H may be too negative.* In this case the cam will slow down the knife so much that it actually will reverse direction and travel backwards. This is not acceptable and only happens when the product length is at least twice that of the knife. For this case, \( L \) must be set to zero and a cam profile developed that will start from rest, accelerate to match line speed then decelerate to a stop. The details of this approach is not addressed here.

*H may be positive.* In this case the distance between the knife blades is larger than a candy bar so the knife will speed up during the cam cycle then slow down and match line speed during the dwell.

*H may be too positive.* In this case the amount of distance the cam profile must make up is so large and the line speed may be so high that the acceleration and deceleration rates are not acceptable to the motors.

Phase can be added or subtracted to center the cutting edge of the knife exactly between the candy bars.

For pre-production setup initialize the ELS master and master phase adjust to zero degrees. Center the cam dwell at zero degrees of the cam. Set the slave phase adjust so that the knife is positioned at its first cut.

During production, use the master phase adjust for registration correction as needed.
D1. CLC DRIVE PARAMETER EDITOR

D1.1 Overview

When opened, the CLC Drive Parameter Editor uploads the current status information for drive #1 (the default). The current drive status is also displayed. The position, Velocity and Acceleration values from the CLC card are displayed along with the feedback status from the selected drive.

The Drive # box allows selection of another drive by entering a drive number or scrolling with the up/down list buttons. Clicking Update uploads the current values for the selected drive.

Since the drive internally generates rate profiles for single axis motion, the programmed acceleration is also displayed. Acceleration is not shown for coordinated motion since the CLC path planner manages acceleration for coordinated motion.
D1.2 File Menu

D1.2.1 Transfer Parameters

This menu item uploads parameters to a file for archiving or viewing, and downloads archived parameters from a file to a CLC card.

![Drive Parameter Transfer Window]

Each drive has one parameter set that can be transferred. Drive parameters need a drive number. The parameter set for the selected drive connected to the CLC may be transferred through the SERCOS communication system.

Parameters may be uploaded in one of two formats. Uploading for archiving saves the file with the ".prm" file extension in the \CLC\SAVESET directory, with the data in the proper format for downloading to the CLC. Uploading for viewing saves the file to the same subdirectory as a text file with a ".txt" extension and may be viewed using Notepad or another ASCII text editor or file viewer. A *.txt parameter set uploaded for viewing cannot be downloaded to the CLC.

D1.2.2 Print

Option not yet available
D1.3 Parameter Menu

D1.3.1 Analog Outputs

Choosing Analog Outputs from the Parameters menu opens the Drive "n" Analog Output Settings dialog box. This is used to manage signal selection and scaling of the AK1 and AK2 analog output drive signals. By default, the current values for drive #1 are uploaded and displayed.

![Analog Outputs Editor](image)

The Drive # box allows selection of another drive by entering a drive number or scrolling with the up/down list buttons. Clicking Update uploads the current values for the selected drive. Scrolling lists of signals available for both outputs include:

- No output
- Current command value
- Velocity feedback value
- Velocity command value
- Position command value difference between each SERCOS cycle (2ms).
- Position feedback value 1 (motor encoder)
- Position feedback value 2 (external encoder)
- Lag error
- Sine signal from motor encoder
- Cosine signal from motor encoder
- Sine signal from external encoder
- Cosine signal from external encoder
Individual data entry boxes allow independent scaling of each output (maximum 10 volt output) to meet the requirements of an external indicator (analog or digital panel meter, etc.).

Clicking on the Save button downloads the signal selections and scaling factors to the selected drive through the CLC and SERCOS communications.

**D1.3.2 Drive Direction**

Drive Direction is an edit window for viewing and setting a drive direction parameters. These parameters invert the direction of the commands to the drive. (i.e. A 5 inch move will move 5 inches negative if the Position Command is set to the reverse direction)

To view settings for a drive:
In the main window select the drive number and press the Update button.

To change a drive's settings:
Settings can only be changed in parameter mode. Select the direction buttons and press the Save button. *See drive parameters S-0-0043 and S-0-0055.*
D1.3.3 Drive Name

This window displays the name of the currently active drive. The default name is the axis number however a custom name can be assigned and saved to any of the drives on the SERCOS ring.

![Edit Name Drive](image)

D1.3.4 Drive Monitoring

![Monitoring](image)

Selecting Drive Monitoring from the Parameters menu opens the Drive Monitoring window and refreshes the following drive information:

- **Position window** (S-0-0057) sets the tolerance distance used to determine if it’s in position.
- **Standstill window** (S-0-0124) sets the velocity to determine if it’s at a standstill.
- **Absolute Encoder Difference** (P-0-0097) sets the maximum distance the motor can move when off, without causing an error 76 on powerup to phase 4.
- **Monitoring window** (S-0-0159) sets the maximum position unit (or percent for some drives) from the command value before the drive issues an error 28.

The current **Maximum Model Deviation** from the command value is displayed in position units (or a percent for some drives). This value can be reset using the button on the bottom of the screen.
### D1.3.5 Drive Tuning

Selecting Drive Tuning from the Parameters menu opens the Drive Tuning dialog box for the currently active drive. The adjustments within the Velocity loop are related to a machines performance. The Current Loop adjustments are set according to the respective motor/drive combinations and should not be altered from their initial operative setting.

![Drive Tuning Diagram](image)

The Velocity Loop **Proportional Gain** data entry box adjusts the gain of the loop feedback path (SERCOS parameter S-0100). The gain is initially adjusted by the drive/motor combination for a 1:1 load/motor inertia ratio. This parameter value may be set from 0 to 65535.

The **Smoothing Time Constant** data entry box sets a low-pass filter that limits the bandwidth of the feedback loop and reduces digital quantization effects (Indramat parameter P-0-0004). The time constant is set in microseconds, any entry under 250\(\mu\)s switches off filtering.

The Velocity Loop **Integral Action Time** data entry box also sets a low pass filter time constant integrating the velocity loop feedback signal (SERCOS parameter S-0-0101). This parameter is typically used to adjust the loop response time, matching the load to motor and reducing overshoot that may result from a rapid (step) change. This filter has a lower frequency breakpoint than the Smoothing/Roll-off value.

The Current Regulator **Proportional Gain** (S-0-0106) and **Integral Action Time** (S-0-0107) adjustments are used for the initial tuning of respective motor/drive combinations. This adjustment should not be changed once it is set for a specific system.
Clicking the Defaults button loads default parameter values into the dialog box data entry fields (CLC must be in Parameter Mode). The default values assume a 1:1 ratio of load inertia to motor inertia.

For further information on tuning Indramat drives and tuning parameters refer to the Digital Servo Drive User's Manual.

**D1.3.6 Drive Limits**

Positive and Negative **Travel Limits** set floating point values for the drive Positive (S-0-0049) and Negative Position Limit Value (S-0-0050) parameters. The Travel Limits are not affected unless Enabled is checked.

![Drive Limitations dialog box](image)

The **bipolar velocity limit** value (S-0-0091) determines the maximum allowable speed in either direction. If the velocity limit value is exceeded, the drive responds by setting the message "ncommand > nlimit" in Class 3 Diagnostics (IDN S-0-0013).

The **bipolar acceleration** parameter (S-0-0138) reduces the maximum acceleration ability of the drive symmetrically around 0, to the programmed value in both directions.

The **bipolar jerk limit** value (S-0-0349) determines the maximum allowable jerk in either direction.

The **bipolar torque limit** value (S-0-0092) determines the maximum allowable torque in either direction. If the torque limit value is exceeded, the drive sets the message "T > Tlimit" in Class 3 Diagnostics (S-0-0013).
D1.3.7 Drive Reference

This menu item will automatically sense the active drives motor encoder type and launch either the single or multi-turn encoder dialog box.

Single-turn Encoder
Selecting incremental encoder from the drive reference menu opens the Homing Setup dialog box and uploads the current parameter values for drive #1 (the default).

Homing does not need to be configured if the Home icon or AXIS/HOME text language command are not used in user programs. Homing is an internal capability of Indramat intelligent drives and requires only that the CLC send a home command to the drive. The actual homing procedure performed by the drive is set by the drive parameters.

The Drive # box allows selection of another drive by entering a drive number or scrolling with the up/down list buttons. Clicking Update uploads the current values for the selected drive.

Because the setup of drive homing parameters is highly system dependent, refer to the Drive User’s Manual for information on the various homing routines and the requirements for setting up the homing parameters.
Multi-turn Encoder
Reference Absolute Encoder instructs the DDS drive to use the current shaft position as the position indicated in the "Reference Distance".

Set the drive number to the drive to be referenced. Move the motor to the position to be referenced. Set the "Reference Distance" to the value to be assigned to this position.

Press the Continue button. A second screen will inform you that all motion on this drive will be referenced to these values. Press OK to accept, or Cancel to Abort.

After successful completion of this procedure, the checkbox 'Drive referenced to machine zero' will be checked.
D1.3.8 Mechanical

Selecting Mechanical from the Parameters menu opens the Mechanical Setup Drive dialog box and uploads the current values from the drive. This dialog box allows easy access to several important parameters, which must be set before running any motion programs.

Data Positioning with respect to the load configures the drive to use Encoder 2 to close the position loop and provide cyclic feedback from drive parameter S-0-0053. This sets S-0-0004 - Bit 11: Use Secondary Encoder for Positioning equal to 1. This option must be set if the CLC is used with DDS-2.1 linear motor firmware.

The Units of measure for position data selection has a choice of inches, millimeters or radians for the CLC system-wide unit of measurement (A-0-0005).

The Type of scaling selection can be linear or rotary (A-0-0004 bit 2). When linear is selected absolute positioning is enabled in the drive. When rotary is selected, position is in degrees, velocity in RPM, and acceleration in radians/sec2.

The Feed Constant allows setting the ratio of movement in system units resulting from each revolution of the driven shaft (S-0-00123). For example, a 5 TPI ball screw provides 0.200 inch movement per revolution. The Calculator icon opens a scientific calculator that may be used to determine the feed constant.

Output (Z2) and Input (Z1) Revolutions permits setting the ratio between the motor shaft and driven shaft. Integer values permit preservation of maximum system accuracy with ratios that result in repeating decimals (i.e. 1:3 = 0.333333…). These values are set in the drive’s
Input Revolutions of Load Gear (S-0-0121) and Output Revolutions of Load Gear (S-0-0122) parameters.

The Modulo value (S-0-0103) is indicated as a maximum rotational value in which the motor will turn before resetting the position to zero. The default value when operating in modulo mode is 360. (Modulo mode is set in the drive by setting bit 7 of IDN S-0-0076, Scaling Options for Position Data).

D1.3.9 Overview

Selecting Overview from the Setup menu opens a View Drive # Sercos Parameters dialog box. This dialog box may be used to view and setup all Drive and CLC user accessible parameters.

<table>
<thead>
<tr>
<th>#</th>
<th>Description</th>
<th>Value</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>S-0-0001 NC Cycle Time (Tncyc)</td>
<td>2000</td>
<td>μs</td>
</tr>
<tr>
<td>01</td>
<td>S-0-0002 Sercos Cycle Time (Tscyc)</td>
<td>2000</td>
<td>μs</td>
</tr>
<tr>
<td>01</td>
<td>S-0-0003 Minimum AT Transmit Starting Time (T1min)</td>
<td>120</td>
<td>μs</td>
</tr>
<tr>
<td>01</td>
<td>S-0-0004 Transmit/Receive Transition Time (TATMT)</td>
<td>20</td>
<td>μs</td>
</tr>
<tr>
<td>01</td>
<td>S-0-0005 Minimum Feedback Acquisition Time (T4min)</td>
<td>80</td>
<td>μs</td>
</tr>
<tr>
<td>01</td>
<td>S-0-0006 AT Transmission Starting Time (T1)</td>
<td>120</td>
<td>μs</td>
</tr>
<tr>
<td>01</td>
<td>S-0-0007 Feedback Acquisition Starting Time (T4)</td>
<td>1920</td>
<td>μs</td>
</tr>
</tbody>
</table>

Parameters may be uploaded for display or editing by selecting one of the “Param Source” radio buttons, then clicking the OK button. Choosing Drives or Axes requires the entry of a Drive or Axes Number in the enabled data entry box. Choosing CLC card allows uploading the CLC card system parameters. Selecting Task requires choosing of one of the four CLC tasks from the enabled pull-down menu.

After the selected list has been loaded, you may scroll the list by clicking on the list scroll bar’s up and down arrowheads, or clicking and dragging the scroll button. Clicking and holding (or dragging) the scroll bar button will display the number of the first viewable parameter in the display list in the upper left of the title bar.

Selecting a list entry and clicking the Edit button opens a parameter editing dialog box that allows you to change the parameter value and update the parameter by downloading it to the CLC or DDS-2 drive.

In addition to uploading all the parameters of a set (i.e., all drive or task parameters, etc.) you
may build a custom set of parameters by clicking on the Build Custom… button and opening a Build Custom Display dialog box.

When the Build Custom Display box opens, the parent View … Parameters dialog box remains active. Selecting a list item from the parent box's list, then clicking Add in the build box adds the selected parameter to the custom parameter list. Only one parameter may be added at one time. Selecting one of the parameters in the custom list, then clicking Delete, removes the item from the custom list.

Since the parent dialog box remains active, the parameter list in the parent dialog box may be changed by re-selecting a Param Source and type. Parameters from the newly uploaded list may then be added to the custom list in the build dialog box. In this manner a custom list may be built by selecting any available parameter from any parameter list.

A previously saved custom list may be loaded and used as a source for building another custom list, however, it must be the first list loaded. You cannot load a custom display list once another list has been loaded without closing the Build Custom Display dialog box.

Clicking on Save pops-up a Name Custom Display dialog box permitting entry of an eight character maximum filename. Custom parameter lists are stored in the CLC.INI file in the CLC sub-directory.

Clicking on the Display Custom button opens a Display Custom List dialog box with a list of the currently saved custom parameter lists by name. Selecting a list and clicking on OK or double clicking the list item uploads and displays the parameters in the selected list.
D1.4 Oscilloscope

This utility is used to capture and display run-time data. The capture can be on the CLC card or on a drive that supports this feature. Selected data is acquired on the drive or card, passed to Visual Motion, and displayed on a graphical format. The graphical display and supporting data can be printed, or the data can be saved to a file for additional review.

D1.4.1 File Menu

Used for getting file data, saving data to a file, printing, and exit.

- **Open** - data from user selected input file is loaded into input data list-box.
- **Save** - data from user selected output is loaded into output data list-box.
- **Print Output** - the oscilloscope graph and its related data table is sent to the printer.
- **Exit** - terminates this utility
D1.4.2 Source Menu

Selects the source from which the oscilloscope will gather signal data.
**Drive 1 to n** - lists the drives on the ring that support the oscilloscope feature.
**Card** - when selected the oscilloscope can then read card variables, parameters and

D1.4.3 Timing

Chooses the sample count, sample rate, and a 50% pre-trigger. Sample counts are 50, 100, 200, 300, 400, and 500 for both card and drive captures. Sampling rates for the card are 2, 4, 8, 16, 32, and 64ms, for the drive they are 0.25, 0.5, 1, 5, 10, 50ms.
D1.4.4 Signal Selection

Selects the signals and trigger options.

When a drive is the selected source used to gather data the possible signals include:
- Position Feedback
- Velocity Feedback
- Velocity Deviation (from commanded value)
- Position Deviation (from commanded value)
- Torque Command Value (required to maintain the commanded Velocity/Position)

When the CLC Card is the selected source used to gather data the possible signals include:
- Program Floats (Fx)
- Program Integer (Ix)
- Global Floats (GFx)
Global Integers (GIx)
Axis Parameters of drives on SERCOS ring*
Registers (value)
Registers (bit)

*Axis parameter must be in cyclic telegram, use parameter A-0-0185 and A-0-0195 to add other drive parameters to cyclic data.

For either source the sample acquisition may be user or signal threshold triggered:

For user triggered captures, data acquisition starts as soon as the necessary handshake is complete. This type of start capture is not deterministic.

For threshold triggered the trigger signals are the same as the signals to view. The trigger polarity options are on positive level, on negative level, or on either. Signal threshold is the signal level to trigger on.

D1.4.5 Abort, Upload and Enable Trigger

The Enable Trigger button starts the data acquisition cycle. It sets the trigger condition, waits for data capture, uploads the data, and displays it.
The Upload button uploads data already captured and displays it.
The Abort button terminates a data acquisition cycle.
E1. CLC TRANSFER UTILITY

E1.1 Contents for CLC_XFER Server Help

CLC_XFER is a Windows based Dynamic Data Exchange Server application used to transfer data to/from Indramat’s CLC motion control cards. Communications is via the CLC_DDE server.

Data transfer operations possible
   All( full archive or restore ).
   All Card Parameters( system, task, and axis )
   A Drive’s parameter set( a drive on the ring )
   All Events of a program
   A Program
   All I/O Mapper Equations
   A Cam Table
   All Points( ABS, REL ) of a program
   All Variables( Integers, Floats ) of a program
   All Zones of a program

E1.2 The Communication Servers Main Window

CLC_XFER window provides access to the help system, transaction log, and about window. During a data transfer, the caption will display the name of the files used in the transfer.

When CLC_XFER is in an icon state, its color will indicate its status. **NOTE: This option is not available in Windows95 (works with Windows 3.1, 3.11).**

*Red* indicates inactive and the last transaction terminated in an error, the message “Failed” is written across the top of the icon.

*Yellow* indicates the transfer server is being used. While a transfer is in progress, the percent complete is displayed on the top of the icon and a corresponding bar graph across the center. The name of the file involved in the transfer is displayed below the icon.

*Green* indicates inactive and the last transaction was successful or it was just spawned. The message “Ready” is written across the top of the icon.
E1.3 Dynamic Data Exchange Interface

A windows application, known as a client, can pass information between other applications known as servers using Dynamic Data Exchange (DDE). A client establishes a conversation with a server specifying a Service and a Topic. Once a conversation has been started, a client may request or send information by specifying an item.

The service name for the transfer utility is “CLC_XFER”.

E1.3.1 To transfer data to or from a file

To transfer data use the Dde EXECUTE command. The topic is the communication type like used for the DDE server (i.e. SERIAL_0). The item is an ASCII string containing the direction(R/W), complete path, data descriptor (see above), and program handle(1-10) or drive ID(1-40).

Data transfer operation codes
a - All( full archive or restore )
c - Card Parameters( system, task, and axis )
d - Drive parameters( a drive on the ring )
e - Events
f - Drive cam 1( a drive on the ring )
g - Drive cam 2( a drive on the ring )
h - Programs
i - I/O Mapper Equations
m - Cam Table
p - Points( ABS, REL )
v - Variables( Integers, Floats )
z - Zones

Example items:
R_c:\cl\project\saveset_a_1 ; archive all data to “saveset” directory
R_c:\cl\project\test.pnt_p_1 ; copy points from program handle 1 to file “test.pnt”
R_c:\cl\project\test.csv_m_1 ; copy cam 1 to file “test.csv”
W_c:\cl\project\test.exc_h_6 ; download file “test.exc” to program handle 6
W_c:\cl\project\drive1.prm_d_1 ; download file “drive1.prm” to drive 1
E1.3.2 To get status or the last error message

To inquire about the status, use the Dde REQUEST command. The topic is “SERVER” and the item is one of the following:

TRANSFER|Status ;returns -2 for error, -1 for ready, 0-100 for percent complete
TRANSFER|ErrorText ;returns last error message

E1.3.3 To change a parameter in the server

To change parameters of the transfer utility, use the Dde POKE command. The topic is “SERVER” and the item is one of the following:

TRANSFER|Response_Timeout ;1-900, set message timeout in seconds
TRANSFER|Display_CLC_Errors ;0-1, clears/sets if errors are displayed from server

E1.4 Activity Log

A record of data transfer transactions is saved to a read only file “XFERLOG.TXT” in the working directory. The file size is limited to 256 lines, the oldest record is deleted to make room for a new entry. Each entry consist of date, time, dde topic, dde item, and status. For failed entries, an indented message discribing the failure will be displayed on the following line.

Activity log example:

06/27/95 09:51:53 - SERIAL_0 - W_c:/clc_1_3/project/test.iom_i_1 - OK
06/27/95 09:47:41 - SERIAL_0 - R_c:/clc_1_3/project/test.zon_z_1 - Failed
!No zones in program to save.
06/27/95 09:47:12 - SERIAL_0 - R_c:/clc_1_3/project/test.var_v_1 - OK
06/27/95 09:46:20 - SERIAL_0 - R_c:/clc_1_3/project/test.var_v_1 - OK
INDEX

!01 SERCOS Error Code#xxxx (xxxx=Error code), B-43
!02 Invalid Parameter Number, B-43
!03 Data is Read Only, B-43
!04 Write Protected in this mode/phase, B-43
!05 Greater than maximum value, B-43
!06 Less than minimum value, B-43
!07 Data is Invalid, B-43
!08 Drive was not found, B-43
!09 Drive not ready for communication, B-43
!10 Drive is not responding, B-43
!11 Service channel is not open., B-44
!12 Invalid Command Class, B-44
!13 Checksum Error: xx (xx= checksum that CLC calculated), B-44
!14 Invalid Command Subclass, B-44
!15 Invalid Parameter Set, B-44
!16 List already in progress, B-44
!17 Invalid Sequence Number, B-44
!18 List has not started, B-44
!19 List is finished, B-44
!20 Parameter is a List, B-44
!21 Parameter is not a List, B-44
!22 Invalid Variable Number, B-44
!23 Insufficient program space, B-45
!24 Maximum number of files exceeded, B-45
!25 Invalid program header, B-45
!26 Checksum Error in Program, B-45
!27 Invalid Program Handle, B-45
!28 Function not Implemented, B-45
!29 File not Found, B-45
!30 Invalid I/O Register, B-45
!31 Invalid Table Index, B-45
!32 Communication Error 32, B-45
!33 Invalid Data Format, B-45
!34 Active program can't be deleted, B-46
!35 Parameter mode is required, B-46
!36 Invalid Event Number, B-46
!37 Invalid Event Function, B-46
!38 Program file version mismatch, B-46
!39 Can't activate while program running, B-46
!40 No programs are active, B-46
!41 System Error: pSOS #XXXX, B-46
!42 Mapper String DD: invalid operator, B-46
!43 Mapper String DD: too many operations, B-46
!44 Mapper String DD: invalid register, B-46
!45 Mapper String DD: invalid bit or mask, B-46
!46 Mapper String DD: register is read-only, B-47
!47 Invalid Unit Number, B-47
!48 VME Bus Error, B-47
!49 VME Communication Handshake Error (D), B-47
!50 Invalid Download Block, B-47
!51 Unit D: Invalid VME Base Address Page, B-47
!52 Axis Disabled, B-47
!53 Waiting for service channel, B-47
!54 List or String is too short, B-47
!55 List or String is too long, B-47
!56 PC Communication Handshake Error, B-48
!57 Mapper String D: string space is full, B-48
!58 Cannot store cam already active for axis D, B-48
!59 SERCOS handshake/busy timeout, B-48
!60 Executable program is too large (ddK), B-48
!61 System Memory Allocation Error, B-48
!62 Cam X data is < 0 or greater than 360, B-48
!63 X-Column does not start at 0 or end at 360, B-48
!64 Not supported in user prog file version 1.1, B-48
!65 Sequencer invalid sequence (D), B-48
!66 Sequencer invalid step (D), B-49
!67 Invalid function number (D), B-49
!68 Function D not accessible in a step, B-49
!69 Too many functions are used (D), B-49
!70 Maximum steps per sequence exceeded (D), B-49
!71 Maximum functions per step exceeded (D), B-49
!72 Program does not include a PLS, B-49
!73 Invalid ABS or REL point index (D), B-49
!74 Error in command execution, B-49
!75 Comm. port buffer overflow, B-49
!76 Invalid Block, B-50
!77 Can't save sequencer while it is running, B-50
!78 Service channel in use, B-50
!79 PID block number does not exist, B-50

001 Initializing System, 8-33
002 Parameter Mode, 8-33
003 Looking for drives, 8-33
004 System is Ready, 8-33
005 Manual Mode, 8-33
006 Automatic Mode: ABCD, 8-33
007 Program Running: ABCD, 8-33
008 Single-Stepping: ABCD, 8-33
009 Select Parameter Mode to Continue, 8-33
010 Breakpoint Reached: ABCD, 8-33

2
201 Invalid jog type or axis selected, 8-34
201 Warning Messages
  Invalid jog type or axis selected, 8-34
202 Drive D is not ready, 8-34
203 Power Lost During Program, 8-34
204 SERCOS Ring was disconnected, 8-34
205 Parameter transfer warning in Task A, 8-34
206 Battery is low: replace it soon, 8-34
207 Axis D position limit reached, 8-34
400 Emergency Stop, 8-34
401 SERCOS Controller Error: DD, 8-34
403 System Error, 8-35
404 Invalid Switch into Phase D, 8-35
405 Phase D: Drive did not respond, 8-35
406 System Error, 8-35
407 Drive D Phase 3 Switch Error, 8-36
408 SERCOS Controller is in test mode, 8-36
409 SERCOS Disconnect Error, 8-36
410 System Error, 8-36
411 Drive D Phase 4 Switch Error, 8-36
412 No drives were found on ring, 8-36
413 I-O board was not found, 8-37
414 Parameters were lost, 8-37
415 Drive D was not found, 8-37
416 Invalid Instruction at XXXX, 8-37
417 SYSTEM ERROR: pSOS #XXXX, 8-37
418 No program is active, 8-37
419 Invalid Program File, 8-37
420 Drive D Shutdown Error, 8-37
421 User Program Stack Overflow, 8-38
422 Parameter transfer error in Task A, 8-38
423 Unimplemented Instruction, 8-38
424 System Error, 8-38
425 Instruction Error: see Task A diag., 8-38
426 Drive D is not ready, 8-38
427 Calc: invalid table index D, 8-38
428 Calc: division by zero, 8-39
429 Calc: too many operands, 8-39
430 Calc instruction: invalid operator, 8-39
431 Calc error: see Task A diag., 8-39
432 Calc: too many nested expressions, 8-39
433 Setup instruction outside of a task, 8-39
434 Axis D configured more than once, 8-39
435 Axis D not associated with a task, 8-39
436 General Compiler Error: XXXX, 8-39
437 Axis D not controlled by this task, 8-40
438 Invalid Axis Selected: D, 8-40
439 Invalid Motion Type: D, 8-40
440 I-O Transfer Error: see task diag., 8-40
441 DMA error while reading from local RAM, 8-40
442 DMA error while reading from VME address, 8-40
443 DMA error while writing to local RAM, 8-40
444 DMA error while writing to VME address, 8-40
445 DMA Access Timeout Error, 8-40
446 DMA Timeout Error, 8-40
447 VME SYSFAIL Detected, 8-40
448 VME Communication Handshake Error (D), 8-40
449 VME Bus Error, 8-40
450 Event D: invalid event type, 8-41
451 Invalid event number D, 8-41
452 More than D event timers armed, 8-41
453 Homing param. transfer error: D, 8-41
454 Axis D homing not complete, 8-41
455 Invalid VME Data Transfer Class, 8-41
456 Invalid VME Address, 8-41
457 Table Bounds Error During VME Read, 8-41
458 Table Bounds Error During VME Write, 8-41
459 Axis D target position out of bounds, 8-41
460 Invalid program D from binary inputs, 8-42
461 System Error, 8-42
462 System Error, 8-42
463 Ratio command: invalid ratio, 8-42
464 Can't activate while program running, 8-42
466 Drive D: scaling type not supported, 8-42
467 Invalid ELS Master Option, 8-42
468 ELS adjustment out of bounds, 8-42
469 Axis D accel <= 0 or > maximum, 8-43
470 Axis D velocity > maximum, 8-43
471 Invalid VME Base Address Page: 0xXXXX, 8-43
472 VME Event Trigger Rejected, 8-43
473 VME Event Trigger For Unit D Failed, 8-43
474 Drive D cyclic data size too large, 8-43
475 Axis D capture already configured, 8-43
476 Axis D: Real Time Bit Setup Error, 8-43
477 Axis D: probe edge not configured, 8-43
478 Calc: operand out of range, 8-43
479 Drive D: too many cyclic data elements, 8-43
480 SERCOS Error: MDT is too large, 8-44
481 Event D is already armed, 8-44
482 Checksum Error in Program, 8-44
483 Parameter Init. Error: see Task A diag, 8-44
484 CLC SYSTEM ERROR, 8-44
485 SERCOS I-O: too many registers configured, 8-44
486 SERCOS Device D is not a drive, 8-44
487 Cam D is invalid or not stored, 8-45
488 Cam Error: See Task A diag., 8-45
489 More than D cam axes selected, 8-45
490 System Memory Allocation Error, 8-45
491 PC Communication Handshake Error, 8-45
492 Programs were lost, 8-45
493 Data was restored from Flash, 8-45
494 Sequencer init. error: see task T diag, 8-45
495 Sequencer error: see task T diag, 8-45
496 Can't Execute this Instruction from an Event, 8-46
497 Limit switch config. error, see ext. diag, 8-46
498 Drive D Shutdown Warning, 8-46
499 Axis number D not supported in this version, 8-46
500 Axis D is not referenced, 8-46
501 Drive D communications error, 8-46
502 ELS and cams not supported in this version, 8-46
503 Executing empty block #D, 8-47
504 Communication Timeout, 8-47
505 Axis D is not configured, 8-47
506 I-O Mapper initialization error, 8-47
507 Option Card Power Supply Error, 8-47

A

absolute point table, 5-22
Absolute Synchronization, C-4
ACAM, C-14
Accel, 6-5
acceleration profiles, 1-7, C-5
acceleration/deceleration, 5-21
Activating a Program (PA), B-29
Add Event Function, 4-11
Add Subroutine, 4-11
addressing
    indirect, 5-18
Aligned Status Bit, C-23
Analog Outputs, D-3
Application (Rotary Cutoff), C-30
Archive, 4-4
arrays, 5-17
AT Modem, A-12
Available Program Memory, 3-24
Axis, 6-6
Axis Fast Jog Velocity, 3-23
Axis Feedback Capture (Registration), 3-102
Axis Jog Control Registers, 2-18
Axis Large Increment, 3-23
Axis Motion Type, 3-78
Axis Parameters
    Axis Feedback Capture (Registration), 3-102
        A-0-0170 Feedback Capture Status, 3-102
        A-0-0171 Probe 1 Positive Captured Position, 3-102
        A-0-0172 Probe 1 Negative Captured Position, 3-102
        A-0-0173 Probe 2 Positive Captured Position, 3-102
        A-0-0174 Probe 2 Negative Captured Position, 3-102
Axis Setup, 3-77
    A-0-0003 Axis Motion Type, 3-78
    A-0-0004 Positioning Options, 3-79
    A-0-0005 Linear Position Units, 3-84
    A-0-0011 Drive I/O Card 1 Setup, 3-87
    A-0-0014 Drive I/O Card 2 Setup, 3-87
    A-0-0017 Drive I/O Card 3 Setup, 3-88
    A-0-0020 Maximum Velocity, 3-88
    A-0-0021 Maximum Acceleration, 3-88
    A-0-0022 Maximum Deceleration, 3-89
    A-0-0025 Maximum Jog Increment, 3-89
    A-0-0026 Maximum Jog Velocity, 3-89
    A-0-0030 Master Axis for Ratio Function, 3-90
Axis Status, 3-93
    A-0-0100 Target Position, 3-93
    A-0-0101 Commanded Position, 3-93
    A-0-0102 Feedback Position, 3-93
    A-0-0110 Programmed Velocity, 3-94
    A-0-0111 Commanded Velocity, 3-94
    A-0-0112 Feedback Velocity, 3-94
    A-0-0120 Programmed Acceleration, 3-95
    A-0-0131 SERCOS Control Word, 3-95
    A-0-0132 SERCOS Status Word, 3-95
    A-0-0140 Single-Axis Status Word, 3-96
    A-0-0141 Torque Mode Commanded Torque, 3-96
    A-0-0142 Torque Feedback (cyclic), 3-96
    A-0-0145 Current Motion Type, 3-96
Electronic Line Shaft, 3-97
    A-0-0151 ELS Programmed Phase Offset, 3-97
    A-0-0153 ELS Phase Adjust Velocity, 3-97
    A-0-0157 ELS Commanded Phase Offset, 3-98
Axis Parameters:
    Axis Parameters, 3-77
Axis Setup, 3-77
Axis Slow Jog Velocity, 3-23
Axis Small Increment, 3-23
Axis Status, 3-93
Axis Status Registers, 2-27
AXIS/EVENT, 7-7
AXIS/HOME, 7-8
AXIS/INITIALIZE, 7-9
AXIS/MOVE (Single Axis, Non-Coordinated), 7-11
AXIS/RATIO, 7-13
AXIS/SPINDLE (Continuous Velocity Mode), 7-15
AXIS/START, 7-17
AXIS/STOP, 7-19
AXIS/WAIT (Axis Wait For In-Position), 7-21
AxisEvt, 6-11

B
Backspaces and White spaces, B-4
Baud Rate, A-7
bipolar velocity limit, D-7
Bit Labels, 4-14
blend segments, 5-20
Branch, 6-12
Breakpoint Control, 4-23
Building Cam Tables, C-10
Bus Arbitration Mode, 3-36
Bus Release Mode, 3-37

C
Calc, 6-14
Call, 7-23
Cam
    Number, 6-18
    Cam Alignment, C-22
    Cam Applications, C-9
    CAM Builder, 4-24
    Cam Commands, C-9
    Cam Equation, C-16
    Cam synchronization, 1-7, C-5
    CAM/ACTIVATE, 7-23
    CAM/ADJUST, 7-25
    CAM/BUILD, 7-26
    CAM/STATUS, 7-28
    Camming, 6-16
    CAPTURE/ENABLE, 7-29
    CAPTURE/SETUP, 7-31
    Card Selection, 4-16
    Checksum, B-3
    Circle, 6-20
    CLC
        Back Plane Relay Time-out, A-5
        Operating System, 1-4
        Overview, 1-3

REV. F, 10/96 INDEX
Response Time-out, A-5
Server Configuration, A-5
Status and Diagnostics, 8-31
Status Display, A-5
System Architecture, 1-3
CLC Compiler Base Code, 8-2
CLC Cycle Control Considerations, 2-13
CLC DDE Server, A-4
CLC Input/Output Systems
  General Description, 2-1
CLC Instructions, 7-4
CLC Registers, 2-7
CLC Reserved Register Tables, 2-32
Clear Current Task, 4-10
command execution, 5-1
Commanded Position, 3-93
Commanded Velocity, 3-94
Communication Error Codes and Messages, B-43
  !01 SERCOS Error Code#xxxx (xxxx=Error code), B-43
  !02 Invalid Parameter Number, B-43
  !03 Data is Read Only, B-43
  !04 Write Protected in this mode/phase, B-43
  !05 Greater than maximum value, B-43
  !06 Less than minimum value, B-43
  !07 Data is Invalid, B-43
  !08 Drive was not found, B-43
  !09 Drive not ready for communication, B-43
  !10 Drive is not responding, B-43
  !11 Service channel is not open., B-44
  !12 Invalid Command Class, B-44
  !13 Checksum Error: xx (xx= checksum that CLC calculated), B-44
  !14 Invalid Command Subclass, B-44
  !15 Invalid Parameter Set, B-44
  !16 List already in progress, B-44
  !17 Invalid Sequence Number, B-44
  !18 List has not started, B-44
  !19 List is finished, B-44
  !20 Parameter is a List, B-44
  !21 Parameter is not a List, B-44
  !22 Invalid Variable Number, B-44
  !23 Insufficient program space, B-45
  !24 Maximum number of files exceeded, B-45
  !25 Invalid program header, B-45
  !26 Checksum Error in Program, B-45
  !27 Invalid Program Handle, B-45
  !28 Function not Implemented, B-45
  !29 File not Found, B-45
  !30 Invalid I/O Register, B-45
  !31 Invalid Table Index, B-45
  !32 Communication Error 32, B-45
  !33 Invalid Data Format, B-45
  !34 Active program can’t be deleted, B-46
  !35 Parameter mode is required, B-46
  !36 Invalid Event Number, B-46
  !37 Invalid Event Function, B-46
  !38 Program file version mismatch, B-46
  !39 Can’t activate while program running, B-46
  !40 No programs are active, B-46
  !41 System Error: pSOS #XXXX, B-46
  !42 Mapper String D: invalid operator, B-46
  !43 Mapper String D: too many operations, B-46
  !44 Mapper String D: invalid register, B-46
  !45 Mapper String D: invalid bit or mask, B-46
  !46 Mapper String D: register is read-only, B-47
  !47 Invalid Unit Number, B-47
  !48 VME Bus Error, B-47
  !49 VME Communication Handshake Error (D), B-47
  !50 Invalid Download Block, B-47
  !51 Unit D: Invalid VME Base Address Page, B-47
  !52 Axis Disabled, B-47
  !53 Waiting for service channel, B-47
  !54 List or String is too short, B-47
  !55 List or String is too long, B-47
  !56 PC Communication Handshake Error, B-48
  !57 Mapper String D
     string space is full, B-48
  !58 Cannot store cam
     already active for axis D, B-48
  !59 SERCOS handshake/busy timeout, B-48
  !60 Executable program is too large (ddK), B-48
  !61 System Memory Allocation Error, B-48
  !62 X-Column does not start at 0 or end at 360, B-48
  !63 X-Column does not start at 0 or end at 360, B-48
  !64 Not supported in user prog file version 1.1, B-48
  !65 Sequencer
     invalid sequence (D), B-48
  !66 Sequencer
     invalid step (D), B-49
  !67 Invalid function number (D), B-49
  !68 Function D not accessible in a step, B-49
  !69 Too many functions are used (D), B-49
  !70 Maximum steps per sequence exceeded (D), B-49
  !71 Maximum functions per step exceeded (D), B-49
  !72 Program does not include a PLS, B-49
  !73 Invalid ABS or REL point index (D), B-49
Communication Errors, B-3
Communication Protocol, B-1
Configuration, 4-17
Connecting Icons, 6-2
constants, 5-13
Conversion functions, 5-16
Coordinated Axis, 3-78
Coordinated Jogging, 2-15
Coordinated Motion, 3-64, 3-85, 4-18
Coordinated X Axis, 3-64
Coordinated Y Axis, 3-65
Coordinated Z Axis, 3-65
Current Instruction, 3-73
Current Instruction Pointer, 3-72
Current Motion Type, 3-96
Current X Position, 3-70
Current Y Position, 3-71
Current Z Position, 3-71
Cyclic Data, 3-81
<table>
<thead>
<tr>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>Daisy-Chained, 6-23</td>
<td>Edit Menu</td>
</tr>
<tr>
<td>data format errors, 6-60</td>
<td>Labels</td>
</tr>
<tr>
<td>Data Menu, 4-32</td>
<td>User Labels, 4-12</td>
</tr>
<tr>
<td>data type</td>
<td></td>
</tr>
<tr>
<td>type conversion, 6-60</td>
<td>Electronic Line Shaft, 3-32, 3-97</td>
</tr>
<tr>
<td>data types</td>
<td>ELS, 6-23</td>
</tr>
<tr>
<td>conversion, 5-15</td>
<td>ELS Commanded Phase Offset, 3-98</td>
</tr>
<tr>
<td>floating point, 5-13</td>
<td>ELS Master Axis, 3-32</td>
</tr>
<tr>
<td>integers, 5-13</td>
<td>ELS Master Current Position, 3-34</td>
</tr>
<tr>
<td>DATA/SIZE (Configure Memory Allocations), 7-33</td>
<td>ELS Master Current Velocity, 3-34</td>
</tr>
<tr>
<td></td>
<td>ELS Master Task Assignment, 3-32</td>
</tr>
<tr>
<td>DDE, 1-4, A-2, A-6</td>
<td>ELS Master Type, 3-32</td>
</tr>
<tr>
<td>Conversations, A-10</td>
<td>ELS Masters, C-1</td>
</tr>
<tr>
<td>Dde Server, A-1</td>
<td>ELS Phase Adjust Velocity, 3-97</td>
</tr>
<tr>
<td>DDS Setup, 3-106</td>
<td>ELS Programmed Phase Offset, 3-97</td>
</tr>
<tr>
<td>Decel, 6-22</td>
<td>ELS Secondary Mode, 3-81</td>
</tr>
<tr>
<td>DEFINE, 7-2</td>
<td>ELS Slave, 3-78</td>
</tr>
<tr>
<td>DEFINE (Define Name For Variable), 7-36</td>
<td>ELS Slaves, C-2</td>
</tr>
<tr>
<td>DELAY (Suspend Task Execution), 7-38</td>
<td>ELS Synchronization Mode, 3-80</td>
</tr>
<tr>
<td>Diagnostic Code, 2-14</td>
<td>ELS/ADJUST (Adjust ELS Axis), 7-39</td>
</tr>
<tr>
<td>Diagnostic Codes, 8-32</td>
<td>ELS/DECEL (Set Master Decel), 7-41</td>
</tr>
<tr>
<td>Diagnostic Log, 4-59</td>
<td>ELS/INIT (Initialize ELS Axes), 7-42</td>
</tr>
<tr>
<td>Diagnostic Messages, 8-32</td>
<td>ELS/MODE (Set ELS Axis Mode), 7-45</td>
</tr>
<tr>
<td>Directives, 7-1</td>
<td>ELS/STOP, 7-46</td>
</tr>
<tr>
<td>DEFINE, 7-2</td>
<td>ELSAdj, 6-26</td>
</tr>
<tr>
<td>EQU (Equate), 7-1</td>
<td>ELSMode, 6-27</td>
</tr>
<tr>
<td>EVENT/START &amp; EVENT/END, 7-3</td>
<td>End of Message, B-4</td>
</tr>
<tr>
<td>TASK/START &amp; TASK/END, 7-3</td>
<td>EQU (Equate), 7-1</td>
</tr>
<tr>
<td>Download Block Size, B-24</td>
<td>Erase All Forcing Masks (RE), B-39</td>
</tr>
<tr>
<td>Drive and CLC Parameters and Subclasses, B-9</td>
<td>Erasing (Deleting) a Program (PE), B-27</td>
</tr>
<tr>
<td>Drive Cam Alignment, C-26</td>
<td>Error Handling, A-6</td>
</tr>
<tr>
<td>Drive Cam Equation, C-24</td>
<td>Error Reaction Mode, 3-16</td>
</tr>
<tr>
<td>Drive Cams, C-24</td>
<td>Event, 6-28</td>
</tr>
<tr>
<td>Drive Direction, D-4</td>
<td>Event Functions, 4-15</td>
</tr>
<tr>
<td>Drive Disable Method, 3-82</td>
<td>Event Table</td>
</tr>
<tr>
<td>Drive Homing, D-8</td>
<td>Event Table Data, B-22</td>
</tr>
<tr>
<td>Drive I/O Card 1 Setup, 3-87</td>
<td>Event Table Data, Row Format, B-23</td>
</tr>
<tr>
<td>Drive I/O Card 2 Setup, 3-87</td>
<td>Event Tables, 5-9, B-21</td>
</tr>
<tr>
<td>Drive I/O Card 3 Setup, 3-88</td>
<td>EVENT/DONE (Signal Event Completed), 7-47</td>
</tr>
<tr>
<td>Drive Limits, D-7</td>
<td>EVENT/END (Mark End of Event), 7-49</td>
</tr>
<tr>
<td>Drive Monitoring, D-5</td>
<td>EVENT/START &amp; EVENT/END, 7-3</td>
</tr>
<tr>
<td>Drive Name, D-5</td>
<td>EVENT/START (Start of Event function), 7-50</td>
</tr>
<tr>
<td>Drive Parameter</td>
<td>EVENT/TRIGGER (Trigger a Task Event), 7-51</td>
</tr>
<tr>
<td>Drive Status, 3-106</td>
<td>EVENT/WAIT (Pause Task for Event Done Signal), 7-53</td>
</tr>
<tr>
<td>Required DDS Setup, 3-106</td>
<td></td>
</tr>
<tr>
<td>Drive Parameter Editor, D-1</td>
<td>events, 1-4, 4-32, 5-2</td>
</tr>
<tr>
<td>Drive Parameters, 3-105</td>
<td>Distance based, 5-3</td>
</tr>
<tr>
<td>Drive Reference, D-7, D-8</td>
<td>feedback capture, 5-8</td>
</tr>
<tr>
<td>Drive Status, 3-106</td>
<td>interrupt input events, 5-6</td>
</tr>
<tr>
<td>Drive Tuning, D-6</td>
<td>repeating axis position, 5-5</td>
</tr>
<tr>
<td>Drives, 4-17, 4-59</td>
<td>Time-based, 5-2</td>
</tr>
<tr>
<td>Drives on Ring, 4-60</td>
<td>VME, 5-7</td>
</tr>
<tr>
<td>Dynamic Data Exchange, A-1</td>
<td>broadcast, 5-7</td>
</tr>
<tr>
<td>Dynamic synchronization, C-6</td>
<td>short address, 5-7</td>
</tr>
<tr>
<td></td>
<td>Executing a Download (PD), B-25</td>
</tr>
<tr>
<td></td>
<td>Executing an Upload (PD), B-27</td>
</tr>
<tr>
<td></td>
<td>Exponential functions, 5-16</td>
</tr>
<tr>
<td>Export User Label File, 4-14</td>
<td></td>
</tr>
</tbody>
</table>
expressions, 5-15
  evaluation, 5-15
operators, 5-16
  Precedence, 5-15
type conversion, 5-15
Extended Diagnostic, 3-30

F
Fatal System Errors, 8-32
Feedback Capture Status, 3-102
Feedback Position, 3-93
Feedback Velocity, 3-94
Find, Find Next, 4-10
Finish, 6-29
First Pass Errors, 8-22
Follow axis feedback, 7-43
Format of Data Sent to the CLC, B-4
  Function arguments, 5-10, 6-57
FUNCTION/ARG, 7-55
FUNCTION/END, 7-56
FUNCTION/START, 7-57

G
gemetry segments, 5-20
Go, 6-29
GOSUB (Go To Subroutine), 7-58
GOTO (Go To Mark), 7-59

H
Help, 4-63
Home, 6-30
Host Serial Port Setup, 3-14

I
I/O, 6-31
I/O Binary Forcing State (RS), B-39
I/O Bit Forcing, 2-6
I/O Device Direction, 3-15
I/O Device Setup, 3-15
I/O Device VME Start Address, 3-16
I/O Forcing Selection (RF), B-38
I/O Forcing State Change (RC), B-38
I/O Mapper, 4-33
I/O Mapper Operators, 2-5
I/O Mapper String Format, 2-4
I/O Register Access (RB), (RX), (RD), B-36
I/O Register Read, B-37
I/O Register Write, B-37
I/O Setup, 4-19
Icon Language Warnings and Error Messages, 8-19
icon palette, 6-1
ICON PROGRAMMING, 6-1
Icons
  Accel, 6-5

Axis, 6-6
AxisEvt, 6-11
Branch, 6-12
Calc, 6-14
Cam, 6-16
Cam Build, 6-18
Circle, 6-20
Connecting Icons, 6-2
Decel, 6-22
ELS, 6-23
ELSAadj, 6-26
ELSMode, 6-27
Event, 6-28
Finish, 6-29
Go, 6-29
Home, 6-30
I/O, 6-31
Join, 6-32
Joint, 6-33
Line, 6-34
List of CLC Icons, 6-4
Move, 6-35
Msg, 6-36
Param, 6-37
Path, 6-39, 6-40
PLS, 6-43
Position, 6-44
Pmbit, 6-45
Pmbint, 6-46
Probe, 6-47
ProbeEvt, 6-48
Ratio, 6-49
Reg, 6-50
RobotOrg, 6-51
RobotTol, 6-51
Scissor, 6-52
Sequencer, 6-52
Size:, 6-54
Start, 6-55
Stop, 6-56
Sub, 6-57
Veloc, 6-58
VME
  Address, 6-60
  Address Bus Width, 6-59
  Byte Order, 6-60
  Data Bus Width, 6-60
  Format, 6-60
  Local data, 6-60
  Transfer count, 6-60
VME:, 6-59
VMEEvt, 6-61
Wait, 6-62
  Axis in Position, 6-62
  Coordinated State, 6-63
IF (If-Else-Endif Conditional Branch), 7-60
Import User Label File, 4-14
Initializing a Download (PW), B-24
Initializing an Upload (PR), B-26
Input/Output Registers, B-36
Instruction Format, 7-4
Instruction Pointer at Error, 3-73
Integral Action Time, D-6
Item Name, A-3

J

jerk limiting, 5-21
Jogging, 4-28
Jogging and Display, 3-22
Join, 6-32
Joint, 6-33

K

KINEMATIC (Use a Kinematic Definition for a Task), 7-64, 7-65
Kinematic Values, 3-68

L

Labels
Bit, 4-14
Export, 4-14
Import, 4-14
Register, 4-13
User, 4-11
Language Selection, 3-13
Last Active Event Number, 3-76
Latch, 2-1, 2-29
Line, 6-34
Linear Position Units, 3-84
List CLC Resident Programs (PH), B-28
List of All Parameters, 3-51, 3-76, 3-104
List of Event Function Marks (PF), B-30
List of Required Parameters, 3-51, 3-76, 3-104
List Variable Labels (PV), B-31
Local variables, 5-10, 6-58
Look Ahead Distance, 3-66

M

Master Axis for Ratio Function, 3-90
Maximum Acceleration, 3-88
Maximum Deceleration, 3-89
Maximum Jog Increment, 3-67, 3-89
Maximum Jog Velocity, 3-67, 3-89
Maximum Model Deviation, D-5
Maximum Path Acceleration, 3-65
Maximum Path Deceleration, 3-66
Maximum Path Speed, 3-65
Maximum Velocity, 3-88
Mechanical, D-10
MESSAGE/DIAG (Task Diagnostic Message Definition), 7-66
MESSAGE/STATUS (Task Status Message Definition), 7-67

Motion, 1-5
circled, 1-6
circular interpolation, 1-6
constant speed, 1-6
kinematics, 1-6
linear interpolation, 1-6
ELS, 1-7, C-1
phase synchronous mode, 1-7
velocity synchronous mode, 1-7
non-coordinated, 1-5
ratioed axes, 1-5
single axis, 1-5
velocity mode, 1-5
Move, 6-35
MOVE/CIRCLE (Coordinated Move with Circular Interpolation), 7-68
MOVE/JOINT (Coordinated Move Joint Point to Point), 7-70
MOVE/LINE (Coordinated Move with Straight Line Interpolation), 7-71
Msg, 6-36
multi-tasking, 5-1
Multi-turn Encoder, D-9

N

names
symbolic, 4-11
Numeric Data Formats, B-4

O

Operating Mode, 3-29
operators
logical, 5-16
mathematical, 5-10, 5-16, 6-15
Oscilloscope, 4-31, D-13
Overview, 4-20

P

P Data, B-16
palette, 6-1
Parallel I/O Device, 3-15
Param, 6-37
Parameter Data Subclass, B-9
Parameter Lists, 3-51, 3-104, B-11
Listing a Parameter, B-11, B-13
Parameter Transfer Commands, 3-3
PARAMETER/BIT (Initialize Parameter Bit), 7-73
PARAMETER/GET (Load Parameter to a Variable), 7-75
PARAMETER/INIT (Initialize a Parameter), 7-77
PARAMETER/SET (Set a Parameter), 7-78
PARAMETERS, 3-1
Axis Feedback Capture (Registration), 3-12
Axis Parameter Lists, 3-12
Axis Parameters - Electronic Line Shaft, 3-12
Axis Setup, 3-11
Axis Status, 3-11
Second Pass Compiler Errors, 8-23
Segment Status, 3-70
SEQ/LIST, 7-98
SEQ/STEP, 7-99
Sequencer, 4-54, 5-11, 6-52, 7-97
List, 5-11
Single Stepping, 5-12
Step List, 5-11
SERCOS, 1-3
Data, 3-103
SERCOS Communication Phase, 3-29
SERCOS Control Word, 3-95
SERCOS Status Word, 3-95
Serial Communications, A-7
Serial Event, A-7
Serial Port, A-7
SERVER
  Topic Name, A-13
  Service Name, A-2
Set Current I/O State with Mask (RM), B-37
  Setup Menu, 4-16
Show Program Flow, 4-32
Shutdown Messages, 8-31, 8-34
  400 Emergency Stop, 8-34
  401 SERCOS Controller Error: DD, 8-34
  403 System Error, 8-35
  404 Invalid Switch into Phase D, 8-35
  405 Phase D Drive did not respond, 8-35
  406 System Error, 8-35
  407 Drive D Phase 3 Switch Error, 8-36
  408 SERCOS Controller is in test mode, 8-36
  409 SERCOS Disconnect Error, 8-36
  410 System Error, 8-36
  411 Drive D Phase 4 Switch Error, 8-36
  412 No drives were found on ring, 8-36
  413 I-O board was not found, 8-37
  414 Parameters were lost, 8-37
  415 Drive D was not found, 8-37
  416 Invalid Instruction at XXXX, 8-37
  417 SYSTEM ERROR: pSOS #XXXX, 8-37
  418 No program is active, 8-37
  419 Invalid Program File, 8-37
  420 Drive D Shutdown Error, 8-37
  421 User Program Stack Overflow, 8-38
  422 Parameter transfer error in Task A, 8-38
  423 Unimplemented Instruction, 8-38
  424 System Error, 8-38
  425 Instruction Error: see Task A diag., 8-38
  426 Drive D is not ready, 8-38
  427 Calc: invalid table index D, 8-38
  428 Calc: division by zero, 8-39
  429 Calc: too many operands, 8-39
  430 Calc instruction: invalid operator, 8-39
  431 Calc error: see Task A diag., 8-39
  432 Calc: too many nested expressions, 8-39
  433 Setup instruction outside of a task, 8-39
  434 Axis D configured more than once, 8-39
  435 Axis D not associated with a task, 8-39
  436 General Compiler Error: XXXX, 8-39
  437 Axis D not controlled by this task, 8-40
  438 Invalid Axis Selected: D, 8-40
  439 Invalid Motion Type: D, 8-40
  440 I-O Transfer Error: see task diag., 8-40
  441 DMA error while reading from local RAM, 8-40
  442 DMA error while reading from VME address, 8-40
  443 DMA error while writing to local RAM, 8-40
  444 DMA error while writing to VME address, 8-40
  445 DMA Access Timeout Error, 8-40
  446 DMA Timeout Error, 8-40
  447 VME SYSFAIL Detected, 8-40
  448 VME Communication Handshake Error (D), 8-40
  449 VME Bus Error, 8-40
  450 Event D: invalid event type, 8-41
  451 Invalid event number D, 8-41
  452 More than D event timers armed, 8-41
  453 Homing param. transfer error: D, 8-41
  454 Axis D homing not complete, 8-41
  455 Invalid VME Data Transfer Class, 8-41
  456 Invalid VME Address, 8-41
  457 Table Bounds Error During VME Read, 8-41
  458 Table Bounds Error During VME Write, 8-41
  459 Axis D target position out of bounds, 8-41
  460 Invalid program D from binary inputs, 8-42
  461 System Error, 8-42
  462 System Error, 8-42
  463 Ratio command: invalid ratio, 8-42
  464 Can't activate while program running, 8-42
  466 Drive D: scaling type not supported, 8-42
  467 Invalid ELS Master Option, 8-42
  468 ELS adjustment out of bounds, 8-42
  469 Axis D accel <= 0 or > maximum, 8-43
  470 Axis D velocity > maximum, 8-43
  471 Invalid VME Base Address Page: 0xXXXX, 8-43
  472 VME Event Trigger Rejected, 8-43
  473 VME Event Trigger For Unit D Failed, 8-43
  474 Drive D cyclic data size too large, 8-43
  475 Axis D capture already configured, 8-43
  476 Axis D: Real Time Bit Setup Error, 8-43
  477 Axis D: probe edge not configured, 8-43
  478 Calc: operand out of range, 8-43
  479 Drive D: too many cyclic data elements, 8-43
  480 SERCOS Error: MDT is too large, 8-44
  481 Event D is already armed, 8-44
  482 Checksum Error in Program, 8-44
  483 Parameter Init. Error: see Task A diag., 8-44
  484 CLC SYSTEM ERROR, 8-44
  485 SERCOS I-O: too many registers configured, 8-44
  486 SERCOS Device D is not a drive, 8-44
  487 Cam D is invalid or not stored, 8-45
  488 Cam Error: See Task A diag., 8-45
  489 More than D cam axes selected, 8-45
  490 System Memory Allocation Error, 8-45
  491 PC Communication Handshake Error, 8-45
  492 Programs were lost, 8-45
  493 Data was restored from Flash, 8-45
  494 Sequencer init. error: see task T diag., 8-45
495 Sequencer error: see task T diag, 8-45
496 Can't Execute this Instruction from an Event, 8-46
497 Limit switch config. error, see ext. diag, 8-46
498 Drive D Shutdown Warning, 8-46
499 Axis number D not supported in this version, 8-46
500 Axis D is not referenced, 8-46
501 Drive D communications error, 8-46
502 ELS and cams not supported in this version, 8-46

Single Axis, 3-78
Single-Axis Status Word, 3-96
Size, 6-54
Slave Secondary Mode, C-7
Smoothing Time Constant, D-6
speed, 5-20
Spline, 6-18
Start, 6-55

Status Messages, 8-31, 8-33
  001 Initializing System, 8-33
  002 Parameter Mode, 8-33
  003 Looking for drives, 8-33
  004 System is Ready, 8-33
  005 Manual Mode, 8-33
  006 Automatic Mode: ABCD, 8-33
  007 Program Running: ABCD, 8-33
  008 Single-Stepping: ABCD, 8-33
  009 Select Parameter Mode to Continue, 8-33
  202 Drive D is not ready, 8-34
  203 Power Lost During Program, 8-34
  204 SERCOS Ring was disconnected, 8-34
  205 Parameter transfer warning in Task A, 8-34
Stop, 6-56
Sub, 6-57
Subroutines, 4-15, 5-10
symbolic names, 4-11
Synchronization Status, C-7
System, 4-60
System Control Register, 2-8
System Diagnostics - Codes and Messages, 8-32
System Parameters
  Electronic Line Shaft, 3-32
    C-0-0150 ELS Master Type, 3-32
    C-0-0151 ELS Master Axis, 3-32
    C-0-0152 ELS Master Task Assignment, 3-32
    C-0-0153 Virtual Master Programmed Velocity, 3-33
    C-0-0154 Virtual Master Programmed Acceleration, 3-33
    C-0-0155 Virtual Master Programmed Deceleration, 3-33
    C-0-0156 Virtual Master E-Stop Deceleration, 3-33
    C-0-0157 ELS Master Current Position, 3-34
    C-0-0158 ELS Master Current Velocity, 3-34
    C-0-0159 Real Master Zero Velocity Window, 3-34
    C-0-0160 Virtual Master Maximum Jog Velocity, 3-34
Jogging and Display, 3-22
    C-0-0042 World Large Increment, 3-22
    C-0-0043 World Small Increment, 3-22
    C-0-0045 World Fast Jog Speed, 3-22
    C-0-0046 World Slow Jog Speed, 3-22
    C-0-0052 Axis Large Increment, 3-23
    C-0-0053 Axis Small Increment, 3-23
    C-0-0055 Axis Fast Jog Velocity, 3-23
    C-0-0056 Axis Slow Jog Velocity, 3-23
Parameter Lists, 3-51, 3-104
  C 1.2000 List of All Parameters, 3-76, 3-104
  C 1.2001 List of Required Parameters, 3-76, 3-104
  C-0-2000 List of All Parameters, 3-51
  C-0-2001 List of Required Parameters, 3-51
  C-0-3100 Cam Tags, 3-59
Program Management, 3-24
  C-0-0090, 3-24
  C-0-0091 Total Program Memory, 3-24
  C-0-0092 Available Program Memory, 3-24
  C-0-0093 Contiguous Program Memory, 3-24
System Setup, 3-13
  C 1.1 Language Selection, 3-13
  C 1.2 Unit Number, 3-13
  C 1.3 Host Serial Port Setup, 3-14
  C 1.4 Pendant Serial Port Setup, 3-14
  C 1.5 Parallel I/O Device, 3-15
  C 1.6 I/O Device Setup, 3-15
  C 1.7 I/O Device Direction, 3-15
  C-0-0008 I/O Device VME Start Address, 3-16
  C-0-0009 Error Reaction Mode, 3-16
System Status, 3-27
  C-0-0120 Operating Mode, 3-29
  C-0-0121 SERCOS Communication Phase, 3-29
  C-0-0124 Extended Diagnostic, 3-30
  C-0-0125 System Timer Value, 3-30
VME Bus, 3-36
  C-0-0200 Bus Arbitration Mode, 3-36
  C-0-0201 Bus Release Mode, 3-37
  C-0-0202 VME/CLC Interface, 3-38
  C-0-0203 VME Arbiter Status, 3-39
  C-0-0232 Unit 4 Window Address Page, 3-42
  C-0-0287 Unit F Window Address Page, 3-51
System Status, 3-27
System Status Register, 2-1, 2-20
System Timer Value, 3-30

T
  T Label Text, B-16
tables, 5-17
  event, 5-19
  indirect addressing, 5-18
  points
    absolute, 5-22
    relative, 5-23
  zone protection, 5-24
Target Point Number, 3-69
Target Position, 3-93
Task Control Registers, 2-10
Task Diagnostic Message, 3-72
Task Jog Control Registers, 2-15
  Coordinated Jogging:, 2-15
  Task Motion Type, 3-62
<table>
<thead>
<tr>
<th>Task Operating Mode, 3-71</th>
<th>AXIS/MOVE (Single Axis, Non-Coordinated), 7-11</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task Parameter Lists, 3-76</td>
<td>AXIS/RATIO, 7-13</td>
</tr>
<tr>
<td>Task Parameters, 3-62</td>
<td>AXIS/SPINDLE (Continuous Velocity Mode), 7-15</td>
</tr>
<tr>
<td>Task Setup, 3-62</td>
<td>AXIS/START, 7-17</td>
</tr>
<tr>
<td>T-0-0001 Task Motion Type, 3-62</td>
<td>AXIS/STOP, 7-19</td>
</tr>
<tr>
<td>T-0-0005 World Position Units, 3-64</td>
<td>AXIS/WAIT (Axis Wait For In-Position), 7-21</td>
</tr>
<tr>
<td>T-0-0011 Coordinated X Axis, 3-64</td>
<td>Call, 7-23</td>
</tr>
<tr>
<td>T-0-0012 Coordinated Y Axis, 3-65</td>
<td>CAM/ACTIVATE, 7-23</td>
</tr>
<tr>
<td>T-0-0013 Coordinated Z Axis, 3-65</td>
<td>CAM/ADJUST, 7-25</td>
</tr>
<tr>
<td>T-0-0020 Maximum Path Speed, 3-65</td>
<td>CAM/BUILD, 7-26</td>
</tr>
<tr>
<td>T-0-0021 Maximum Path Acceleration, 3-65</td>
<td>CAM/STATUS, 7-28</td>
</tr>
<tr>
<td>T-0-0022 Maximum Path Deceleration, 3-66</td>
<td>CAPTURE/ENABLE, 7-29</td>
</tr>
<tr>
<td>T-0-0023 Look Ahead Distance, 3-66</td>
<td>CAPTURE/SETUP, 7-31</td>
</tr>
<tr>
<td>T-0-0024 Velocity Override, 3-67</td>
<td>DATA/RELEASE (Device Fault Release), 7-33</td>
</tr>
<tr>
<td>T-0-0025 Maximum Jog Increment, 3-67</td>
<td>DEFINE (Define Name For Variable), 7-36</td>
</tr>
<tr>
<td>T-0-0026 Maximum Jog Velocity, 3-67</td>
<td>DELAY (Suspend Task Execution), 7-38</td>
</tr>
<tr>
<td>T-0-0050 Kinematic Value 1, 3-68</td>
<td>ELS/ADJUST (Adjust ELS Axis), 7-39</td>
</tr>
<tr>
<td>T-0-0051 Kinematic Value 2, 3-68</td>
<td>ELS/DECEL (Set Master Decel), 7-41</td>
</tr>
<tr>
<td>T-0-0052 Kinematic Value 3, 3-68</td>
<td>ELS/INIT (Initialize ELS Axes), 7-42</td>
</tr>
<tr>
<td>T-0-0053 Kinematic Value 4, 3-68</td>
<td>ELS/MODE (Set ELS Axis Mode), 7-45</td>
</tr>
<tr>
<td>T-0-0054 Kinematic Value 5, 3-68</td>
<td>ELS/STOP, 7-46</td>
</tr>
<tr>
<td>T-0-0055 Kinematic Value 6, 3-69</td>
<td>EVENT/DONE (Signal Event Completed), 7-47</td>
</tr>
<tr>
<td>T-0-0056 Kinematic Value 7, 3-69</td>
<td>EVENT/END (Mark End of Event), 7-49</td>
</tr>
<tr>
<td>T-0-0057 Kinematic Value 8, 3-69</td>
<td>EVENT/START (Start of Event function), 7-50</td>
</tr>
<tr>
<td>T-0-0058 Kinematic Value 9, 3-69</td>
<td>EVENT/TRIGGER (Trigger a Task Event), 7-51</td>
</tr>
<tr>
<td>Task Status, 3-69</td>
<td>EVENT/WAIT (Pause Task for Event Done Signal), 7-53</td>
</tr>
<tr>
<td>T-0-0100 Target Point Number, 3-69</td>
<td>FUNCTION/ARG, 7-55</td>
</tr>
<tr>
<td>T-0-0101 Segment Status, 3-70</td>
<td>FUNCTION/END, 7-56</td>
</tr>
<tr>
<td>T-0-0102 Rate Limit Status, 3-70</td>
<td>FUNCTION/START, 7-57</td>
</tr>
<tr>
<td>T-0-0111 Current X Position, 3-70</td>
<td>GOSUB (Go To Subroutine), 7-58</td>
</tr>
<tr>
<td>T-0-0112 Current Y Position, 3-71</td>
<td>GOTO (Go To Mark), 7-59</td>
</tr>
<tr>
<td>T-0-0113 Current Z Position, 3-71</td>
<td>IF (If-Else-Endif Conditional Branch), 7-60</td>
</tr>
<tr>
<td>T-0-0120 Task Operating Mode, 3-71</td>
<td>Instruction Format, 7-4</td>
</tr>
<tr>
<td>T-0-0122 Task Diagnostic Message, 3-72</td>
<td>KINEMATIC (Use a Kinematic Definition for a Task), 7-64</td>
</tr>
<tr>
<td>T-0-0123 Task Status Message, 3-72</td>
<td>MESSAGE/DIAG (Task Diagnostic Message Definition), 7-66</td>
</tr>
<tr>
<td>T-0-0130 Current Instruction Pointer, 3-72</td>
<td>MESSAGE/STATUS (Task Status Message Definition), 7-67</td>
</tr>
<tr>
<td>T-0-0131 Current Instruction, 3-73</td>
<td>MOVE/CIRCLE (Coordinated Move with Circular Interpolation), 7-68</td>
</tr>
<tr>
<td>T-0-0132 Instruction Pointer at Error, 3-73</td>
<td>MOVE/JOINT (Coordinated Move Joint Point to Point), 7-70</td>
</tr>
<tr>
<td>T-0-0200 Last Active Event Number, 3-76</td>
<td>MOVE/LINE (Coordinated Move with Straight Line Interpolation), 7-71</td>
</tr>
<tr>
<td>Task Setup, 3-62</td>
<td>PARAMETER/BIT (Initialize Parameter Bit), 7-73</td>
</tr>
<tr>
<td>Task Status, 3-69, 3-71</td>
<td>PARAMETER/GET (Load Parameter to a Variable), 7-75</td>
</tr>
<tr>
<td>Task Status Message, 3-72</td>
<td>PARAMETER/INIT (Initialize a Parameter), 7-77</td>
</tr>
<tr>
<td>Task Status Registers, 2-22</td>
<td>PARAMETER/SET (Set a Parameter), 7-78</td>
</tr>
<tr>
<td>TASK/AXES (Task Axes Definition), 7-100</td>
<td>PATH/ABORT (Aborts Coordinated Motion), 7-80</td>
</tr>
<tr>
<td>TASK/END (Mark the End of a Task), 7-102</td>
<td>PATH/POSITION (Get Current Path Absolute Position), 7-81</td>
</tr>
<tr>
<td>TASK/START &amp; TASK/END, 7-3</td>
<td>PATH/RESUME (Resume Coordinated Motion), 7-82</td>
</tr>
<tr>
<td>TASK/START (Define the Start of a Task(s)), 7-103</td>
<td>PATH/STOP (Halt Coordinated Motion), 7-83</td>
</tr>
<tr>
<td>tasks, 1-4, 4-61</td>
<td>PATH/WAIT (Pause Program for Motion), 7-84</td>
</tr>
<tr>
<td>Teach Pendant, 1-4, 3-43</td>
<td>PLC/CLEAR (Clear I/O Register Bit), 7-85, 7-86</td>
</tr>
<tr>
<td>Control, 2-1, 2-31</td>
<td></td>
</tr>
<tr>
<td>Status, 2-1, 2-30</td>
<td></td>
</tr>
<tr>
<td>Text, 8-2</td>
<td></td>
</tr>
<tr>
<td>Text Language Error Messages, 8-22</td>
<td></td>
</tr>
<tr>
<td>Text Language Programming</td>
<td></td>
</tr>
<tr>
<td>CLC Instructions</td>
<td></td>
</tr>
<tr>
<td>AXIS/EVENT, 7-7</td>
<td></td>
</tr>
<tr>
<td>AXIS/HOME, 7-8</td>
<td></td>
</tr>
<tr>
<td>AXIS/INITIALIZE, 7-9</td>
<td></td>
</tr>
<tr>
<td>Directives</td>
<td>Page</td>
</tr>
<tr>
<td>------------</td>
<td>------</td>
</tr>
<tr>
<td>PLC/READ (Read I/O Register(s))</td>
<td>7-87</td>
</tr>
<tr>
<td>PLC/SET (Set I/O Register Bit)</td>
<td>7-88</td>
</tr>
<tr>
<td>PLC/TEST (Test I/O Register Bit)</td>
<td>7-89</td>
</tr>
<tr>
<td>PLC/WAIT (Pause Program for I/O)</td>
<td>7-90</td>
</tr>
<tr>
<td>PLC/WRITE (Write to I/O Register(s))</td>
<td>7-91</td>
</tr>
<tr>
<td>PLS/INIT (Compiler directive)</td>
<td>7-92</td>
</tr>
<tr>
<td>RETURN (Return From Subroutine)</td>
<td>7-94</td>
</tr>
<tr>
<td>ROBOT/ORIGIN</td>
<td>7-95</td>
</tr>
<tr>
<td>ROBOT/TOOL</td>
<td>7-96</td>
</tr>
<tr>
<td>SEQ/LIST</td>
<td>7-98</td>
</tr>
<tr>
<td>SEQ/STEP</td>
<td>7-99</td>
</tr>
<tr>
<td>SEQUENCER</td>
<td>7-97</td>
</tr>
<tr>
<td>TASK/START &amp; TASK/END</td>
<td>7-3</td>
</tr>
<tr>
<td>The Status Menu</td>
<td>4-59</td>
</tr>
<tr>
<td>The Tools Menu</td>
<td>4-23</td>
</tr>
<tr>
<td>Topic Name</td>
<td>A-2</td>
</tr>
<tr>
<td>Torque Feedback (cyclic)</td>
<td>3-96</td>
</tr>
<tr>
<td>Torque Mode Commanded Torque</td>
<td>3-96</td>
</tr>
<tr>
<td>Total Program Memory</td>
<td>3-24</td>
</tr>
<tr>
<td>Transcendental functions</td>
<td>5-16</td>
</tr>
<tr>
<td>Transfer Cams</td>
<td>4-5</td>
</tr>
<tr>
<td>Transfer Events</td>
<td>4-6</td>
</tr>
<tr>
<td>Transfer I/O Mapper</td>
<td>4-6</td>
</tr>
<tr>
<td>Transfer Parameters</td>
<td>4-7, D-2</td>
</tr>
<tr>
<td>Transfer Points</td>
<td>4-8</td>
</tr>
<tr>
<td>Transfer Tables Between Programs (PX)</td>
<td>B-32, B-34</td>
</tr>
<tr>
<td>Transfer Variables</td>
<td>4-8</td>
</tr>
<tr>
<td>Transfer Zones</td>
<td>4-9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Unit</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unit 4 Window Address Page</td>
<td>3-42</td>
</tr>
<tr>
<td>Unit F Window Address Page</td>
<td>3-51</td>
</tr>
<tr>
<td>Unit Number</td>
<td>3-13</td>
</tr>
<tr>
<td>Unlatch</td>
<td>2-29</td>
</tr>
<tr>
<td>User Labels</td>
<td>4-12</td>
</tr>
<tr>
<td>User Program Variables</td>
<td>B-16</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>V</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Variables</td>
<td>4-57, 5-13</td>
</tr>
<tr>
<td>global</td>
<td>5-14</td>
</tr>
<tr>
<td>VCAM</td>
<td>C-12</td>
</tr>
<tr>
<td>Veloc</td>
<td>6-58</td>
</tr>
<tr>
<td>Velocity Mode</td>
<td>3-78</td>
</tr>
<tr>
<td>Velocity Mode Ramp</td>
<td>3-81</td>
</tr>
<tr>
<td>Velocity Override</td>
<td>3-67</td>
</tr>
<tr>
<td>velocity profiles</td>
<td>C-5</td>
</tr>
<tr>
<td>Velocity synchronization</td>
<td>1-7, C-5</td>
</tr>
<tr>
<td>View Menu</td>
<td>4-15</td>
</tr>
<tr>
<td>Virtual Master</td>
<td>6-23</td>
</tr>
<tr>
<td>Virtual Master E-Stop Deceleration</td>
<td>3-33</td>
</tr>
<tr>
<td>Virtual Master Maximum Jog Velocity</td>
<td>3-34</td>
</tr>
<tr>
<td>Virtual Master Programmed Acceleration</td>
<td>3-33</td>
</tr>
<tr>
<td>Virtual Master Programmed Deceleration</td>
<td>3-33</td>
</tr>
<tr>
<td>Virtual Master Programmed Velocity</td>
<td>3-33</td>
</tr>
<tr>
<td>Virtual master through SERCOS</td>
<td>7-43</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>W</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Wait</td>
<td>6-62</td>
</tr>
<tr>
<td>Warning Messages</td>
<td>8-31, 8-34</td>
</tr>
<tr>
<td>World Fast Jog Speed</td>
<td>3-22</td>
</tr>
<tr>
<td>World Large Increment</td>
<td>3-22</td>
</tr>
<tr>
<td>World Position Units</td>
<td>3-64</td>
</tr>
<tr>
<td>World Slow Jog Speed</td>
<td>3-22</td>
</tr>
<tr>
<td>World Small Increment</td>
<td>3-22</td>
</tr>
<tr>
<td>Writing Data to the CLC</td>
<td>B-2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Z</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>zone protection table</td>
<td>5-24</td>
</tr>
<tr>
<td>Zones</td>
<td>4-58, 5-24</td>
</tr>
<tr>
<td>Zoom Out</td>
<td>4-15</td>
</tr>
</tbody>
</table>
READER COMMENT CARD AND REGISTRATION FORM

We welcome your evaluation of this manual. Please complete this form to help us to improve our publications and also to receive notification of revisions to this manual.

Publication: CLC Multi-Axis Coordinated Motion Control
Publication No. IAE 74792
Revision F, October, 1996

Please indicate the degree to which each statement applies to you by circling whether you ① strongly agree, ② agree, ③ are uncertain, ④ disagree, or ⑤ strongly disagree with the statement.

1. The manual is well organized. ① ② ③ ④ ⑤
2. I can find the information I want. ① ② ③ ④ ⑤
3. The index is thorough. ① ② ③ ④ ⑤
4. The information is easy to understand. ① ② ③ ④ ⑤
5. The manual is clearly written. ① ② ③ ④ ⑤
6. Concepts and vocabulary are easy to understand. ① ② ③ ④ ⑤
7. Examples are clear and helpful. ① ② ③ ④ ⑤
8. The manual contains enough illustrations. ① ② ③ ④ ⑤
9. Layout and format enhance the manual’s usefulness. ① ② ③ ④ ⑤
10. The quality of this manual would influence any repeat purchase decision. ① ② ③ ④ ⑤
11. The scope of the information in the manual meets my needs. ① ② ③ ④ ⑤

Name __________________________________________
Company __________________________________________
Address __________________________________________ Dept _______________________
City ____________________________ State ____________ Zip _______________________
Telephone (___) __________________________
Comments: __________________________