If you should have any questions about the U600 board or comments regarding the documentation, please refer to Aerotech online at:
http://www.aerotech.com

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

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

UNIDEX 600 and UNIDEX 620 are products of Aerotech, Inc.
Windows, Windows 95, and Windows NT are registered trademarks of Microsoft.
Windows and Windows NT are products of Microsoft Corporation.
Visual Basic® (VB®) is a product of Microsoft Corporation.
LabView® is a product of National Instruments
The 80960 RISC processor is a product of Intel Corporation.

The UNIDEX 600 Series Library Reference Manual Revision History:

    Rev 1.0   September 20, 1996
    Rev 1.1   May 1, 1997
    Rev 1.2   Aug 21, 1998
    Rev 1.3   June 12, 2000
    Rev 1.4   December 11, 2001

© Aerotech, Inc., 2001
TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION ................................................................. 1-1
  1.1. Overview .................................................................................. 1-1
       1.1.1. Developing an Application ........................................... 1-2
  1.2. Building ..................................................................................... 1-4
  1.3. Calling ......................................................................................... 1-5
  1.4. Visual Basic Programming Quick Start ....................................... 1-6
       1.4.1. Developing an Application ........................................... 1-6
  1.5. C/C++ Language Programming Quick Start .................................. 1-7
       1.5.1. Developing an Application ........................................... 1-7
       1.5.2. C++ Language Example Programs ............................... 1-7
  1.6. LabView Programming Quick Start ............................................. 1-8
       1.6.1. LabView Library Example Programs ............................... 1-8
  1.7. Debugging .................................................................................. 1-9
       1.7.1. AerDebug .................................................................. 1-10
  1.8. Error Handling ........................................................................... 1-11
       1.8.1. Error Returns .............................................................. 1-11
       1.8.2. Programming Errors ..................................................... 1-12
       1.8.3. Faults ........................................................................ 1-12
  1.9. Examples .................................................................................... 1-14

CHAPTER 2: AUTOMATION PROGRAM FUNCTIONS ............................. 2-1
  2.1. Overview ................................................................................... 2-1
  2.2. AerAutoProgGetNumPrograms ..................................................... 2-2
  2.3. AerAutoProgGetProgram ............................................................... 2-3
  2.4. AerAutoProgSetProgram ............................................................... 2-4
  2.5. AerAutoProgAddProgram ............................................................. 2-5
  2.6. AerAutoProgRemoveProgram ..................................................... 2-6

CHAPTER 3: AUXILIARY TABLE FUNCTIONS ........................................ 3-1
  3.1. Introduction ............................................................................... 3-1
  3.2. AerAuxTableAllocate ................................................................. 3-2
  3.3. AerAuxTableFree .......................................................................... 3-3
  3.4. AerAuxTableGetMode ................................................................. 3-4
  3.5. AerAuxTableGetPoint ................................................................. 3-5
  3.6. AerAuxTableGetMultipoints ......................................................... 3-6
  3.7. AerAuxTableGetStatus ............................................................... 3-7
  3.8. AerAuxTableGetTables ............................................................... 3-8
  3.9. AerAuxTableSetMode ................................................................. 3-9
  3.10. AerAuxTableSetPoint ................................................................. 3-10
  3.11. AerAuxTableSetMultipoints ......................................................... 3-11
  3.12. Related Axis Parameters ............................................................. 3-12

CHAPTER 4: AXIS CALIBRATION FUNCTIONS ...................................... 4-1
  4.1. Introduction ............................................................................... 4-1
  4.2. AerAxisCalAllocateTable ............................................................ 4-4
  4.3. AerAxisCalFileDownload ............................................................. 4-5
  4.4. AerAxisCalFreeTable ................................................................. 4-7
  4.5. AerAxisCalGet ................................................................. 4-8
### Table of Contents

#### 4.6. AerAxisCalGetItems .................................................. 4-9
#### 4.7. AerAxisCalGetMode .................................................. 4-10
#### 4.8. AerAxisCalGetPacket .............................................. 4-11
#### 4.9. AerAxisCalGetPoint .................................................. 4-12
#### 4.10. AerAxisCalGetRawPosition ................................. 4-13
#### 4.11. AerAxisCalGetStatus ............................................... 4-14
#### 4.12. AerAxisCalGetStatusPacket .................................. 4-15
#### 4.13. AerAxisCalRemoveAll ............................................ 4-16
#### 4.14. AerAxisCalReset ..................................................... 4-17
#### 4.15. AerAxisCalSet .......................................................... 4-18
#### 4.16. AerAxisCalSetMode ................................................ 4-19
#### 4.17. AerAxisCalSetPoint .................................................. 4-20
#### 4.18. 2D Calibration ......................................................... 4-21
#### 4.19. AerAxisCal2DAllocateTable .................................... 4-22
#### 4.20. AerAxisCal2DFileDownload ..................................... 4-23
#### 4.21. AerAxisCal2DFreeTable ............................................ 4-24
#### 4.22. AerAxisCal2DGetData ................................................ 4-25
#### 4.23. AerAxisCal2DGetMode .............................................. 4-26
#### 4.24. AerAxisCal2DGetPoint .............................................. 4-27
#### 4.25. AerAxisCal2DSetData ............................................... 4-28
#### 4.26. AerAxisCal2DSetMode .............................................. 4-29
#### 4.27. AerAxisCal2DSetPoint ............................................... 4-30
#### 4.28. Related Axis Parameters .......................................... 4-31

### CHAPTER 5: ELECTRONIC CAMMING FUNCTIONS .............. 5-1

#### 5.1. Introduction .............................................................. 5-1
#### 5.1.1. Required Camming Steps ........................................ 5-3
#### 5.2. AerCamTableAllocate ................................................ 5-4
#### 5.3. AerCamTableCalcCoeff ............................................. 5-5
#### 5.4. AerCamTableFileDownload ....................................... 5-6
#### 5.5. AerCamTableFree ....................................................... 5-9
#### 5.6. AerCamTableGetMode ................................................ 5-10
#### 5.7. AerCamTableGetMultiPoints ..................................... 5-11
#### 5.8. AerCamTableGetPoint ................................................ 5-12
#### 5.9. AerCamTableGetStatus ............................................. 5-13
#### 5.10. AerCamTableGetTrack .............................................. 5-14
#### 5.11. AerCamTableSetMode ............................................... 5-15
#### 5.12. AerCamTableSetMultiPoints .................................... 5-17
#### 5.13. AerCamTableSetPoint ................................................ 5-18
#### 5.14. AerCamTableSetTrack .............................................. 5-19
#### 5.15. Related Axis Parameters .......................................... 5-20

### CHAPTER 6: COMPILER FUNCTIONS ................................. 6-1

#### 6.1. Introduction .............................................................. 6-1
#### 6.2. AerCompilerAbort ................................................... 6-4
#### 6.3. AerCompilerAddDefinesFile ...................................... 6-5
#### 6.4. AerCompilerAutoIncludeEx ....................................... 6-6
#### 6.5. AerCompilerAutoRun ................................................. 6-8
#### 6.6. AerCompilerClose (AerCompilerDestroy) ................... 6-12
#### 6.7. AerCompilerCompileFile ............................................ 6-13
#### 6.8. AerCompilerCompileLine .......................................... 6-15
| 6.9. | AerCompilerDownload                                      | 6-16 |
| 6.10. | AerCompilerDownloadEx                                    | 6-17 |
| 6.11. | AerCompilerDownloadImmediate                             | 6-19 |
| 6.12. | AerCompilerErrGetCode                                    | 6-21 |
| 6.13. | AerCompilerErrGetData                                     | 6-22 |
| 6.15. | AerCompilerErrsGetNumOf                                  | 6-24 |
| 6.16. | AerCompilerErrGetText                                    | 6-25 |
| 6.17. | AerCompilerFileNameToHandle                              | 6-26 |
| 6.18. | AerCompilerGetLineText                                   | 6-27 |
| 6.19. | AerCompilerGetNumOfLines                                 | 6-28 |
| 6.20. | AerCompilerGetProgVarByName                              | 6-29 |
| 6.21. | AerCompilerGetProgVarNumber                              | 6-30 |
| 6.22. | AerCompilerGetProgVarTotal                               | 6-31 |
| 6.23. | AerCompilerGetQueueMode                                  | 6-32 |
| 6.24. | AerCompilerGetStatus                                     | 6-33 |
| 6.25. | AerCompilerGetUniqueID                                   | 6-34 |
| 6.26. | AerCompilerLoadAxisNames                                 | 6-35 |
| 6.27. | AerCompilerOpen (AerCompilerCreate)                      | 6-36 |
| 6.28. | AerCompilerOpenEx                                        | 6-37 |
| 6.29. | AerCompilerRemoveDefinesFile                             | 6-38 |
| 6.30. | AerCompilerRunImmediate                                 | 6-39 |
| 6.31. | AerCompilerSetQueueMode                                  | 6-41 |

CHAPTER 7: **AXIS CONFIGURATION FUNCTIONS** .............................................. 7-1

7.1. Introduction ........................................................................................ 7-1
7.1.1. Encoder and Resolver/Inductosyn Board Resolution ...................... 7-2
7.1.2. Axis Configuration File Format ..................................................... 7-2
7.2. AerConfig .......................................................................................... 7-5
7.3. AerConfigDownloadFile ........................................................................ 7-6
7.4. AerConfigEncoder ............................................................................... 7-7
7.5. AerConfigEncoderHall ......................................................................... 7-8
7.6. AerConfigGet .................................................................................... 7-9
7.7. AerConfigGetEncoder .......................................................................... 7-10
7.8. AerConfigGetEncoderHall ................................................................... 7-11
7.9. AerConfigGetHallPolePairs ................................................................ 7-12
7.10. AerConfigGetMaster .......................................................................... 7-13
7.11. AerConfigGetMasterAxis ................................................................... 7-14
7.12. AerConfigGetResolver ........................................................................ 7-15
7.13. AerConfigGetResolverHall .................................................................. 7-16
7.14. AerConfigHallPolePairs ..................................................................... 7-17
7.15. AerConfigMaster ................................................................................ 7-18
7.16. AerConfigMasterAxis ......................................................................... 7-19
7.17. AerConfigMasterAuto ......................................................................... 7-21
7.18. AerConfigReadPacket ........................................................................ 7-22
7.19. AerConfigReadPacketEx ..................................................................... 7-23
7.20. AerConfigResolver ............................................................................. 7-24
7.21. AerConfigResolverHall ....................................................................... 7-25
7.22. AerConfigWritePacket ........................................................................ 7-26
CHAPTER 8: DATA CENTER FUNCTIONS ....................................................... 8-1
  8.1. Introduction .................................................................................... 8-1
  8.2. AerDCGetAxisDirectEx .................................................................... 8-2
  8.3. AerDCGetTaskDirect ....................................................................... 8-3

CHAPTER 9: ERROR FUNCTIONS ................................................................. 9-1
  9.1. Introduction .................................................................................... 9-1
  9.2. AERERRGETMESS .......................................................................... 9-2
  9.3. AerErrGetMessage and AerErrGetMessageEx .................................. 9-3
  9.4. AerErrGetSeverity ......................................................................... 9-4
  9.5. AerErrLogError ............................................................................. 9-5
  9.6. AerErrLogFileClose ....................................................................... 9-6
  9.7. AerErrLogFileOpen ....................................................................... 9-7
  9.8. AerErrMessageBox ....................................................................... 9-8

CHAPTER 10: EVENT FUNCTIONS ............................................................... 10-1
  10.1. Introduction .................................................................................. 10-1
  10.2. AerEventCloseEvent ..................................................................... 10-2
  10.3. AerEventCreateEvent ................................................................... 10-3
  10.4. AerEventGenerateInt ..................................................................... 10-5
  10.5. AerEventTest ............................................................................... 10-6

CHAPTER 11: MEMORY FUNCTIONS ............................................................ 11-1
  11.1. Introduction .................................................................................. 11-1
  11.2. AerMemCheck ............................................................................... 11-2
  11.3. AerMemGetFree .......................................................................... 11-3

CHAPTER 12: MOVE FUNCTIONS ................................................................. 12-1
  12.1. Introduction .................................................................................. 12-1
  12.2. AerJogGetMode .......................................................................... 12-3
  12.3. AerJogSetMode .......................................................................... 12-4
  12.4. AerMoveAbort, AerMoveMAbort ................................................. 12-5
  12.5. AerMoveAbsolute, AerMoveMAbsolute ....................................... 12-6
  12.6. AerMoveAxis .............................................................................. 12-7
  12.7. AerMoveFeedhold, AerMoveMFeedhold ....................................... 12-9
  12.8. AerMoveFreerun, AerMoveMFreerun .......................................... 12-10
  12.9. AerMoveHalt, AerMoveMHalt ..................................................... 12-11
  12.10. AerMoveHome, AerMoveMHome .............................................. 12-12
  12.11. AerMoveHomeNoLimit, AerMoveMHomeNoLimit ...................... 12-13
  12.12. AerMoveHomeQuick, AerMoveMHomeQuick ............................. 12-14
  12.15. AerMoveInfeedSlave, AerMoveMInfeedSlave ............................. 12-18
  12.16. AerMoveLinear .......................................................................... 12-19
  12.17. AerMoveMulti ........................................................................... 12-20
  12.18. AerMoveOscillate, AerMoveM Oscillate .................................. 12-22
  12.19. AerMoveQueueAbsolute, AerMoveMQueueAbsolute ................. 12-23
  12.20. AerMoveQueueFlush, AerMoveMQueueFlush ............................ 12-24
  12.21. AerMoveQueueHold .................................................................. 12-25
12.22. AerMoveQueueIncremental, AerMoveMQueueIncremental ............ 12-26
12.25. AerMoveWaitDone ..................................................................... 12-29
12.26. AerMoveWaitDoneMulti ............................................................. 12-30

CHAPTER 13: PARAMETER FUNCTIONS .................................................. 13-1
13.1. Introduction .................................................................................. 13-1
13.2. AerParmAxisDownloadFile ............................................................ 13-2
13.3. AerParmAxisGetCount .................................................................. 13-3
13.4. AerParmAxisGetInfo ..................................................................... 13-4
13.5. AerParmAxisGetValue ................................................................... 13-6
13.6. AerParmAxisReadValue ............................................................... 13-7
13.7. AerParmAxisSetValue ................................................................... 13-8
13.8. AerParmAxisWriteValue ............................................................... 13-9
13.9. AerParmGlobalDownloadFile ......................................................... 13-10
13.10. AerParmGlobalGetCount ............................................................. 13-11
13.11. AerParmGlobalGetInfo ................................................................. 13-12
13.12. AerParmGlobalGetValue ............................................................... 13-14
13.13. AerParmGlobalReadValue ........................................................... 13-15
13.14. AerParmGlobalSetValue ............................................................... 13-16
13.15. AerParmGlobalWriteValue ........................................................... 13-17
13.16. AerParmMachineDownloadFile ..................................................... 13-18
13.17. AerParmMachineGetCount ............................................................ 13-19
13.18. AerParmMachineGetInfo ............................................................... 13-20
13.20. AerParmMachineReadValue .......................................................... 13-23
13.22. AerParmMachineWriteValue ........................................................ 13-25
13.23. AerParmMAxisGetValue ............................................................... 13-26
13.25. AerParmTaskDownloadFile ......................................................... 13-28
13.27. AerParmTaskGetInfo ................................................................. 13-30
13.28. AerParmTaskGetValue ................................................................. 13-32
13.29. AerParmTaskReadValue ............................................................. 13-33
13.30. AerParmTaskSetValue ................................................................. 13-34
13.31. AerParmTaskWriteValue ............................................................. 13-35

CHAPTER 14: PROBE FUNCTIONS .......................................................... 14-1
14.1. Introduction .................................................................................. 14-1
14.2. AerProbeDisable .......................................................................... 14-2
14.3. AerProbeEnable ........................................................................... 14-3
14.4. AerProbeGetPosition .................................................................... 14-4
14.5. AerProbeGetStatus ....................................................................... 14-5
14.6. AerProbeSetInput ......................................................................... 14-6
14.7. Example Code Fragment to Use Probe Functions ......................... 14-7

CHAPTER 15: PROFILE FUNCTIONS ...................................................... 15-1
15.1. Introduction .................................................................................. 15-1
15.2. AerProfileFlushQueue ................................................................. 15-2
15.3. AerProfileFlushQueueMulti ............................................................. 15-3
15.4. AerProfileLoadQueue ................................................................. 15-4
15.5. AerProfileLoadQueueMulti ........................................................... 15-5
15.6. AerProfileStartQueue ................................................................. 15-6
15.7. AerProfileStartQueueMulti........................................................... 15-7
15.8. AerProfileStopQueue ................................................................. 15-8
15.9. AerProfileStopQueueMulti........................................................... 15-9

CHAPTER 16: PROGRAMMING ERROR FUNCTIONS ........................................... 16-1
16.1. Introduction ...................................................................................... 16-1
16.2. AerProgErrFlush ........................................................................... 16-2
16.3. AerProgErrGet ............................................................................. 16-3
16.4. AerProgErrGetAltStatusMessage ................................................... 16-4
16.5. AerProgErrGetFaultMessage ......................................................... 16-5
16.6. AerProgErrGetMessage ............................................................... 16-6
16.7. AerProgErrGetMotionMessage ....................................................... 16-7
16.8. AerProgErrGetServoMessage ....................................................... 16-8
16.9. AerProgErrGetStatusMessage ....................................................... 16-9
16.10. AerProgErrSetUserFault .............................................................. 16-10
16.11. AerProgErrWaitModeGet ............................................................. 16-11
16.12. AerProgErrWaitModeSet ............................................................. 16-12

CHAPTER 17: PROGRAM FUNCTIONS ............................................................. 17-1
17.1. Introduction ...................................................................................... 17-1
17.2. AerProgramFree ............................................................................ 17-2
17.3. AerProgramGetBreakPoint .............................................................. 17-3
17.4. AerProgramGetHandle ................................................................... 17-4
17.5. AerProgramGetHeader .................................................................. 17-5
17.6. AerProgramGetInfo ...................................................................... 17-6
17.7. AerProgramGetNumber .................................................................. 17-7
17.8. AerProgramSetBreakPoint ............................................................. 17-8

CHAPTER 18: PSO (LASER) FUNCTIONS ......................................................... 18-1
18.1. Introduction ...................................................................................... 18-1
18.2. AerPSOAllocateFiringTable ............................................................ 18-2
18.3. AerPSOFirePulse .......................................................................... 18-3
18.4. AerPSOSetDigitalOutput ............................................................... 18-4
18.5. AerPSOSetFiringDist ..................................................................... 18-5
18.6. AerPSOSetMultAnalogOutput ........................................................ 18-6
18.7. AerPSOSetPulse ........................................................................... 18-7
18.8. AerPSOSetTracking ...................................................................... 18-9
18.9. AerPSOStopFiring .......................................................................... 18-10
18.10. AerPSOWriteMultFiringPoints ...................................................... 18-11

CHAPTER 19: REGISTRY FUNCTIONS ............................................................... 19-1
19.1. Introduction ...................................................................................... 19-1
19.2. AerRegGetDefDevice .................................................................... 19-3
19.3. AerRegGetDefDeviceInfo ............................................................... 19-4
19.4. AerRegGetDeviceInfo .................................................................... 19-5
19.5. AerRegGetFileName ...................................................................... 19-7
19.6. AerRegQueryCardCount ............................................................... 19-8
19.7.  AerRegQueryCardList ................................................................. 19-9
19.8.  AerRegSetDefDevice ................................................................. 19-10
19.9.  AerRegSetDeviceInfo ................................................................. 19-11
19.10. AerRegSetFileName .................................................................. 19-12

CHAPTER 20:  STRIP FUNCTIONS ......................................................... 20-1
20.1.  Introduction .............................................................................. 20-1
20.2.  AerStripAllocate ................................................................. 20-2
20.3.  AerStripFree ................................................................. 20-3
20.4.  AerStripGetIOPosLatchStatus .................................................. 20-4
20.5.  AerStripGetSample ................................................................. 20-5
20.6.  AerStripGetStatus ................................................................. 20-6
20.7.  AerStripGlobalAllocate ........................................................... 20-7
20.8.  AerStripGlobalAllocateTrigger ................................................ 20-8
20.9.  AerStripGlobalFree ................................................................. 20-9
20.10. AerStripGlobalFreeTrigger ..................................................... 20-10
20.11. AerStripGlobalGetImmediate .................................................. 20-11
20.12. AerStripGlobalGetQueue .......................................................... 20-12
20.15. AerStripGlobalGetSample ....................................................... 20-15
20.16. AerStripGlobalGetStatus ........................................................ 20-16
20.17. AerStripGlobalGetTriggerPoint ............................................... 20-17
20.18. AerStripGlobalGetTriggerStatus .............................................. 20-18
20.19. AerStripGlobalHalt ................................................................. 20-19
20.20. AerStripGlobalHold ............................................................... 20-20
20.21. AerStripGlobalRelease ............................................................ 20-21
20.22. AerStripGlobalSetTrigger ......................................................... 20-22
20.23. AerStripGlobalSetTriggerPoint ................................................ 20-23
20.25. AerStripSetIOPosLatch ............................................................. 20-25
20.26. AerStripSetTrigger ................................................................. 20-26
20.27. AerStripTrigger ................................................................. 20-27

CHAPTER 21:  SYSTEM FUNCTIONS .................................................. 21-1
21.1.  Introduction .............................................................................. 21-1
21.2.  AerSysClose ................................................................. 21-2
21.3.  AerSysCloseAll ................................................................. 21-3
21.4.  AerSysDownload ................................................................. 21-4
21.5.  AerSysFaultAck ................................................................. 21-5
21.6.  AerSysGetDeviceID .............................................................. 21-6
21.7.  AerSysInitSystem ................................................................. 21-7
21.8.  AerSysOpen ................................................................. 21-8
21.9.  AerSysOpenEx ................................................................. 21-9
21.10. AerSysReset ................................................................. 21-10

CHAPTER 22:  TASK FUNCTIONS .......................................................... 22-1
22.1.  Introduction .............................................................................. 22-1
22.1.1.  Task Callback Functions .................................................. 22-2
22.1.2.  Extending the CNC with a CALLDLL Statement .............. 22-2
22.2.  AerTaskCallbackContinue .................................................... 22-3
Table of Contents

CHAPTER 22: TASK FUNCTIONS ................................................................. 22-1

22.1. Introduction ...................................................................................... 22-1
22.2. AerTaskCallBackContinueEx .............................................................. 22-2
22.3. AerTaskCallBackGetEx ...................................................................... 22-3
22.4. AerTaskCallBackReturnSetDouble ..................................................... 22-4
22.5. AerTaskCallBackReturnSetString ....................................................... 22-5
22.6. AerTaskCallBackSetValueDouble ....................................................... 22-6
22.7. AerTaskCallBackSetValueGetString ................................................... 22-7
22.8. AerTaskCallBackValueGetDouble ..................................................... 22-8
22.9. AerTaskCallBackValueGetDWORD .................................................... 22-9
22.10. AerTaskCallBackValueGetString .................................................... 22-10
22.11. AerTaskCallBackValueMakeString .................................................. 22-11
22.12. AerTaskCallBackValueSetDWORD .................................................. 22-12
22.13. AerTaskCallBackValueSetString .................................................... 22-13
22.15. AerTaskGetPhysAxisFromTaskAxis .................................................. 22-15
22.16. AerTaskGetTaskAxisFromPhysAxis .................................................. 22-16
22.17. AerTaskImmediateBindAxis ............................................................. 22-17
22.18. AerTaskImmediateCaptureAxis ....................................................... 22-18
22.19. AerTaskImmediateFreeAxis ............................................................. 22-19
22.20. AerTaskImmediateMapAxis ............................................................. 22-20
22.21. AerTaskModeGetName ................................................................... 22-21
22.22. AerTaskProgramAbort ..................................................................... 22-22
22.23. AerTaskProgramAssociate ............................................................. 22-23
22.24. AerTaskProgramDeAssociate ......................................................... 22-24
22.25. AerTaskProgramExecute ................................................................. 22-25
22.26. AerTaskProgramGetLineUser ........................................................... 22-26
22.27. AerTaskProgramReset .................................................................... 22-27
22.28. AerTaskProgramSetLineUser ............................................................ 22-28
22.29. AerTaskProgramStop ...................................................................... 22-29
22.30. AerTaskReset .................................................................................. 22-30
22.31. AerTaskSetExecuteMode ................................................................. 22-31
22.32. AerTaskSetRetraceMode .................................................................. 22-32
22.33. AerTaskSetSlewMode ...................................................................... 22-33
22.34. AerTaskStatusGetName .................................................................. 22-34

CHAPTER 23: TOOL FUNCTIONS ............................................................... 23-1

23.1. Introduction ...................................................................................... 23-1
23.2. AerToolFileDownload ................................................................. 23-2
23.3. AerToolFileGetNumTools ............................................................... 23-3
23.4. AerToolFileGetTool ...................................................................... 23-4
23.5. AerToolFileSetTool ...................................................................... 23-5
23.6. AerToolFileAddTool ..................................................................... 23-6
23.7. AerToolFileRemoveTool ............................................................... 23-7
23.8. AerToolTableAllocate ................................................................. 23-8
23.9. AerToolTableFree .......................................................................... 23-9
23.10. AerToolTableGetNumTools ........................................................... 23-10
23.11. AerToolTableSetTool ................................................................. 23-11
23.12. AerToolTableGetTool .................................................................. 23-12

CHAPTER 24: TORQUE FUNCTIONS ......................................................... 24-1

24.1. Introduction ...................................................................................... 24-1
24.2. AerTorqueGetMode ...................................................................... 24-2
24.3. AerTorqueSetMode ...................................................................... 24-3
# U600 Series Library Manual

## Table of Contents

### CHAPTER 25: UTILITY FUNCTIONS

25.1. Introduction ................................................................. 25-1  
25.2. AerUtilAlignMaster .................................................. 25-2  
25.3. AerUtilHitWatchdogTimer ......................................... 25-3  
25.4. AerUtilMotorSet ...................................................... 25-4  
25.5. AerUtilSet1kHzServoLoop ....................................... 25-5  
25.6. AerUtilSet4kHzServoLoop ....................................... 25-6

### CHAPTER 26: VARIABLE FUNCTIONS

26.1. Introduction ............................................................. 26-1  
26.2. AerVarAnalogGetDouble .......................................... 26-2  
26.3. AerVarGlobalGetDouble ........................................... 26-3  
26.4. AerVarGlobalGetString ............................................ 26-4  
26.5. AerVarGlobalSetDouble ........................................... 26-5  
26.6. AerVarGlobalSetString ............................................ 26-6  
26.7. AerVarProgramGetDouble ....................................... 26-7  
26.8. AerVarProgramSetDouble ....................................... 26-8  
26.9. AerVarProgramSetDouble ....................................... 26-9  
26.10. AerVarStackGetDouble ......................................... 26-10  
26.11. AerVarStackSetDouble ......................................... 26-11  
26.12. AerVarStackSetDouble ......................................... 26-12  
26.15. AerVarTaskGetDouble ........................................... 26-15  
26.16. AerVarTaskGetDouble ........................................... 26-16

### CHAPTER 27: VERSION FUNCTIONS

27.1. Introduction ............................................................. 27-1  
27.2. AerVerGetBootVersion ............................................. 27-2  
27.3. AerVerGetCpuStatus ............................................... 27-3  
27.4. AerVerGetDrvVersion .............................................. 27-4  
27.5. AerVerGetImgVersion .............................................. 27-5  
27.6. AerVerGetLibVersion .............................................. 27-6  
27.7. AerVerGetOS ............................................................ 27-7  
27.8. AerVerStringToVersion .......................................... 27-8  
27.9. AerVerToString ...................................................... 27-9

### CHAPTER 28: VIRTUAL I/O FUNCTIONS

28.1. Introduction ............................................................. 28-1  
28.2. AerVirtGetBinaryInput ........................................... 28-2  
28.3. AerVirtGetBinaryInputByteEx .................................. 28-3  
28.4. AerVirtGetBinaryInputDWordEx .............................. 28-4  
28.5. AerVirtGetBinaryInputWordArray ............................ 28-5  
28.6. AerVirtGetBinaryInputWordEx ................................ 28-6  
28.7. AerVirtGetBinaryIO ................................................ 28-7  
28.8. AerVirtGetBinaryOutput ......................................... 28-8  
28.9. AerVirtGetBinaryOutputByteEx ............................... 28-9  
28.10. AerVirtGetBinaryOutputDWordEx .......................... 28-10  
28.11. AerVirtGetBinaryOutputWordArray ........................ 28-11  
## Table of Contents

AerVirtGetRegisterInputEx ............................................................ 28-15
AerVirtGetRegisterIO ................................................................. 28-16
AerVirtGetRegisterOutput ......................................................... 28-17
AerVirtGetRegisterOutputArray .................................................. 28-18
AerVirtGetRegisterOutputEx ....................................................... 28-19
AerVirtSetBinaryInput ............................................................... 28-20
AerVirtSetBinaryInputByteEx ...................................................... 28-21
AerVirtSetBinaryInputDWordEx ................................................... 28-22
AerVirtSetBinaryInputWordArray ............................................... 28-23
AerVirtSetBinaryInputWordEx ..................................................... 28-24
AerVirtSetBinaryIO ................................................................. 28-25
AerVirtSetBinaryOutput ........................................................... 28-26
AerVirtSetBinaryOutputByteEx .................................................... 28-27
AerVirtSetBinaryOutputDWordEx ............................................... 28-28
AerVirtSetBinaryOutputWordArray .............................................. 28-29
AerVirtSetBinaryOutputWordEx .................................................. 28-30
AerVirtSetRegisterInput ............................................................ 28-31
AerVirtSetRegisterInputArray .................................................... 28-32
AerVirtSetRegisterInputEx ........................................................ 28-33
AerVirtSetRegisterIO .................................................................. 28-34
AerVirtSetRegisterOutput .......................................................... 28-35
AerVirtSetRegisterOutputArray .................................................. 28-36
AerVirtSetRegisterOutputEx ....................................................... 28-37

## APPENDIX A:

### CONSTANTS FOR C LANGUAGE AND LABVIEW USERS

| A.1 | Description .......................................................... | A-1 |
| A.2 | AER_BP ................................................................. | A-2 |
| A.3 | AER_CARD ............................................................... | A-3 |
| A.4 | AER_CPUSTAT .......................................................... | A-4 |
| A.5 | AER_DRV600_DEFAULT ................................................ | A-5 |
| A.6 | AER_EVENT ............................................................... | A-6 |
| A.7 | AER_FBTYPE_XXXX .................................................... | A-7 |
| A.8 | AER_IOTYPE_XXXX ..................................................... | A-8 |
| A.9 | AER_PROBESTATUS_XXXX .............................................. | A-9 |
| A.10 | AER_MFBTYPE_XXXX .................................................. | A-10 |
| A.11 | AER_SPARETYPE1_XXXX ............................................... | A-11 |
| A.12 | AER_SPARETYPE2_XXXX ............................................... | A-12 |
| A.13 | AER_UNIDEX ............................................................ | A-13 |
| A.14 | AERCAM ................................................................. | A-14 |
| A.15 | AERCMLR ................................................................. | A-15 |
| A.16 | AERERR_TYPE .......................................................... | A-16 |
| A.17 | AERREGID ............................................................... | A-17 |
| A.18 | AERTOOL ................................................................. | A-18 |
| A.19 | ALT_STAT ................................................................. | A-19 |
| A.20 | ANALOGINDEX .......................................................... | A-20 |
| A.21 | AXISINDEX ............................................................... | A-21 |
| A.22 | AXISMASK ............................................................... | A-22 |
| A.23 | AXISPARM ............................................................... | A-23 |
| A.24 | AXISTYPE ................................................................. | A-26 |
| A.25 | CSPARMINDEX .......................................................... | A-27 |
APPENDIX B: VISUAL BASIC CONSTANTS

B.1. Description ................................................................. B-1
B.2. aerAnalogIndex ......................................................... B-2
B.3. aerAnalogMask .......................................................... B-3
B.4. aerAutoProgSystem .................................................. B-4
B.5. aerAutoProgType ....................................................... B-5
B.6. aerAxisIndex ............................................................ B-6
B.7. aerAxisMask ............................................................. B-7
B.8. aerAxisParm ............................................................. B-8
B.9. aerAxisState ............................................................. B-11
B.10. aerAxisType ............................................................ B-12
B.11. aerBit ................................................................. B-13
B.12. aerBrkPnt ............................................................... B-14
B.13. aerCard ................................................................. B-15
B.14. aerCompileError ..................................................... B-16
B.15. aerCPUStatus ........................................................ B-17
B.16. aerDeviceID ............................................................ B-18
B.17. aerDrv600Default .................................................... B-19
B.18. aerErrType ............................................................. B-20
B.19. aerEvent ............................................................... B-21
B.20. aerFBType ............................................................. B-22
B.21. aerFLT ................................................................. B-23
B.22. aerGStripMode ....................................................... B-24
Table of Contents

B.23. aerGStripStatus ................................................................. B-25
B.24. aerGStripLogicMode .......................................................... B-26
B.25. aerGStripLogic ................................................................. B-27
B.26. aerGlobalParm ................................................................. B-28
B.27. aerImgExecuting ............................................................... B-29
B.28. aerIOType ........................................................................... B-30
B.29. aerMachParm ................................................................. B-31
B.30. aerMax ................................................................. B-32
B.31. aerMaxVirtIO ................................................................. B-33
B.32. aerMFBType ................................................................. B-34
B.33. aerMove ................................................................. B-35
B.34. aerNoErr ................................................................. B-36
B.35. aerOS ................................................................. B-37
B.36. aerParm ................................................................. B-38
B.37. aerParmAttr ................................................................. B-39
B.38. aerParmDisplay .............................................................. B-40
B.39. aerParmType ................................................................. B-41
B.40. aerPhysAxisIndex ............................................................ B-42
B.41. aerProbeStatus ............................................................... B-43
B.42. aerProgramMsg ............................................................... B-44
B.43. aerPSOChannelMask ........................................................ B-45
B.44. aerPSOMode ................................................................. B-46
B.45. aerPSOTable ................................................................. B-47
B.46. aerPtrType ................................................................. B-48
B.47. aerPulseWidth ................................................................. B-49
B.48. aerRegID ................................................................. B-50
B.49. aerReturnType ................................................................. B-51
B.50. aerRIO ................................................................. B-52
B.51. aerServoStat ................................................................. B-53
B.52. aerStat ................................................................. B-54
B.53. aerStripIOPosLatch ........................................................ B-55
B.54. aerStatus ................................................................. B-56
B.55. aerSP1Type ................................................................. B-57
B.56. aerSP2Type ................................................................. B-58
B.57. aerTaskAxisIndex ........................................................... B-59
B.58. aerTaskExec ................................................................. B-60
B.59. aerTaskIndex ................................................................. B-61
B.60. aerTaskMask ................................................................. B-62
B.61. aerTaskMode ................................................................. B-63
B.62. aerTaskParm ................................................................. B-64
B.63. aerTaskStatus ............................................................... B-67
B.64. aerTool ................................................................. B-68
B.65. aerVersion ................................................................. B-69
B.66. Q ................................................................. B-70

APPENDIX C: STRUCTURES AND DATA TYPES (C LANGUAGE) ................. C-1

C.1. Description ........................................................................... C-1
C.2. AER_AUX_POINT ............................................................... C-2
C.3. AER_AXISCAL_PACKET .................................................... C-3
C.4. AER_AXISCAL_STATUS_PACKET ........................................ C-4
C.5. AER_AXIS_DATA_EX ........................................................... C-5
C.6. AER_CAM_GETPOINT ................................................................. C-6
C.7. AER_CAM_SETPOINT ............................................................... C-7
C.8. AER_CFG_FBPACKET_ENCODER ............................................. C-8
C.9. AER_CFG_FBPACKET_ENCODER_HALL .................................. C-9
C.10. AER_CFG_FBPACKET_HPVME ................................................. C-10
C.11. AER_CFG_FBPACKET_NULL ................................................. C-11
C.12. AER_CFG_FBPACKET_RESOLVER ....................................... C-12
C.13. AER_CFG_FBPACKET_RESOLVER_HALL .............................. C-13
C.14. AER_CFG_FBPACKET_STEPPER ......................................... C-14
C.15. AER_CFG_IOPACKET_D2A .................................................. C-15
C.16. AER_CFG_MASTER_PACKET ............................................... C-16
C.17. AER_CFG_PACKET .............................................................. C-17
C.18. AER_CFG_SPI_PACKET_ENCODER ....................................... C-19
C.19. AER_CFG_SPI_PACKET_RESOLVER ..................................... C-20
C.20. AER_COMPILE_ERROR_DATA ............................................. C-21
C.21. AER_COMPILE_STATUS_DATA ............................................ C-22
C.22. AER_GSTRIP_AXIS_DATA .................................................... C-23
C.23. AER_GSTRIP_SAMPLE ....................................................... C-24
C.24. AER_GSTRIP_SYSTEM_DATA .............................................. C-25
C.25. AER_MOTN_LINEAR_PACKET ............................................. C-26
C.26. AER_MOTN_PACKET ........................................................... C-27
C.27. AER_PARM_INFO ............................................................... C-28
C.28. AER_PROBE_POS_PACKET ................................................ C-29
C.29. AER_PROBE_STATUS_PACKET .......................................... C-30
C.30. AER_PROFILE ................................................................. C-31
C.31. AER_PROG_FAULT ............................................................. C-32
C.32. AER_PROG_HANDLE .......................................................... C-33
C.33. AER_PROG_HEADER .......................................................... C-34
C.34. AER_PROG_INFO .............................................................. C-35
C.35. AER_PROG_LABEL ............................................................. C-36
C.36. AER_PROG_LABEL_INFO .................................................... C-37
C.37. AER_PROG_STATUS ........................................................... C-38
C.38. AER_PROG_STATUS ........................................................... C-39
C.39. AER_PSO_D2A ................................................................. C-40
C.40. AER_PSO_D2A_PACKET ..................................................... C-41
C.41. AER_PULSE_DEFINE .......................................................... C-42
C.42. AER_PULSE_DEFINE_RAMP .............................................. C-43
C.43. AER_PULSE_WIDTH ........................................................... C-44
C.44. AER_REG_DEVICE_INFO ................................................... C-45
C.45. AER_STRIP_SAMPLE ........................................................ C-46
C.46. AER_TASK_DATA ............................................................. C-47
C.47. AER_TASK_MODE ............................................................. C-48
C.48. AER_TASK_STATUS ........................................................... C-49
C.49. AER_TORQ_PACKET .......................................................... C-51
C.50. AER_U600_INFO ............................................................... C-52
C.51. AER_U631_INFO ............................................................... C-53
C.52. AER_UNIDEX_INFO .......................................................... C-54
C.53. AER_VERSION ................................................................. C-55
C.54. AERERR_CODE ............................................................... C-56
C.55. AERVIRT_BINARY_DATA .................................................. C-57
C.56. AERVIRT_REGISTER_DATA .............................................. C-58
LIST OF FIGURES

Figure 5-1. Master/Slave Profile ................................................................. 5-2

Figure 12-1. Graph of Position and Speed ............................................... 12-22
LIST OF TABLES

Table 4-1.  Axis Calibration.................................................................................. 4-3
Table 10-1.  Axis Processor Generated Interrupts ............................................. 10-3
Table 12-1.  AerMove Functions ........................................................................ 12-7
Table 12-2.  AerMoveMulti Functions ................................................................ 12-20
Table 13-1.  AerParm Groups (see constants below).......................................... 13-1
Table 19-1.  Device Default Values................................................................. 19-2
Table 20-1.  STRIP_IOPOSLATCHXXXX (aerStriplOPosLatchXXXX) Constants ................................................................................ 20-27
Table R-1.  Revisions ....................................................................................... R-1
PREFACE

The Preface gives the reader an overview of topics covered in each of chapter and conventions used in this manual. The following topics are contained in this manual:

CHAPTER 1: INTRODUCTION

Chapter 1 contains an overview of the UNIDEX 600 reference library manual.

Section 1.4. is a Visual Basic Quick Start guide.
Section 1.5. is a C Language Programming Quick Start section.
Section 1.6. is a LabView Programming Quick Start section.

CHAPTER 2: AUTOMATION PROGRAM FUNCTIONS

This chapter describes the routines used within the MMI600 for the Program Automation feature (i.e., the auto running and auto including programs).

CHAPTER 3: AUXILIARY TABLE FUNCTIONS

This chapter contains information about the auxiliary table functions that fire the auxiliary outputs based on the position of a given axis. Provided with each function is the prototype and a list of applicable parameters.

CHAPTER 4: AXIS CALIBRATION FUNCTIONS

Information about the axis calibration functions that apply correction values to a given axis based on another axis’ position are given in Chapter 3. These functions typically produce an error look-up table from an accurate position measurement device, such as a laser interferometer, to correct inaccuracies of a mechanical device, like a ball screw. Provided with each function is the prototype and a list of applicable parameters.

CHAPTER 5: ELECTRONIC CAMMING FUNCTIONS

This chapter supplies information about the electronic camming functions that allow the user to synchronize slave axes to the motion of a master axis. Provided with each function is the prototype and a list of applicable parameters.

CHAPTER 6: COMPILER FUNCTIONS

Chapter 5 provides information on the compiler functions that accept CNC program lines and entire programs. Provided with each function is the prototype and a list of applicable parameters.

CHAPTER 7: AXIS CONFIGURATION FUNCTIONS

This chapter contains information about the axis configuration functions that allow the controller’s axes to be configured for various resolutions and feedback devices. Provided with each function is the prototype and a list of applicable parameters.

CHAPTER 8: DATA CENTER FUNCTIONS

Functions for retrieving blocks of axis and task data are detailed in the Data Center functions chapter. Provided with each function is the prototype and a list of applicable parameters.
CHAPTER 9:  ERROR FUNCTIONS

This chapter contains information on the error functions that allow the programmer to access ASCII error strings for the defined error codes. Provided with each function is the prototype and a list of applicable parameters.

CHAPTER 10:  EVENT FUNCTIONS

Provided in this chapter is information on event functions that the UNIDEX 600 series controllers use to communicate back to the application running on the host PC. Provided with each function is the prototype and a list of applicable parameters.

CHAPTER 11:  MEMORY FUNCTIONS

This chapter contains information on the memory functions that read, write, or query memory on the axis processor card. Provided with each function is the prototype and a list of applicable parameters.

CHAPTER 12:  MOVE FUNCTIONS

Contained in this chapter is information on the move functions that implement basic motion and homing tasks. Provided with each function is the prototype and a list of applicable parameters.

CHAPTER 13:  PARAMETER FUNCTIONS

This chapter contains information on the parameter functions that perform the reading and writing of parameter values. Provided with each function is the prototype and a list of applicable parameters.

CHAPTER 14:  PROBE FUNCTIONS

The probe functions give the programmer access to an input that collects position information. These functions are detailed in Chapter 13. The prototype and a list of applicable parameters are also given.

CHAPTER 15:  PROFILE FUNCTIONS

The Profiling Functions give the programmer low-level access to derive their own profile motion. Provided with each function is the prototype and a list of applicable parameters.

CHAPTER 16:  PROGRAMMING ERROR FUNCTIONS

This chapter provides information on the program functions that return data regarding programming errors or ASCII strings. Provided with each function is the prototype and a list of applicable parameters.

CHAPTER 17:  PROGRAM FUNCTIONS

This chapter contains information on functions that allow the caller to obtain information about, and set associated parameters for a program already residing on the axis processor card.
CHAPTER 18: PSO (LASER) FUNCTIONS
Information on the PSO laser functions, which permit Aerotech’s Position Synchronized Output (PSO) card to be configured for tracking up to three axes of motion and generating an output signal to fire the laser, are provided in Chapter 17. Provided with each function is the prototype and a list of applicable parameters.

CHAPTER 19: REGISTRY FUNCTIONS
This chapter contains information on the registry functions that are used to configure the UNIDEX 600 series controllers for either Windows 95 or Windows NT Operating System. Provided with each function is the prototype and a list of applicable parameters.

CHAPTER 20: STRIP CHARTING FUNCTIONS
This chapter contains information on the strip charting functions that provide an interface for single or multiple axis data collection. Provided with each function is the prototype and a list of applicable parameters.

CHAPTER 21: SYSTEM FUNCTIONS
This chapter provides information on the system functions used to initialize, configure, and control the UNIDEX 600 series controllers. Provided with each function is the prototype and a list of applicable parameters.

CHAPTER 22: TASK FUNCTIONS
This chapter contains information about functions that allow the user to execute and control the execution of CNC programs already residing on the axis processor.

CHAPTER 23: TOOL FUNCTIONS
This chapter contains information about functions that are responsible for managing Tool Files and Tool Tables on the axis processor.

CHAPTER 24: TORQUE FUNCTIONS
Contained in this chapter is information on the torque functions that allow constant torque to be maintained on an axis independent of speed. Provided with each function is the prototype and a list of applicable parameters.

CHAPTER 25: UTILITY FUNCTIONS
This chapter contains information on the utility functions that are a miscellaneous group of functions. Provided with each function is the prototype and a list of applicable parameters.

CHAPTER 26: VARIABLE FUNCTIONS
This chapter contains information about variable functions that read and write to the user variables within the UNIDEX 600 Series controllers.
CHAPTER 27: VERSION FUNCTIONS

This chapter contains information on the version functions that provide information about the current version of the UNIDEX 600 series controller firmware and library (AerSys.DLL) executing on the system. Provided with each function is the prototype and list of applicable parameters.

CHAPTER 28: VIRTUAL I/O FUNCTIONS

This chapter contains information on the Virtual I/O functions that read and write to the virtual I/O of the UNIDEX 600 Series controllers.

APPENDIX A: CONSTANTS FOR C LANGUAGE AND LABVIEW USERS

Appendix A contains definitions of all the constants used within the library's (AerSys.DLL).

APPENDIX B: VISUAL BASIC CONSTANTS

Appendix B contains definitions of all the constants used within the library's (AerSys.DLL).

APPENDIX C: STRUCTURES AND DATA TYPES

Appendix C contains definitions of all the structures used within the library (AerSys.DLL).

APPENDIX D: WARRANTY AND FIELD SERVICE

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

INDEX

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

REVISION HISTORY

This section lists the changes made for the current revision.
CUSTOMER SURVEY FORM

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

Throughout this manual the following conventions are used:

- The terms UNIDEX 600, UNIDEX 620, U600, and U620 are used interchangeably throughout this manual.
- Most functions have a “C” and Visual Basic (VB) prototype below them and a symbol in the outer margin indicates which type it is, see right margin.
- All Aerotech functions available to the user have the prefix “Aer.” The word following the prefix refers to the logical group the function belongs (e.g., all configuration functions are prefixed with “AerConfig”.
- All Aerotech functions that are only intended for internal use have a prefix of “aer” (e.g., the function AerCamTableGetStatus function calls the internal function aerCamTableGetStatusPacket. These functions may or may not be retrievable from outside the AERSYS.DLL depending on the context.
- Some cases in the manual refer to a whole group of functions using the suffix “xxxx” (e.g., AerCmplrxxxx) to indicate generality. See AerSyncxxxx functions, means that the text is relevant to all functions beginning with “AerSync.”
- *Italic font* is used to illustrate functions and parameters (e.g., AerCamTableGetStatus, FAULTMASK, AERERR_CODE AerAxisCalSet( HAERCTRL hAerCtrl, AXISINDEX iAxis, WORD wTable, DWORD dwInput, DWORD dwScale );).
- “Axis processor” refers to the Intel 80960 RISC processor within UNIDEX 600, and the 620 system that perform all the motion and CNC program execution.
- “Front-end” is used to refer to any application running on the host PC that communicates with the axis processor.
- This manual uses the symbol "∇ ∇ ∇" to indicate the end of a chapter.

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

∇ ∇ ∇
CHAPTER 1: INTRODUCTION

In This Section:
- Overview .......................................................... 1-1
- Building ............................................................ 1-4
- Calling .............................................................. 1-5
- Visual Basic Programming Quick Start ............... 1-6
- C/C++ Language Programming Quick Start .... 1-7
- LabView Programming Quick Start .................... 1-8
- Debugging ........................................................ 1-9
- Error Handling ............................................... 1-11
- Examples ........................................................ 1-14

1.1. Overview

MS Visual Basic and LabView programmers are strongly recommended to purchase our SDK600-NT, instead of using the library calls contained in this manual. Using the SDK600-NT will simplify and speed up the application development process. These ActiveX objects are proven integral parts of Aerotech’s MMI600 CNC application and utilities. For documentation on the SDK600-NT, visit our website at www.aerotech.com and download the U600 SDK help file.

The UNIDEX 600 software is designed to run in Microsoft’s Win32 environment. This includes Windows 95 and Windows NT. No effort has been made to support Win32s. Due to the nature of Win32, a device driver is necessary for hardware communications. The basic components necessary for running an application on the U600 Series controller includes the following: an image (or firmware) file, a device driver, and the Aerotech System DLLs. The image file is the operating system of the controller. The device driver is responsible for communications between the PC and controller. The System DLLs are responsible for communications between the user application and device driver. The image files and Aerotech System DLLs are operating system independent; the device driver is operating system dependent.

As a programmer, the Aerotech System DLL is the component that is used for all communications. This is the only interface available to the programmer. In fact, it is the ONLY interface that Aerotech uses to communicate to the U600 Series controller. It is the building block of all U600 software components, including the software development kit (SDK600-NT).

The U600 software does not have any built in multitasking limitations. Any Aerotech utility program, the U600 MMI, and any user application should be able to be run together. The controller only relies on the device driver and system libraries to feed it commands. Once a command is sent to the controller, it will run until completion unless a fault occurs or otherwise directed by the user.
1.1.1. Developing an Application

The following steps are necessary for developing an application:

1. **Install the UNIDEX 600 Series Libraries and Utilities.**
   
The disks have all the necessary DLLs, libraries, image files, and device drivers. They also have all the example programs that are referenced in this manual. The installation program also configures the registry.

2. **Develop the application, with error handling.**

   **Error Codes**

   All functions return an AERERR_CODE. This is a 32 bit value. See Section 1.8.: Error Handling and Chapter 8: Error Functions for more detail. These may indicate programming faults, but do not indicate axis faults (see step 6 below and Section 1.8.3: Faults).

   The return code for the library functions should always be tested for errors.

3. **Establish communications.**

   Every application needs to establish communications with the device driver. This is done by calling AerSysOpen. This establishes the link between the libraries, device driver, and image files. This function returns a handle, referred to as an HAERCTRL (read H-Aer-Control), which is passed to all functions that require communications to the device driver and ultimately the controller.

4. **Begin execution of the image (firmware), and configure the controller.**

   The firmware is the operating system of the controller. It is required for execution before any commands are sent to the controller. This is accomplished by calling AerSysInitSystem.

   It is not necessary to download the image file each time an application starts up. It is only required once per power-up. This allows for configuration and setup from utility applications at various stages if desired. In fact, the download will fail if the image is already executing. See AerSysReset to halt execution of the image file.

   Configure the controller via its .INI files stored in the \U600\INI folder. The U600 Series User's Guide, P/N EDU157 contains additional information on the steps involved in axis configuration. See the Axis Configuration and Parameter Functions in this manual to setup an axis programmatically.

5. **Make motion.**

   At this point, the user has many options in making motion.

   For asynchronous motion see the Move Functions.

   For executing CNC G-Code programs see the Compiler, Program, and Task Functions.

   For master-slave motion see the Electronic Camming Functions.

Most status information is available through the various parameters. This includes feedback information (position, velocity, position command, etc.) and fault information. The *U600 Series User’s Guide, P/N EDU157* contains detailed information on all the available parameters. See the *Parameter Functions* for details on getting and setting these values.

Fault conditions can be monitored by polling of the FAULT axis parameter, TaskFault task parameter, or the use of a callback event. For more general information on the callback mechanism, see the *U600 Series User’s Guide, P/N EDU157*. See the *Event Functions* for programming information.

I/O can be manipulated by using the *Virtual I/O Functions*. These allow for integrating foreign I/O into the controller.

To manipulate variables associated with a CNC program, see the *Variable Functions*.

7. Close communications.

When finished, do not forget to close the communications channel. This will ensure that system resources are properly cleaned up. See *AerSysClose* for more details.

8. Redistribute the application and the registry.

After creating the application, the user may have to move it to another machine. Keep in mind that not only are Aerotech DLLs, image files, and device drivers needed, but the user also needs to configure the registry.

The AerReg utility allows the user to setup or change the system registry. This utility can be used to configure machines appropriately or see the *Registry Functions* to configure the registry programmatically.

Although the information can be manipulated directly through the operating system or with the Win32 Registry functions, we strongly recommend using the provided functions or utilities. The UNIDEX 600 registry is setup differently between Windows NT and Windows 95, these functions automatically take into account these differences.
1.2. Building

The user should ALWAYS build applications with exception handling enabled, so the routines can properly detect errors. The user must include the Aerotech-provided header file: AERSYS.H in files that call Aerotech functions. This header file will include all other Aerotech provided header files that must be in the same directory as the AERSYS.H file. The user must link with the Aerotech provided libraries AERSYS.LIB and AERERR.LIB. In addition, if making AerCompilerxxxx calls, the user must also link with AERCMPLR.LIB.

At execution time, the Aerotech provided DLLs AERSYS.DLL and AERERR.DLL must be within the system file search path. In addition, if making “AerCompilerxxxx” calls, the user must have the files: AERCMPLR.DLL, SSSCAN.DFA, and SSSCAN.LLR files within the path also.

The example programs referenced in this manual are distributed with the software and installed in an “examples” subdirectory in the user’s machine.
1.3. Calling

Almost all Aerotech functions require the first parameter to be of type HAERCTRL. This is a handle to the controller. These functions send and sometimes receive data from the controller. The user must use an AerSysOpen call to obtain a valid HAERCTRL before calling any function requiring controller communication. However, some functions do not communicate to the controller (for example AerErrGetMessage, which returns an error string for the specified error code). These functions do not have an HAERCTRL argument.
1.4. Visual Basic Programming Quick Start

Using Aerotech’s defines within VisualBasic, requires adding the Aerotech Type Library to VisualBasic. This is done by selecting the Project Menu and the References Menu selection. Many Aerotech functions require these defines.

If the Aerotech System Type library is shown, click its checkbox, then click OK. Otherwise, click the browser button and locate AerSys.dll, usually found in the \U600\Bin folder – click its check box and click OK.

You may now press F2 in VisualBasic or select the Object Browser from the View menu to see the programming defines within he Aerotech System Type Library.

1.4.1. Developing an Application

See \U600\Samples\Lib\VisualBasic\RunPgm.VBP

See \U600\Samples\ReadMe.Txt for the latest information on examples. Be sure to read Section 1.7 Debugging for information on debugging your application.
1.5. **C/C++ Language Programming Quick Start**

Add the following folders to your include file path:

\U600\Include
\U600\ACL (C++ Only)

Add the following files to your lib file path:

\U600\Lib

1.5.1. **Developing an Application**

\U600\Samples\Lib\AexAux\*.c \ auxiliary tables
\U600\Samples\Lib\AexCal\*.c \ axis calibration
\U600\Samples\Lib\AexCam\*.c \ electronic camming
\U600\Samples\Lib\AexCfg\*.c \ axis configuration
\U600\Samples\Lib\AexCmplr\*.c \ compiling a CNC program
\U600\Samples\Lib\AexHand\*.c \ handwheel
\U600\Samples\Lib\AexMove\*.c \ AerMove functions
\U600\Samples\Lib\AexProbe\*.c \ touch probe
\U600\Samples\Lib\AexProf\*.c \ profile queue
\U600\Samples\Lib\AexProg\*.c \ run a CNC program
\U600\Samples\Lib\AexPSO\*.c \ (PSO) laser firing
\U600\Samples\Lib\AexStrip\*.c \ strip charting (AerTune, AerPlot)
\U600\Samples\Lib\AexSys\*.c \ controller initialization
\U600\Samples\Lib\AexTask\*.c \ program task control
\U600\Samples\Lib\AexTool\*.c \ tool tables
\U600\Samples\Lib\AexVirt\*.c \ virtual I/O (binary I/O)

See \U600\Samples\ReadMe.Txt for the latest information on examples.

Be sure to read Section 1.7 Debugging, for information on debugging your application.

1.5.2. **C++ Language Example Programs**

\U600\Samples\Lib\AexEvent\*.cpp
\U600\Samples\AerCBack\*.cpp

See \U600\Samples\ReadMe.Txt for the latest information on examples.

Be sure to read Section 1.7 Debugging for information on debugging your application.
1.6.  LabView Programming Quick Start

LabView programmers should use the C language function definitions and their respective constants in the C Language Constants Appendix.

1.6.1.  LabView Library Example Programs

These example .VI’s illustrate to the end user, how to call Aerotech library functions from within LabView. You will notice that all of the Library .vi names correspond to a function call in the Library Reference manuals, EDU156, which is the reference point for all of the .vi’s and their parameters.

There are many examples in the \U600\Samples\LabView\Lib folder. Open one of the following .vi’s as a starting point for the example’s:

Simple .vi’s:
- ExGlobalVarReadWrite.vi ; Read/write global variable’s
- ReadWriteRegisters.vi ; Read/write registers
- ReadWriteDoubleandString.vi ; Read/write globals & strings

Complex .vi’s:
- Aer_Control.vi ; Main .vi that allows the 30 AerXxx library functions to be executed (called)

The examples in the \U600\Samples\LabView\SDK folder require the SDK600 software package, but, are simpler and easier to use than those in the Lib folder, which is why they are recommended, particularly for new programmers.

See \U600\Samples\ReadMe.Txt for the latest information on examples.

Be sure to read Section 1.7 Debugging for information on debugging your application.
1.7. Debugging

MS Visual Basic and LabView programmers are strongly recommended to purchase our SDK600-NT, instead of using the library calls contained in this manual. Following this suggestion will simplify and speed up the application development process. These ActiveX objects are proven integral parts of Aerotech’s MMI600 application and utilities. For documentation on the SDK600-NT, visit our website at www.aerotech.com and download the U600 SDK help file.

Debugging User Applications (VB, C, C++) and LabView

1. To debug your application, we recommend purchasing our MMI600 CNC application and/or using the AerDebug.exe and AerStat.exe utilities provided with UNIDEX 600, to verify the operation of your application. See below for more details.

2. There are a number of different types of errors that the controller may return. Such as:

   a. CNC Compiler errors - A CNC program has a syntax error, use the AerCompilerErr() functions to determine the cause of the error(s). Or, use the MMI600 to compile the program manually, and view the errors. Aerdebug.exe will also compile programs and show errors (see AerDebug.exe quick guide below).

   b. Return Code errors - These come back as an error code in a SDK/library call. They occur when an error is detected before the command could be executed by the U600. It is very important to examine the return code from the Aerotech SDK/library calls to determine problems within your application! You can translate any error code into text, by using AerErrGetMessage() within your application, or use the following method in AerDebug.exe:

   For example, the error number 3758628868 (or 0xE0082004) may be converted to text within AerDebug as follows:

   \[
   \text{SET T TASKFAULT 3758628868} \quad \text{SET T TASKFAULT 0xE0082004} \\
   \text{TSKI} \quad \text{TSKI} \\
   \text{: decimal format} \quad \text{: hexadecimal format} \\
   \text{: display task information} \quad \text{: display task information}
   \]

   Then the TSKI command will display a text message for the numeric error, as follows:

   ..
   Fault: 0xE0082004 ERROR: Drive is disabled
   ..

   c. Axis/Task faults - These occur during command execution. The MMI600 will indicate these errors on the manual/run pages. You can also use the TSKI AerDebug command, to view axis/task faults. But AerStat.exe, will graphically display the axis faults in detail (select the "FAULT" tab). See the U600 help file, under "faults" for more details on when axis/task faults occur.
3. The MMI600 offers functionality in an easy-to-understand Windows format. AerDebug is equally as functional as a debugger for your application, but AerDebug.exe is a command line interface, and so you need to know the commands.

1.7.1. **AerDebug**

The AerDebug application is the main debugging tool available to the user. Virtually every library function capability, including direct read/write access to the controller memory, can be tested in AerDebug. AerDebug is a "console" Windows application, operated by simple one-line commands. It can safely run simultaneously with any other application that communicates with the controller. Refer to the *U600 Series User’s Guide, P/N EDU157*, for a cross listing of which AerDebug commands execute which library functions. Here is a quick guide to the most useful AerDebug commands:

- **RDO** - Resets the U600 controller, downloads the image.
- **Q** - Quits AerDebug.exe
- **TSKI** - Shows useful status and axis/task faults for the "current" task
- **SET/GET** - For getting and setting parameter values for the "current" axis or task.
- **TKn** - Changes "current" task to "n" (AerDebug always starts in task 1, so you don't need this if you only use task 1)
- **Axn** - Changes "current" axis to "n" (AerDebug always starts in axis 1)
- **DIR** - Shows names of all programs currently on the controller, and their status.
- **EXEP** - Compiles, downloads, associates and runs a program on the current task. (see the commands below for manually and sequentially executing these steps)
- **PRGC** - Compiles a program.
- **PRGE** - Shows compile errors in a program (if any) after a compile.
- **PRGL** - Downloads a program.
- **TSKI** - Associates a program to a task
- **TSKP** - Runs an associated program.
1.8. Error Handling

Error processing from Aerotech library functions falls under three broad categories: Error Returns, Programming Errors, and Faults. This subject is complex due to the presence of two processors (the host PC and the controller) and the many options available.

1.8.1. Error Returns

All Aerotech library functions return an error code of type “AERERR_CODE” (this and all other AERERR_xxxx constants are defined in AERCODE.H and in the appendices of this manual). The user can obtain more information for a given error code by calling the AerErrGetMessage function. All Aerotech functions return the constant: AERERR_NOERR (which is 0) for successful execution.

To get the text description of an error code, call AerErrGetMessage.
1.8.2. Programming Errors

If the library function does not communicate with the controller (first parameter is not of type HAERCTRL) then the error will be indicated in this return code. In these cases, when testing for an error, it is sufficient just to test the function return code.

However, if the function does communicate with the controller (most functions do), then the return value may indicate no error when an error actually occurred, if the programming error wait mode is false (see AerProgErrMsgWaitModeSet). Note that the default condition for the programming error wait code is true, so normally this is not an issue (see AerProgErrMsgWaitModeSet).

1.8.3. Faults

Faults are a third way errors in library calls can communicate back to the calling application (in addition to Return Codes and Programming Errors). However, the fault mechanism traps more than just bad library calls (called programming errors), in fact programming errors are only one of the sixteen types of axis faults that can occur (refer to the FLT_xxxx constants). Many error conditions (such as position error) are not directly related to any library call and cannot be considered programming errors.

Using the FAULTMASK and INTMASK axis parameters, the programmer can make the controller generate interrupts when faults occur that the front-end can detect and process, if desired. As an error detection mechanism for bad library calls, this method has a speed advantage. The library does not need to wait for function completion, since the controller finds the error. This makes maximum or “parallel” use of the two processors. However, faults may make it difficult to diagnose the cause of the library call error (the interrupt will occur at some indeterminable time after the library call completes).

If the front-end application is not set up to receive interrupts, they are ignored. Refer to the AerEvent chapter, as well as the FAULTMASK and INTMASK axis parameters, to see how to receive interrupts.

There are two types of faults: axis faults and task faults. Task faults are fatal errors in CNC program execution, such as when a CNC program tries to divide by zero. Axis faults are motion related problems, such as a position error. Please see the TaskFault task parameter for more details, or the U600MMI.hlp file.

After a task fault is detected, the front-end application may examine the TaskFault task parameter to determine which fault occurred. The TaskFault parameter is an AER960RET_xxxx error code, whose textual meaning can be retrieved via an AerErrMsgMessage call.

Axis faults may or may not affect motion or cause interrupts based on the FAULTMASK axis parameter setting. It is extremely important that the programmer makes certain that the FAULTMASK (and associated mask parameters) are set properly for the application to ensure safe operation of the system.

The user can obtain information on an axis fault by looking at the FAULT axis parameter. This parameter is a bitmask, where the bits are FLT_xxxx constants and are detailed in Appendix A and Appendix B. Please see the FAULT axis parameter for more details, or the U600MMI.hlp file.
The user has the capability of suppressing certain axis faults or specifying that certain actions be performed for certain faults, such as disabling or stopping a motor. The axis parameter \texttt{FAULTMASK} determines if a fault will be acted upon. It has the same bit definitions as the \texttt{FAULT} parameter. If a bit in \texttt{FAULTMASK} is set \texttt{TRUE} (on), that fault will be acted upon. If acted upon, then the \texttt{INTMASK} axis parameter determines whether to generate an interrupt back to the front-end application. It has the same bitmask as the \texttt{FAULT} and \texttt{FAULTMASK} parameters.

Regardless of whether an interrupt is generated, the user can specify certain emergency actions be performed when an axis fault occurs. The axis parameters \texttt{ABORTMASK}, \texttt{AUXMASK}, \texttt{BRAKEMASK}, \texttt{DISABLEMASK}, and \texttt{HALTMASK} are all bitmasks like the \texttt{FAULT} parameter and perform their action on the axis generating the fault based upon the bits set. These actions are very high speed as they occur only in the controller and require no communication to the front-end. We recommend that disabling axes based on faults be handled through these masks rather than relying on the interrupt response time of the front-end processor.

See the U600MMI.hlp file for a thorough description of Fault masks.
1.9. Examples

Example C code can be found in the \samples\LIB sub directories of U600 software. Example Visual Basic code can be found in the \U600\Samples\Lib\VisualBasic folder.
CHAPTER 2:  AUTOMATION PROGRAM FUNCTIONS

In This Section:
- Overview.............................................................2-1
- AerAutoProgGetNumPrograms .............................. 2-2
- AerAutoProgGetProgram........................................ 2-3
- AerAutoProgSetProgram....................................... 2-4
- AerAutoProgAddProgram................................. 2-5
- AerAutoProgRemoveProgram ............................... 2-6

2.1. Overview

The Automation Program functions (AerAutoProgXXX) are responsible for managing the program automation file. By default, this file is U600Auto.ini and can be queried from the system by using AerRegGetFileName. This function can be used in conjunction with the AerCompilerAutoIncludeEx and AerCompilerAutoRun functions. The U600MMI uses the functions to support its Program Automation feature.

The purpose of program automation is to be used as part of the initialization sequence to initialize any compiler handles (HCOMPILER) and to download/execute any programs necessary.

There are several different types of Automated Programs:

- INCLUDE – Program is a define file. When loaded, it is automatically included with every program that is compiled, see AerCompilerAutoIncludeEx.
- DOWNLOAD_ONLY – Program is downloaded. This is generally a subroutine file, and not actually executed.
- RUN_SILENT, RUN – Program is downloaded and then executed on the given task.
- RUN_IMMEDIATE – Program is run immediately. It can only contain valid immediate commands.
- LOAD – Program is downloaded and associated to the given task (ready to run); however, it is not executed.

The AerCompilerAutoIncludeEx and AerCompilerAutoRun functions are used to execute the programs according to their automation type.
2.2. **AerAutoProgGetNumPrograms**

AERERR_CODE AerAutoProgGetNumPrograms (LPCTSTR pszFile, PDWORD pdwNumPrograms)

Declare Function AerAutoProgGetNumPrograms Lib “AERSYS.DLL” (ByRef pszFile As String, ByRef pdwLastLineLoaded As Long) As Long

**Parameters**

- **pszFile** Automation File.
- **pdwNumPrograms** Number of programs in the automation file.

This function returns the number of programs that are in the program automation file. The name of the Automation file can be retrieved with `AerRegGetFileName`.

**C Language and LabView Constants**

- `AERREGID_AutomationFile`

**VB Constants**

- `aerRegIDAutomationFile`

**See Also**

- `AerCompilerAutoIncludeEx`
- `AerCompilerAutoRun`
- `AerRegGetFileName`
2.3. **AerAutoProgGetProgram**

AERERR_CODE AerAutoProgGetProgram ( LPCTSTR pszFile, DWORD dwProg,
  PDWORD pdwSystem, LPTSTR pszProg, PDWORD pdwType,
  PTASKMASK pmTask )

Declare Function AerAutoProgGetProgram Lib "AERSYS.DLL" (ByRef pszFile As String, ByVal dwProg As Long, ByRef pdwSystem As Long, ByRef pszProg As String, ByRef pdwType As Long, ByRef pdwTaskMask As Long) As Long

**Parameters**

- **pszFile** Automation File.
- **dwProg** Which program to retrieve from file (0-Based).
- **pdwSystem** Is this a system file or user file (see constants)
- **pszProg** Name of the program.
- **pdwType** What type of program (see constants)
- **pdwTaskMask** Which tasks the program should be applied to

This function returns the automation information for the given program.

The `pdwSystem` specifies whether the program is a system or user file (see constants). A system file identifies a file that Aerotech added to the automation file. This includes (AerParam.pgm and Mcode.pgm). These files are treated no different from any other automation files except for the fact that the U600 MMI does not allow the removing of these files from the automation file.

The name of the Automation file can be retrieved with `AerRegGetFileName`.

**C Language and LabView Constants**

- AUTOPROG_SYSTEM_XXXX
- AUTOPROG_TYPE_XXXX

**VB Constants**

- aerAutoProgTypeXXXX
- aerAutoProgSystemXXXX

**See Also**

- AerCompilerAutoIncludeEx
- AerCompilerAutoRun
- AerRegGetFileName
2.4. AerAutoProgSetProgram

AERERR_CODE AerAutoProgSetProgram (LPCTSTR pszFile, DWORD dwProg, DWORD dwSystem, LPCTSTR pszProg, DWORD dwType, TASKMASK mTask)

Declare Function AerAutoProgSetProgram Lib “AERSYS.DLL” (ByRef pszFile As String, ByVal dwProg As Long, ByVal dwSystem As Long, ByVal pszProg As String, ByVal dwType As Long, ByVal dwTaskMask As Long) As Long

Parameters

pszFile Automation File.
dwProg Which program to retrieve from file (0-Based).
dwSystem Is this a system file or user file (see constants)
pszProg Name of the program.
dwType What type of program (see constants)
dwTaskMask Which tasks the program should be applied to

This function sets the information for the specified program. An error will occur if 
dwProg is not a valid program number. A valid program number is 0..NumPrograms: 
0..NumPrograms-1 will update the specified program in the Automation File. If dwProg 
equals NumPrograms then the program is added to the Automation file with 
AerAutoProgAddProgram.

C Language and LabView Constants

AUTOPROG_SYSTEM_XXXX
AUTOPROG_TYPE_XXXX

VB Constants

aerAutoProgTypeXXXX
aerAutoProgSystemXXXX

See Also

AerCompilerAutoIncludeEx
AerCompilerAutoRun
AerRegGetFileName
2.5. **AerAutoProgAddProgram**

AERERR_CODE AerAutoProgAddProgram (LPCTSTR pszFile, DWORD dwSystem, LPCTSTR szProg, DWORD dwType, TASKMASK mTask)

Declare Function AerAutoProgAddProgram Lib “AERSYS.DLL” (ByRef pszFile As String, ByVal dwSystem As Long, ByVal pszProg As String, ByVal dwType As Long, ByVal dwTaskMask As Long) As Long

**Parameters**

- **pszFile**  
  Automation File.
- **dwSystem**  
  Is this a system file or user file (see constants)
- **pszProg**  
  Name of the program.
- **dwType**  
  What type of program (see constants)
- **dwTaskMask**  
  Which tasks the program should be applied to

This function adds a new program to the Automation File.

The name of the Automation file can be retrieved with AerRegGetFileName.

**C Language and LabView Constants**

- `AUTOPROG_SYSTEM_XXXX`
- `AUTOPROG_TYPE_XXXX`

**VB Constants**

- `aerAutoProgTypeXXXX`
- `aerAutoProgSystemXXXX`

**See Also**

- AerCompilerAutoIncludeEx
- AerCompilerAutoRun
- AerRegGetFileName
2.6. AerAutoProgRemoveProgram

AERERR_CODE AerAutoProgRemoveProgram (LPCTSTR pszFile, DWORD dwProg )

Declare Function AerAutoProgRemoveProgram Lib “AERSYS.DLL” (ByRef pszFile As String, ByVal dwProg As Long) As Long

Parameters

pszFile
Automation File.
dwProg
Which program to retrieve from file (0-Based).

This function removes the specified program from the Automation File.

The name of the Automation file can be retrieved with AerRegGetFileName.

See Also

AerCompilerAutoIncludeEx
AerCompilerAutoRun
AerRegGetFileName

∇ ∇ ∇
CHAPTER 3: AUXILIARY TABLE FUNCTIONS

3.1. Introduction

Auxiliary tables set the auxiliary (mode) binary output based on the position of a given axis. This provides the user with a very tightly coupled motion and I/O capability. The value of the auxiliary output is also reflected in the value of the AUX axis parameter. An axis is specified with the range of positions along this axis where the firing is done. With each position range specified, a logic level (0 or 1) is set to determine the state of the auxiliary output when the axis position is within that range. More details on how positions fire the auxiliary outputs is in the AerAuxTableSetPoint function description.

For appropriate information on connecting user hardware for interfacing to the auxiliary outputs, refer to the U600 Hardware Manual, P/N EDU154. The active state of the auxiliary output when Aux = 1 is determined by the IOLEVEL axis parameter.

The interface to the auxiliary tables is similar to that of the cam tables. For that reason, the axis that an auxiliary table is linked to is often called a “master axis.”

There are five basic steps required to utilize Auxiliary table output capabilities – they are:

- Connect user-supplied hardware to the auxiliary output and ensure the IOLEVEL axis parameter indicates the appropriate output for the desired logic level settings.
- Allocate space for the table (see AerAuxTableAllocate)
- Load the auxiliary table (see AerAuxTableSetPoint, and AerAuxTableSetMultPoints)
- Configure a master axis (see AerConfigMaster)
- Associate the table to an axis (see AerAuxTableSetMode)
3.2. AerAuxTableAllocate

AERERR_CODE AerAuxTableAllocate( HAERCCTRL hAerCtrl, WORD wTable,
DWORD dwSize);

Declare Function AerAuxTableAllocate Lib "AERSYS.DLL" ( ByVal hAerCtrl As Long,
ByVal wTable As Integer, ByVal dwSize As Long ) As Long

Parameters
   hAerCtrl      Handle to the controller.
   wTable        Number referencing this table.
   dwSize        Maximum number of points that will be in the table.

This function allocates storage on the controller for user auxiliary output tables. Each point in the auxiliary table requires 6 bytes of memory on the controller. Table numbers may be from zero up to the maximum number returned by AerAuxTableGetTables.

See Also
   AerAuxTableFree

Example
   samples\lib\AexAux.c
3.3. **AerAuxTableFree**

AERERR_CODE AerAuxTableFree( HAERCTRL hAerCtrl, WORD wTable );

Declare Function AerAuxTableFree Lib "AERSYS.DLL" ( ByVal hAerCtrl As Long,
    ByVal wTable As Integer ) As Long

### Parameters

- **hAerCtrl**: Handle to the controller.
- **wTable**: Table number of table to free.

This function frees the specified auxiliary table. Table numbers may range from zero up to the maximum number returned by *AerAuxTableGetTables*.

### See Also

- AerAuxTableAllocate
- AerAuxTableGetTables

### Example

```
samples\lib\AexAux.c
```
### 3.4. AerAuxTableGetMode

AERERR_CODE AerAuxTableGetMode( HAERCTRL hAerCtrl, AXISINDEX iAxis, PWORD pwTable, PWORD pwMode );

Declare Function AerAuxTableGetMode Lib "AERSYS.DLL" ( ByVal hAerCtrl As Long, ByVal iAxis As Long, ByRef pwTable As Integer, ByRef pwMode As Integer ) As Long

**Parameters**

- **hAerCtrl**
  Handle to the controller.

- **iAxis**
  Axis to get the mode of.

- **pwTable**
  Pointer to word to receive table number.

- **pwMode**
  Pointer to word to receive table mode.

This function returns the table number and mode (0=inactive, 1=active) of an auxiliary table associated with a given axis. If the axis is not associated to an auxiliary table, both the table number and mode return as zero.

**See Also**

AerAuxTableSetMode

**Example**

samples\lib\AexAux.c
3.5. **AerAuxTableGetPoint**

AERERR_CODE AerAuxTableGetPoint( HAERCTRL hAerCtrl, WORD wTable, DWORD dwPoint, PLONG plMaster, PWORD pwLevel );

Declare Function AerAuxTableGetPoint Lib "AERSYS.DLL" ( ByVal hAerCtrl As Long, ByVal wTable As Integer, ByVal dwPoint As Long, ByRef plMaster as Long, ByRef pwLevel As Integer ) As Long

**Parameters**

- `hAerCtrl`  Handle to the controller.
- `wTable` Table number.
- `dwPoint` Point number in this table.
- `plMaster` Pointer to long value to receive coordinate value (in machine counts).
- `pwLevel` Pointer to word to receive level to set the auxiliary output.

`AerAuxTableGetPoint` returns the data for a point in the specified auxiliary table. Table numbers may range from zero up to the maximum number returned by `AerAuxTableGetTables`. Point numbers may range from zero up to size-1, where size is the number of points allocated in the table.

**See Also**

`AerAuxTableSetPoint`

**Example**

samples\lib\AexAux.c
### 3.6. AerAuxTableGetMultPoints

AERERR_CODE AerAuxTableGetMultPoints (HAERCTRL hAerCtrl, WORD wTable, DWORD dwStart, DWORD dwNumPoints, PAER_AUX_POINT pPoint);

**Parameters**

- `hAerCtrl` Handle to the controller.
- `wTable` Table number.
- `dwStart` Starting point number.
- `dwNumPoints` Number of points to get.
- `pPoint` Pointer to the first structure in an array of structures that will receive the data for each point. See Appendix C: Structures for details on the AER_AUX_POINT structure.

This function returns multiple points from an auxiliary table. Table numbers may be from zero up to the maximum number returned by `AerAuxTableGetTables`. It is the users responsibility to insure that `pPoint` actually points to an array of AER_AUX_POINT structures, of size `dwNumPoints`, or unexpected results may occur.

**See Also**

- `AerAuxTableSetMultPoints`

**Example**

`samples\lib\AexAux.c`
3.7. AerAuxTableGetStatus

AERERR_CODE AerAuxTableGetStatus( HAERCTRL hAerCtrl, WORD wTable, PDWORD pdwSize, PWORD pwStatus );

Declare Function AerAuxTableGetStatus Lib "AERSYS.DLL" ( ByVal hAerCtrl As Long, ByVal wTable As Integer, ByRef pdwSize As Long, ByRef pwStatus As Integer ) As Long

Parameters

- **hAerCtrl** Handle to the controller.
- **wTable** Table number.
- **pdwSize** Pointer to double word to receive the number of points in the table.
- **pwStatus** Pointer to word to receive the Status of this auxiliary table.

Table numbers may range from zero up to the maximum number returned by AerAuxTableGetTables. If the specified table has not been allocated, or is out of range, then both the size and the status return as zero. The table status returns as one of the following values:

- 0 ; Table has not been allocated.
- 1 ; Table has been successfully allocated.

This function does not determine if the table is associated to an axis. (see AerAuxTableGetMode)

See Also

- AerAuxTableGetMode

Example

samples\lib\AexAux.c
3.8. AerAuxTableGetTables

AERERR_CODE AerAuxTableGetTables( HAERCTRL hAerCtrl, PWORD pwNumTables );

Declare Function AerAuxTableGetTables Lib "AERSYS.DLL" (ByVal hAerCtrl As Long, ByRef pwNumTables As Integer) As Long

Parameters

- **hAerCtrl**: Handle to the controller.
- **pwNumTables**: Pointer to word to hold the number of tables currently allocated.

This function returns the absolute maximum number of tables allowed on the controller. In reality, the user may have fewer tables available, based on the controller’s available memory and the number of points in each table.

| Multiple auxiliary tables per axis are allowed. |

See Also

- AerAuxTableAllocate

Example

```
samples\lib\AexAux.c
```
3.9. \textbf{AerAuxTableSetMode}

AERERR_CODE AerAuxTableSetMode( HAERCTRL hAerCtrl, AXISINDEX iAxis, 
WORD wTable, WORD wMode);

Declare Function AerAuxTableSetMode Lib "AERSYS.DLL" ( ByVal hAerCtrl As 
Long, ByVal iAxis As Long, ByVal wTable As Integer, ByVal wMode 
As Integer ) As Long

**Parameters**

\begin{tabular}{ll}
\textit{hAerCtrl} & Handle to the controller. \\
\textit{iAxis} & Axis to associate the table to. \\
\textit{wTable} & Table number. \\
\textit{wMode} & Mode to activate. \\
\end{tabular}

Auxiliary tables are associated with axes with the \textit{AerAuxTableSetMode} command. The mode of operation for multiple axes can be defined simultaneously and table numbers may be from zero up to the maximum number returned by \textit{AerAuxTableGetTables}. The mode parameter must be in the range of minus sixteen (-16) through one (1). A mode of zero (0) disassociates a table from an axis. A mode of one (1) associates the table to an axis. A value of 1 through \(-16\) associates the table and defines a binary output (0 through 15, respectively) on the U600 card to be used in place of the MODE output. Binary outputs are only supported on the U600 card, not the PSO-PC or 4EN-PC cards due to speed issues.

Auxiliary tables are not allocated to any specific axis.

**See Also**

\textit{AerAuxTableGetMode}

**Example**

```
samples\lib\AexAux.c
```
3.10. AerAuxTableSetPoint

AERERR_CODE AerAuxTableSetPoint( HAERCTRL hAerCtrl, WORD wTable, DWORD dwPoint, LONG lMaster, WORD wLevel );

 Declare Function AerAuxTableSetPoint Lib "AERSYS.DLL" ( ByVal hAerCtrl As Long, ByVal wTable As Integer, ByVal dwPoint As Long, ByVal lMaster As Long, ByVal wLevel As Integer ) As Long

Parameters

hAerCtrl            Handle to the controller.
wTable              Table number.
dwPoint              Point number in this table.
lMaster              Coordinate value (in machine counts).
wLevel               Level to set the auxiliary output.

AerAuxTableSetPoint sets a point in a particular auxiliary table. Points can be written in any order, and can be overwritten, but cannot be changed while a table is associated with a particular axis (see AerAuxTableSetMode). Table numbers may be from zero up to the maximum number returned by AerAuxTableGetTables. Point numbers may be from zero to size-1, where size is the number of points allocated in the table. Coordinate values must be monotonically increasing with point number. The specified level must be zero or one.

Whenever a table is associated with an axis and the master position is at the given coordinate value, or between the given coordinate value and the coordinate value of the next point in the table, the level for the given point is set on the auxiliary output. This means that for all master positions above the last point (the last point set by the user) the auxiliary output will be set to the level specified for the last point in the table. Similarly, for all master positions less than the coordinate value of the first point (point 0) in the table, the auxiliary output is set to the level assigned to the first point.

It is assumed that the user has defined values for all of the points that have been allocated. If this is not true, the result is unpredictable.

See Also

AerAuxTableGetPoint
AerAuxTableSetMultPoints

Example

samples\lib\AexAux.c
3.11.  AerAuxTableSetMultPoints

AERERR_CODE AerAuxTableSetMultPoints( HAERCTRL hAerCtrl, WORD wTable,
    DWORD dwStart, DWORD dwCount, PAER_AUX_POINT pPoint );

Parameters

    hAerCtrl     Handle to the controller.
    wTable      Table number.
    dwStart     Starting point number.
    dwCount     Number of points to set.
    pPoint       Pointer to the first structure in an array of structures that
                 contain the data for each point. Refer to Appendix C:
                 Structures for details on the AER_AUX_POINT structure.

This function writes multiple points to a particular table, assigning each point consecutive
point numbers starting with the point number indicated by dwStart. It is slightly faster
than setting single points. Table numbers may be from zero up to the maximum number
returned by AerAuxTableGetTables. It is the user’s responsibility to insure that pPoint
actually points to an array of AER_AUX_POINT structures of size dwNumPoints.

See Also

    AerAuxTableGetPoints
    AerAuxTableSetPoint

Example

    samples\lib\AexAux.c
3.12. Related Axis Parameters

See the U600MMI.hlp file for more information on these parameters.

**AUXOFFSET** - This parameter is added to the master position before doing the auxiliary table lookup. For example, if the table covers master positions from 0 to 360 degrees, and the actual master position is 2 degrees, the **AUXOFFSET** parameter is 3 degrees, then the controller uses the value of 5 degrees as the master position to look up in the table.

**IOLEVEL** - A bit within this parameter may be used to determine the active state of the auxiliary (mode) line.

**MASTERLENGTH** - This parameter causes the master axis position to “roll over” every **MASTERLENGTH** count. If **MASTERLENGTH** was set at 10000 counts, then the master axis position would assume the following sequence: 9998-9999-0000-0001. This would, in effect, cause the firing pattern specified by the auxiliary table to repeat every 10,000 counts. If the **MASTERLENGTH** is not specified, then the auxiliary output does not change when the master axis position is no longer within the range specified by the maximum and minimum master position table points.

**MASTERPOS** - This parameter is the position used to affect the auxiliary output. The user can direct that either the master axis’ actual position or the master axis’ commanded position be used for the slave’s **MASTERPOS** parameter (see the AerConfigMaster function).

The **MASTERPOS** is affected by the **MASTERLENGTH** parameter.
CHAPTER 4: AXIS CALIBRATION FUNCTIONS

In This Section:

- Introduction ...................................................... 4-1
- AerAxisCalAllocateTable ................................ 4-4
- AerAxisCalFileDownload ................................ 4-5
- AerAxisCalFreeTable....................................... 4-7
- AerAxisCalGet ................................................. 4-8
- AerAxisCalGetItems ................................. 4-9
- AerAxisCalGetMode .................................... 4-10
- AerAxisCalGetPacket .................................. 4-11
- AerAxisCalGetPoint ..................................... 4-12
- AerAxisCalGetRawPosition........................... 4-13
- AerAxisCalGetStatus ................................. 4-14
- AerAxisCalGetStatusPacket.......................... 4-15
- AerAxisCalRemoveAll ................................... 4-16
- AerAxisCalReset ............................................ 4-17
- AerAxisCalSet ................................................ 4-18
- AerAxisCalSetMode ...................................... 4-19
- AerAxisCalSetPoint ....................................... 4-20
- 2D Calibration ................................................ 4-21
- AerAxisCal2DAllocateTable ......................... 4-22
- AerAxisCal2DFileDownload ......................... 4-23
- AerAxisCal2DFreeTable................................ 4-24
- AerAxisCal2DGetData................................... 4-25
- AerAxisCal2DGetMode................................. 4-26
- AerAxisCal2DGetPoint.................................. 4-27
- AerAxisCal2DGetData................................. 4-28
- AerAxisCal2DSetMode.................................. 4-29
- AerAxisCal2DSetPoint .................................. 4-30
- Related Axis Parameters .......................... 4-31

4.1. Introduction

Axis calibration tables apply correction values to a given axis’ position based on its own or another axis’ position. These functions are typically used after producing an error lookup table from an accurate position measurement device such as a laser interferometer. These tables are used to correct for mechanical inaccuracies of a mechanical device like a ball screw. The lookup table will allow the mechanical device to be positioned to a much higher degree of accuracy.

The axis to which the position adjustment is made is called the “corrected” axis. The axis whose position is used to lookup a correction value is called the “master” axis. In most applications, the master axis and the corrected axis are the same, meaning the axis uses its own position to determine the correction value at that position. However, in some cases, such as in orthogonality correction, the master axis can be a different axis. In any case, the axis correction table is a series of master positions and corresponding correction values. The servo loop continually reads the current master axis position, uses this to obtain a correction value, and adds the correction value to the corrected axis’ position.
All table entries consist of a master position and a correction value, where no two table entries can have the same master position (see AerAxisCalSetPoint for more details on table entries). If the current master axis position is equal to a table entry master position, the correction value for that table entry is used as the current correction value. If, however, the current master axis position lies in-between two master positions listed in the table, then linear interpolation is used to find a correction value between the two correction values listed in the table. Finally, if the current master position lies outside of the table, that is, all table entry master positions are either greater or less than the current master axis position, then the nearest “end point” of the table is used to obtain the correction value. For example, if the lowest master position in the table is 3, with a correction value of 5, then the correction value is 5 for all master positions from 3 to minus infinity. This means that normally the user would insure zero axis calibration outside the desired range, by placing a zero calibration point on either end of the table.

Once the perturbation or correction value for the corrected axis has been determined by table lookup, this value is added onto the raw position as seen from the encoder, and used as the “actual” position of the axes. The position command (target position) is unaffected by calibration (raw position is the RAWPOS axis parameter, actual position is the POS axis parameter, and position command is the POSCMD axis parameter). Therefore, a positive correction factor will cause the axis to stop short of its commanded position, while a negative correction factor will cause motion beyond the commanded position. All corrections are bi-directional; that is, the correction value applied will be the same regardless of the sign or value of the velocity.

Note that when running under calibration, the position values are different than may be expected at first. The actual position (POS) is unchanged by calibration, but the reverse of the calibration value is reflected in the raw position (RAWPOS). This is because under negligible position error, the servo loop forces actual position to track position command, which is unaffected by the calibration. So regardless of how axis correction alters position, the servo loop will shift the position so that it equals the position command. But the raw position on the encoder will be different under axis calibration, given the same position command and a zero position error. As a result of this behavior, the velocity reported by the controller will be unchanged by axis calibration because velocity is computed from position, not raw position.

Multiple correction tables can be applied to a single corrected axis. The effect of each table is additive. Each table comes up with its own correction value independently from the other tables and the final correction value is the sum of the values from all the tables. Up to eight tables can be used to correct an axis. There is a maximum of 100 tables for the entire system.

Axis calibration is “activated” or in place immediately after the AerAxisCalSetMode call. Users can check whether axis correction is activated for a particular axis by checking the “Error mapping enabled” bit of the SERVOSTATUS Axis Parameter for that axis (use AerStat for this). Alternatively, users can access the U600 MMI and look for a “C” in the status column immediately to the right of the axis name. Note that axis calibration is used in all motion, including jogging and homing. The user must take care that the master axis is homed before activation so that the range of calibration is accurate or synchronized.
Axis calibration has no effect on a virtual axis (axes with null feedback).

There are eight steps to using axis calibration tables. These steps are listed below along with the function(s) detailing that step (Table 4-1). Note that AerAxisCalDownloadFile will perform steps 1 through 4 in one step.

Note that AerAxisCalDownloadFile will perform steps 1 through 4 in one step.

Table 4-1. Axis Calibration

<table>
<thead>
<tr>
<th>Axis Calibration Steps</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Allocate axis calibration table</td>
<td>Refer to AerAxisCalAllocateTable function.</td>
</tr>
<tr>
<td>2. Fill the table with the error data</td>
<td>Refer to AerAxisCalSetPoint or AerAxisCalDownloadFile functions.</td>
</tr>
<tr>
<td>3. Associate a table to an axis</td>
<td>Refer to AerAxisCalSet function.</td>
</tr>
<tr>
<td>4. Enable the axis calibration table</td>
<td>Refer to AerAxisCalSetMode function.</td>
</tr>
<tr>
<td>5. Perform the calibrated motion:</td>
<td></td>
</tr>
<tr>
<td>Enable the drive</td>
<td>Refer to AerParmSet function.</td>
</tr>
<tr>
<td>Home the axis</td>
<td>Refer to AerMoveHome* functions.</td>
</tr>
<tr>
<td>Execute the motion</td>
<td>Refer to AerMove* functions.</td>
</tr>
<tr>
<td>Disable the drive</td>
<td>Refer to AerParmSet function.</td>
</tr>
<tr>
<td>6. Disable the axis calibration table</td>
<td>Refer to AerAxisCalSetMode function.</td>
</tr>
<tr>
<td>7. Disassociate calibration table from axis</td>
<td>See AerAxisCalReset function.</td>
</tr>
<tr>
<td>8. Free axis calibration table</td>
<td>Refer to AerAxisCalFreeTable function.</td>
</tr>
</tbody>
</table>

Users are not allowed to set the position of an axis manually when in axis correction mode.
4.2. AerAxisCalAllocateTable

AERERR_CODE AerAxisCalAllocateTable (HAERCTRL hAerCtrl, WORD wTable, WORD wSize);

Declare Function AerAxisCalAllocateTable Lib "AERSYS.DLL" (ByVal hAerCtrl As Long, ByVal wTable As Integer, ByVal wSize As Integer) As Long

Parameters
- **hAerCtrl** Handle to the controller.
- **wTable** Error table number (first table is zero index).
- **wSize** Number of entries or points that will be in the table.

This function allocates storage space on the controller for an axis calibration table. The maximum number and size of axis calibration tables permitted depends on the amount of available memory on the axis processor. The maximum per axis is 8. There are 100 tables permitted for use among all axes.

See Also
- AerAxisCalGetStatus
- AerAxisCalGetStatusPacket
- AerAxisCalFreeTable
- AerAxisCalSetPoint
- AerAxisCalGetPoint
- AerAxisCalReset
- AerAxisCalSetMode
- AerAxisCalGetMode
- AerAxisCalGetRawPosition
- AerAxisCalSet
- AerAxisCalGetItems
- AerAxisCalGet
- AerAxisCalGetPacket

Example

Samples\LIB\AexCal\AexCal.c
4.3. AerAxisCalFileDownload

AERERR_CODE AerAxisCalFileDownload (HAERCTRL hAerCtrl, AXISINDEX iCorrectedAxis, AXISINDEX iMasterAxis, LPCTSTR pszFileName);

Declare Function AerAxisCalFileDownload Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal iCorrectedAxis As Long, ByVal iMasterAxis As Long, ByVal pszFileName As String) As Long

Parameters

hAerCtrl Handle to controller.
iCorrectedAxis Index to axis to be corrected.
iMasterAxis Master axis.
pszFileName Filename holding axis data.

This function reads an ASCII file containing calibration data, forms a calibration table, and attaches it to the given axis. The master axis is the axis from which the lookup position that is used in the table is obtained. The user can attach multiple tables to a single axis and the effect of the various tables is summed together for a final calibration value.

The format of the ASCII file is described in the paragraphs below.

See the U600MMI.hlp file for an example.

The file must consist of data lines and comment lines. A line is defined as a string of characters terminated by any number of consecutive line terminator characters. A line terminator character is a carriage return or a line feed.

Data lines are those whose first “non-whitespace” character is a “number.” All other lines are comment lines. “Whitespace” characters are the tab, formfeed, comma, or space characters. A “number” character is a digit, decimal point, or minus sign.

After the first data line is read-in, the program will continue reading in data lines until an End of File or comment line is seen. Then the file is closed.

COMMENT LINES

The file must contain comment lines, preceding the first data line, providing three pieces of information:

• Number of points in the file
• Units of the position values
• Units and scale of the calibration values

Note that the axis to which the correction is applied is not specified in the file. It is specified when the filename is given in the axis configuration wizard of the MMI600 program, or this function.

These three required header values must be preceded (on the same line) by certain keywords. These keywords may appear on any comment line (that precede the first data line) and may appear anywhere on the line. Identification of the keyword is case insensitive.
The number of target points must be preceded by the keyword “Number of Target Positions.” The first number character after this keyword will begin the number read-in as the number of target positions. The program must read-in exactly the number of data lines given by the “Number of Target Positions.”

The units of the position values must be preceded by the keyword “Pos. Value.” The first non-whitespace character that is not “(” after the keyword, begins the position units descriptor. Units descriptors can be “nanometers,” “deg,” “degrees,” “mm,” “in,” “microns,” “millimeters,” or “inches.” Identification of the position units descriptor is case insensitive.

The units and scale factor of the compensation values must be preceded by the keyword “Compensation (.” The first non-whitespace character which is not “(” after the keyword, begins the scale factor. The first non-whitespace character after the scale factor must be the compensation units descriptor. Compensation Units descriptors can be “nanometers,” ”deg,” “degrees,” “mm,” “in,” “microns,” “millimeters,” or “inches.” Identification of the compensation units descriptor is case insensitive.

DATA LINES

Each data line must have three numbers, the first of which must be an integer. All text following the first three numbers is ignored. One or more whitespace characters must separate the numbers.

The first number is the point number. It must be a positive integer. These numbers are not used.

The second number is the position value. This value is assumed to be in the units specified in the comment lines at the beginning of the file. Position values must be monotonically increasing or decreasing throughout the file but need not be evenly spaced.

The third number is the compensation value that goes with the position value. It is assumed to be in the units specified in the comment lines at the beginning of the file (i.e. if the value read-in is -2 and the compensation read-in was .25 microns, then the actual compensation value is -.5 microns). Compensation values are always measured relative to the zero compensation value, that is, the position command that would exist if no compensation were being performed.

See Also
AerAxisCalRemoveAll

Example
IniAexCal.c
U600MMI.hlp file
4.4. AerAxisCalFreeTable

AERERR_CODE AerAxisCalFreeTable (HAERCTRL hAerCtrl, WORD wTable);

Declare Function AerAxisCalFreeTable Lib "AERSYS.DLL" (ByVal hAerCtrl As Long,
          ByVal wTable As Integer) As Long

Parameters
      hAerCtrl    Handle to the controller.
      wTable     Error table number (first table is 0).

This function deallocates an axis calibration table on the axis processor, freeing the
memory. If the table is currently in use or not allocated, an error returns.

See Also
      AerAxisAllocateTable
      AerAxisCalGetStatus
      AerAxisCalGetStatusPacket
      AerAxisCalSetPoint
      AerAxisCalGetPoint
      AerAxisCalReset
      AerAxisCalSetMode
      AerAxisCalGetMode
      AerAxisCalGetRawPosition
      AerAxisCalSet
      AerAxisCalGetItems
      AerAxisCalGet
      AerAxisCalGetPacket

Example
      Samples\LIB\AexCal\AexCal.c
4.5. AerAxisCalGet

AERERR_CODE AerAxisCalGet (HAERCTRL hAerCtrl, AXISINDEX iAxis, WORD wItem, PWORD pwTable, PDWORD pdwInput, PDWORD pdwScale);

Declare Function AerAxisCalGet Lib "AERSYS.DLL" (ByVal hAerCtrl As Long, ByVal iAxis As Long, ByVal wItem As Long, ByRef pwTable As Integer, ByRef pdwInput As Long, ByRef pdwScale As Long) As Long

Parameters
- hAerCtrl Handle to the controller.
- iAxis Axis being corrected (see constants).
- wItem Item number associated with the axis (first table is index 0).
- pwTable Pointer to receive the table number.
- pdwInput Pointer to receive the axis mask of the “master” axis.
- pdwScale Pointer to receive the scale of the correction errors.

This function returns pointers to the table status for the specified axis and specified item. Note that the function, AerAxisCalGetItems, can be used to return the number of items in the table. The pwTable variable is the table number. The pdwInput variable is the “master” axis. The pdwScale variable is the scale of the axis correction error.

C Language and LabView Constants

    AXISINDEX_1 to AXISINDEX_16

VB Constants

    aerAxisIndex1 to aerAxisIndex16

See Also

    AerAxisAllocateTable
    AerAxisCalGetStatus
    AerAxisCalGetStatusPacket
    AerAxisCalFreeTable
    AerAxisCalSetPoint
    AerAxisCalGetPoint
    AerAxisCalReset
    AerAxisCalSetMode
    AerAxisCalGetMode
    AerAxisCalGetRawPosition
    AerAxisCalSet
    AerAxisCalGetItems
    AerAxisCalGetPacket

Example

    Samples\LIB\AexCal\AexCal.c
4.6. AerAxisCalGetItems

AERERR_CODE AerAxisCalGetItems (HAERCTRL hAerCtrl, AXISINDEX iAxis, PWORD pwItems);

Declare Function AerAxisCalGetItems Lib "AERSYS.DLL" (ByVal hAerCtrl As Long, ByVal iAxis As Long, ByRef pwItems As Integer) As Long

Parameters
- **hAerCtrl** Handle to the controller.
- **iAxis** Axis number (see constants).
- **pwItems** Pointer to receive the number of items or axis calibration tables active for the specified axis.

This function returns a pointer to the number of items or axis calibration tables associated to the specified axis. To determine the table number associated with an item call *AerAxisCalGetPacket*.

**C Language and LabView Constants**

```
AXISINDEX_1
to
AXISINDEX_16
```

**VB Constants**

```
aerAxisIndex1
to
aerAxisIndex16
```

**See Also**

- AerAxisAllocateTable
- AerAxisCalGetStatus
- AerAxisCalGetStatusPacket
- AerAxisCalFreeTable
- AerAxisCalSetPoint
- AerAxisCalGetPoint
- AerAxisCalReset
- AerAxisCalSetMode
- AerAxisCalGetMode
- AerAxisCalGetRawPosition
- AerAxisCalSet
- AerAxisCalGet
- AerAxisCalGetPacket

**Example**

```
Samples\LIB\AexCal\AexCal.c
```
4.7. **AerAxisCalGetMode**

AERERR_CODE AerAxisCalGetMode (HAERCTRL hAerCtrl, AXISINDEX iAxis, PWORD pwMode);

Declare Function AerAxisCalGetModeLib Lib "AERSYS.DLL" (ByVal hAerCtrl As Long, ByVal iAxis As Long, ByRef pwMode As Integer) As Long

**Parameters**

- **hAerCtrl** Handle to the controller.
- **iAxis** Axis index (see constants).
- **pwMode** Pointer to receive the state of the axis calibration tables.

This function returns the state of the axis calibration tables for the specified axis. The `pwMode` returned will be 1 if axis calibration is enabled, or 0 if it is disabled.

**C Language and LabView Constants**

- AXISINDEX_1 to AXISINDEX_16

**VB Constants**

- aerAxisIndex1 to aerAxisIndex16

**See Also**

- AerAxisAllocateTable
- AerAxisCalGetStatus
- AerAxisCalGetStatusPacket
- AerAxisCalFreeTable
- AerAxisCalSetPoint
- AerAxisCalGetPoint
- AerAxisCalReset
- AerAxisCalSetMode
- AerAxisCalGetRawPosition
- AerAxisCalSet
- AerAxisCalGetItems
- AerAxisCalGet
- AerAxisCalGetPacket

**Example**

Samples\LIB\AexCal\AexCal.c
4.8. AerAxisCalGetPacket

AERERR_CODE AerAxisCalGetPacket (HAERCTRL hAerCtrl, AXISINDEX iAxis,
         WORD wItem, PAER_AXISCAL_PACKET pEMap);

Parameters

  hAerCtrl   Handle to the controller.
  iAxis      Axis number being corrected (see constants).
  wItem      Item number associated with the axis (0 through n-1).
  pEMap      Pointer to a structure of type AER_AXISCAL_PACKET.

This function returns a pointer to a structure of type AER_AXISCAL_PACKET
indicating the table status for the specified axis. The function AerAxisCalGetItems returns
the number of items, where each item has a table associated with it. The wTable variable
within the structure is the table number. The dwInput variable within the structure is the axis
number causing the errors or the axis whose positions are specified in the axis
correction table.

C Language and LabView Constants

  AXISINDEX_1
      to
  AXISINDEX_16

See Also

  AerAxisAllocateTable
  AerAxisCalGetStatus
  AerAxisCalGetStatusPacket
  AerAxisCalFreeTable
  AerAxisCalSetPoint
  AerAxisCalGetPoint
  AerAxisCalReset
  AerAxisCalSetMode
  AerAxisCalGetMode
  AerAxisCalGetRawPosition
  AerAxisCalSet
  AerAxisCalGetItems
  AerAxisCalGet

Example

  Samples\LIB\AexCal\AexCal.c
4.9. AerAxisCalGetPoint

AERERR_CODE AerAxisCalGetPoint (HAERCTRL hAerCtrl, WORD wTable, WORD wPoint, PLONG plPosition, PLONG plCorrection, PWORD pwStatus);

Declare Function AerAxisCalGetPoint Lib "AERSYS.DLL" (ByVal hAerCtrl As Long, ByVal wTable As Integer, ByVal wPoint As Integer, ByRef plPosition As Long, ByRef plCorrection As Long, ByRef pwStatus As Integer) As Long

Parameters

- hAerCtrl: Handle to the controller.
- wTable: Pointer to the error table number (first table is 0).
- wPoint: Table entry number.
- plPosition: Pointer to receive the absolute position in machine steps.
- plCorrection: Pointer to receive the correction value at specified position, in machine steps.
- pwStatus: Pointer to the status of the error correction point.

This function retrieves an entry within the specified axis calibration table on the axis processor. The lPosition indicates the absolute position corrected in machine steps referenced to the home position (See AerMoveHomexxx functions) by the lCorrection value. The lCorrection is an absolute value in machine steps to be added to the commanded position to reach the desired absolute position. The “pwStatus” parameter will be 1 if the entry is a valid point in the table otherwise it is 0.

Axis calibration tables are not enabled until after the axis has been homed.

See Also

- AerAxisAllocateTable
- AerAxisCalGetStatus
- AerAxisCalGetStatusPacket
- AerAxisCalFreeTable
- AerAxisCalSetPoint
- AerAxisCalReset
- AerAxisCalSetMode
- AerAxisCalGetMode
- AerAxisCalGetRawPosition
- AerAxisCalSet
- AerAxisCalGetItems
- AerAxisCalGet
- AerAxisCalGetPacket

Example

Samples\LIB\AexCal\AexCal.c
4.10. AerAxisCalGetRawPosition

AERERR_CODE AerAxisCalGetRawPosition (HAERCTRL hAerCtrl, AXISINDEX iAxis, PLONG plPos);

Declare Function AerAxisCalRawPosition Lib "AERSYS.DLL" (ByVal hAerCtrl As Long, ByVal iAxis As Long, ByRef plPos As Long) As Long

Parameters
  hAerCtrl  Handle to the controller.
  iAxis     Axis to return state of axis calibration (see constants).
  plPos     Pointer to the uncorrected absolute position of the axis.

This function returns the true absolute position of the specified axis before axis calibration is applied.

C Language and LabView Constants
  AXISINDEX_1 to AXISINDEX_16

VB Constants
  aerAxisIndex1 to aerAxisIndex16

See Also
  AerAxisAllocateTable
  AerAxisCalGetStatus
  AerAxisCalGetStatusPacket
  AerAxisCalFreeTable
  AerAxisCalSetPoint
  AerAxisCalGetPoint
  AerAxisCalReset
  AerAxisCalSetMode
  AerAxisCalGetMode
  AerAxisCalSet
  AerAxisCalGetItems
  AerAxisCalGet
  AerAxisCalGetPacket

Example
  Samples\LIB\AexCal\AexCal.c
4.11. **AerAxisCalGetStatus**

AERERR_CODE AerAxisCalGetStatus (HAERCTRL hAerCtrl, WORD wTable,
    PWORD pwSize, PWORD pwStatus);

Declare Function AerAxisCalGetStatus Lib "AERSYS.DLL" (ByVal hAerCtrl As Long,
    ByVal wTable As Integer, ByRef pwSize As Integer, ByRef pwStatus As Integer) As Long

**Parameters**

- **hAerCtrl** Handle to the controller.
- **wTable** Error table number (first table is 0).
- **pwSize** Pointer to receive the number of entries in the table.
- **pwStatus** Pointer to receive the status of error table.

This function returns the status of an axis calibration table on the axis processor. The **pwSize** parameter indicates the number of entries in the specified axis calibration table. The **pwStatus** parameter is 1 if the table has been previously allocated otherwise it is 0. If **pwStatus** is returned as 0, then the number of entries and status are also returned as 0.

If the table has not been allocated, both **pwSize** and **pwStatus** will be zero.

**See Also**

- AerAxisAllocateTable
- AerAxisCalGetStatusPacket
- AerAxisCalFreeTable
- AerAxisCalSetPoint
- AerAxisCalGetPoint
- AerAxisCalReset
- AerAxisCalSetMode
- AerAxisCalGetMode
- AerAxisCalGetRawPosition
- AerAxisCalSet
- AerAxisCalGetItems
- AerAxisCalGet
- AerAxisCalGetPacket

**Example**

Samples\LIB\AexCal\AexCal.c
4.12. AerAxisCalGetStatusPacket

AERERR_CODE AerAxisCalGetStatusPacket( HAERCTRL hAerCtrl, WORD wTable,  
                                          PAER_AXISCAL_STATUS_PACKET pStatus );

Parameters

  hAerCtrl  Handle to the controller.
  wTable   Error table number (first table is 0).
  pStatus  Pointer to a structure of type AER_AXISCAL_STATUS_PACKET ,  
            indicating the status of the specified error table.

This function returns the status of the specified axis calibration table on the axis  
processor. The structure of type AER_AXISCAL_STATUS_PACKET contains two  
elements indicating the size of the axis correction table and the status of the table; 1 if  
applied, else 0.

See Also

  AerAxisAllocateTable
  AerAxisCalGetStatus
  AerAxisCalFreeTable
  AerAxisCalSetPoint
  AerAxisCalGetPoint
  AerAxisCalReset
  AerAxisCalSetMode
  AerAxisCalGetMode
  AerAxisCalGetRawPosition
  AerAxisCalSet
  AerAxisCalGetItems
  AerAxisCalGet
  AerAxisCalGetPacket

Example

  Samples\LIB\AexCal\AexCal.c
4.13. **AerAxisCalRemoveAll**

AERERR_CODE AerAxisCalRemoveAll( HAERCTRL hAerCtrl, AXISINDEX iCorrectedAxis);

Declare Function AerAxisCalRemoveAll Lib “AERSYS.DLL” ( ByVal hAerCtrl As Long, ByVal iCorrectedAxis As Long) As Long

**Parameters**

- **hAerCtrl** Handle to controller.
- **iCorrectedAxis** Index of axis to have all correction tables removed from (see constants).

This function removes all calibration tables that are currently affecting the given axis. The tables, if any, are removed from the controller’s memory.

**C Language and LabView Constants**

- **AXISINDEX_1** to **AXISINDEX_16**

**VB Constants**

- **aerAxisIndex1** to **aerAxisIndex16**

**See Also**

- AerAxisCalAllocateTable

AERERR_CODE AerAxisCalReset (HAERCTRL hAerCtrl, AXISINDEX iAxis);

Declare Function AerAxisCalReset Lib "AERSYS.DLL" (ByVal hAerCtrl As Long,
ByVal iAxis As Long) As Long

Parameters
   hAerCtrl   Handle to the controller.
   iAxis      Axis to deactivate axis calibration for (see constants).

This function disassociates all items or axis calibration tables for the specified axis. The
axis correction table must be disabled or an error returns. This function does not
deallocate the table from memory. The AerAxisCalFreeTable function is used for this
purpose.

C Language and LabView Constants
   AXISINDEX_1
   to
   AXISINDEX_16

VB Constants
   aerAxisIndex1
   to
   aerAxisIndex16

See Also
   AerAxisAllocateTable
   AerAxisCalGetStatus
   AerAxisCalGetStatusPacket
   AerAxisCalFreeTable
   AerAxisCalSetPoint
   AerAxisCalGetPoint
   AerAxisCalSetMode
   AerAxisCalGetMode
   AerAxisCalGetRawPosition
   AerAxisCalSet
   AerAxisCalGetItems
   AerAxisCalGet
   AerAxisCalGetPacket

Example
   Samples\LIB\AexCal\AexCal.c
4.15.  AerAxisCalSet

AERERR_CODE AerAxisCalSet (HAERCTRL hAerCtrl, AXISINDEX iAxis, WORD wTable, DWORD dwInput, DWORD dwScale);

Declare Function AerAxisCalSet Lib "AERSYS.DLL" (ByVal hAerCtrl As Long, ByVal iAxis As Long, ByVal wTable As Long, ByVal dwInput As Long, ByVal dwScale As Long) As Long

Parameters

hAerCtrl    Handle to the controller.
iAxis       Axis to correct (see constants).
wTable      Table number (first table is zero index).
dwInput     Mask of the axes causing correction.
dwScale     Scale for linear correction.

This function associates an axis correction table to a physical axis. The “dwScale” parameter is a scale factor in machine steps to multiply the axis error correction value by, before applying it to the axis. The maximum number of tables that can be attached to one axis is eight. A scale factor of zero disables scaling; (i.e. it is equivalent to providing a scaling factor of one). For example, axis correction table #3 contains correction values for axis #1 as a result of change in position of axis #4. The proper use of the function for this assignment would be:

C  AerAxisCalSet (hAerCtrl, AXISINDEX_1, 3, AXISMASK_4, 0);
VB  AerAxisCalSet (hAerCtrl, aerAxisIndex1, 3, aerAxisMask4, 0)

where hAerCtrl is the handle to the controller. AXISINDEX_1 represents the axis index for the axis to correct. The 3 is the table number, AXISMASK_4 represents axis #4, and 0 is the scale for the correction factor. Following this function, a call should be made to AerAxisCalSetMode to enable axis calibration for the specified axis.

C Language and LabView Constants

AXISINDEX_1  to  AXISMASK_1
AXISINDEX_16  to  AXISMASK_16

VB Constants

aerAxisIndex1  to  aerAxisMask1
aerAxisIndex16 to  aerAxisMask16

See Also
AerAxisAllocateTable  AerAxisCalGetPoint  AerAxisCalGetRawPosition
AerAxisCalGetStatus  AerAxisCalReset  AerAxisCalGetItems
AerAxisCalGetStatusPacket  AerAxisCalSetMode  AerAxisCalGet
AerAxisCalFreeTable  AerAxisCalGetMode  AerAxisCalGetPacket
AerAxisCalSetPoint

Example

Samples\LIB\AexCal\AexCal.c
4.16. AerAxisCalSetMode

AERERR_CODE AerAxisCalSetMode (HAERCTRL hAerCtrl, AXISINDEX iAxis, WORD wMode);

Declare Function AerAxisCalSetMode Lib "AERSYS.DLL" (ByVal hAerCtrl As Long, ByVal iAxis As Long, ByVal wMode As Long) As Long

Parameters
  hAerCtrl Handle to the controller.
  iAxis Axis number to deactivate axis calibration for (see constants).
  wMode Enable or disable axis calibration tables for the specified axis.

This function enables/disables all axis calibration tables for the specified axis. The wMode parameter is set to 1 to enable axis calibration or 0 to disable axis calibration. The specified axis must be disabled (drive axis parameter off) or an error occurs. If the axis is already in the specified mode, no action is taken and no error is delivered.

C Language and LabView Constants
AXISINDEX_1
to
AXISINDEX_16

VB Constants
aerAxisIndex1
to
aerAxisIndex16

See Also
AerAxisAllocateTable
AerAxisCalGetStatus
AerAxisCalGetStatusPacket
AerAxisCalFreeTable
AerAxisCalSetPoint
AerAxisCalGetPoint
AerAxisCalReset
AerAxisCalGetMode
AerAxisCalGetRawPosition
AerAxisCalSet
AerAxisCalGetItems
AerAxisCalGet
AerAxisCalGetPacket

Example
Samples\LIB\AexCal\AexCal.c
4.17.  **AerAxisCalSetPoint**

AERERR_CODE AerAxisCalSetPoint (HAERCTRL hAerCtrl, WORD wTable, WORD wPoint, LONG lPosition, LONG lCorrection);

Declare Function AerAxisCalSetPoint Lib "AERSYS.DLL" (ByVal hAerCtrl As Long, ByVal wTable As Integer, ByVal wPoint As Integer, ByVal lPosition As Long, ByVal lCorrection As Long) As Long

**Parameters**

- **hAerCtrl** Handle to the controller.
- **wTable** Pointer to the error table number (0 is first error table).
- **wPoint** Table entry number (0 is first point).
- **lPosition** Absolute position in machine steps to correct at.
- **lCorrection** Correction value at specified position, in machine steps.

This function sets an entry within the specified axis calibration table on the axis processor. The **lPosition** to correct at is an absolute position in machine steps referenced from the home position (See **AerMoveHomexxxx** functions) and the **lCorrection** value is an absolute value in machine steps to be added to the position to reach the desired absolute position. Axis calibration tables are not enabled until the axis has been homed.

Points must be monotonically ordered by **lPosition** value, with no two points having the same **lPosition** value. Points must also be ascending in **lPosition** value.

Users can assign points to tables that are associated already. The functions **AerAxisCalSet** and **AerAxisCalSetMode** are used to assign the table to a physical axis and activate the table(s) for that axis.

**See Also**

- **AerAxisAllocateTable**
- **AerAxisCalGetStatus**
- **AerAxisCalGetStatusPacket**
- **AerAxisCalFreeTable**
- **AerAxisCalGetPoint**
- **AerAxisCalReset**
- **AerAxisCalSetMode**
- **AerAxisCalGetMode**
- **AerAxisCalGetRawPosition**
- **AerAxisCalSet**
- **AerAxisCalGetItems**
- **AerAxisCalGet**
- **AerAxisCalGetPacket**

**Example**

Samples\LIB\AexCal\AexCal.c
4.18. 2D Calibration

Two dimensional (2D) axis calibration is similar to axis calibration, however it allows up to 3 axes to be corrected against the position of 2 axes. The 2D Axis calibration has the following syntax:

- Generic delimiter: no enforced structure and spaces, tabs, commas acceptable between data
- Comments are written between ‘;’ and next CRLF.
- All tables will assume 3D therefore correction for z will always be 0 if 2D
- Step size of data to be defined
- # elements are not pre-defined, but # columns are
- Error data is “absolute” not “relative”

:MULTI
; input axis
; 1
; output axis
; 1
; 2
; 3
; axis 1 sample distance; axis 2 sample distance
; axis 1 offset into table; axis 2 offset into table
; # columns (# points / row); # rows to be computed from total elements
; \[ \epsilon_{out1} \epsilon_{out2} \epsilon_{out3} \] i j . . .

Actual Table File Example

:MULTI ; Designator for multidimensional table information
1 2 ; input axes are #1 and #2
1 2 3 ; output axes are #1, #2, #3, entering ‘0’ if no third axis
1000 1000 ; 1000 machine steps between samples for #1, same for #2
0 0 ; no offset either axes (used as pointer in table to resolve marker location)
5 ; 5 points/row, therefore columns need not be specified.
1 1 0 2 2 0 3 3 0 2 2 0 3 3 0 ; data reflects 2D correction since element #3 is 0.
2 1 0 3 2 0 3 2 0 3 2 0 2 3 0

Note that the above format is for readability only. Actual data may look like:

1 2 1 2 3 1000 1000 0 0 5 1 1 0 2 2 0 3 3 0 2 2 0 3 3 0 2 1 0 3 2 0 3 2 0 3 2 0 3 2 0 3 2 0 3 2 0 2 3 0

A physical representation of the data follows in row and column format for readability and reconciliation with the stage orientation:

\[
\begin{bmatrix}
110_{1000,1000} & 220_{2000,1000} & 330_{3000,1000} & 220_{4000,1000} & 330_{5000,1000} \\
210_{1000,2000} & 320_{2000,2000} & 320_{3000,2000} & 320_{4000,2000} & 230_{5000,2000}
\end{bmatrix}
\]

See the U600MMI.hlp file for more information.
4.19. **AerAxisCal2DAllocateTable**

AERERR_CODE AerAxisCal2DAllocateTable (HAERCTRL hAerCtrl, WORD wRows, WORD wCols);

Declare Function AerAxisCal2DAllocateTable Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal wRows As Integer, ByVal wCols As Integer) As Long

**Parameters**

- **hAerCtrl** Handle to the controller.
- **wRows** Number of rows in calibration table.
- **wCols** Number of columns in each row of the calibration table.

This function allocates a 2D calibration table. The size of the table is specified by the $wRows \times wCols$.

**See Also**

*AerAxisCal2DfreeTable*
4.20. AerAxisCal2DFileDownload

AERERR_CODE AerAxisCal2DFileDownload (HAERCTRL hAerCtrl, LPCTSTR pszFile);

Declare Function AerAxisCal2DFileDownload Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal pszFile As String) As Long

Parameters

  hAerCtrl   Handle to the controller.
  pszFile    2D-Axis Calibration file.

This function downloads the specified axis calibration file to the controller.
4.21. AerAxisCal2DFreeTable

AERERR_CODE AerAxisCal2DFreeTable (HAERCTRL hAerCtrl);

Declare Function AerAxisCal2DFreeTable Lib “AERSYS.DLL” (ByVal hAerCtrl As Long) As Long

Parameters

hAerCtrl Handle to the controller.

This function frees the 2D calibration table.

See Also

AerAxisCal2DAllocateTable
4.22. AerAxisCal2DGetData

AERERR_CODE AerAxisCal2DGetData ( HAERCTRL hAerCtrl, PAXISINDEX piIn1, PAXISINDEX piIn2, PDWORD pdwIncr1, PDWORD pdwIncr2, PAXISINDEX piOut1, PAXISINDEX piOut2, PAXISINDEX piOut3, PDWORD pdwNumRows, PDWORD pdwNumCols );

Declare Function AerAxisCal2DGetData Lib “AERSYS.DLL” ( ByVal hAerCtrl As Long, ByRef piIn1 As Long, ByRef piIn2 As Long, ByRef pdwIncr1 As Long, ByRef pdwIncr2 As Long, ByRef piOut1 As Long, ByRef piOut2 As Long, ByRef piOut3 As Long, ByRef pdwNumRows As Long, ByRef pdwNumCols As Long ) As Long

Parameters

- *hAerCtrl* Handle to the controller.
- *piIn1* Input axis 1 (see constants).
- *piIn2* Input axis 2 (see constants).
- *pdwIncr1* Increment distance between samples for axis 1.
- *pdwIncr2* Increment distance between samples for axis 2.
- *piOut1* Output axis 1 (see constants).
- *piOut2* Output axis 2 (see constants).
- *piOut3* Output axis 3 (see constants).
- *pdwNumRows* Number of rows.
- *pdwNumCols* Number of cols.

This function returns the information associated with the 2D calibration table allocated on the controller.

C Language and LabView Constants

```
AXISINDEX_1
to
AXISINDEX_16
```

VB Constants

```
aerAxisIndex1
to
aerAxisIndex16
```

See Also

```
AerAxisCal2DSetData
```
4.23. AerAxisCal2DGetMode

AERERR_CODE AerAxisCal2DGetMode ( HAERCTRL hAerCtrl, PWORD pwMode);

Declare Function AerAxisCal2DGetMode Lib "AERSYS.DLL" ( ByVal hAerCtrl As Long, ByRef pwMode As Integer) As Long

Parameters

    hAerCtrl    Handle to the controller.
    pwMode      Current mode of 2D calibration table (0-disabled, 1-enabled).

This function returns the current state of the 2D Calibration Table.

See Also

    AerAxisCal2DSetMode
4.24. AerAxisCal2DGetPoint

AERERR_CODE AerAxisCal2DGetPoint (HAERCTRL hAerCtrl, WORD wRow,  
WORD wCol, PDWORD pdwX, PDWORD pdwY, PDWORD pdwZ);

Declare Function AerAxisCal2DGetPoint Lib “AERSYS.DLL” ( ByVal hAerCtrl As  
Long, ByVal wRow As Integer, ByVal wCol As Integer, ByRef pdwX  
As Long, ByRef pdwY As Long, ByRef pdwZ As Long ) As Long

Parameters

- **hAerCtrl**: Handle to the controller.
- **wRow**: Which row in calibration table to retrieve.
- **wCol**: Which column in calibration table to retrieve.
- **pdwX**: Correction value for axis 1.
- **pdwY**: Correction value for axis 2.
- **pdwZ**: Correction value for axis 3.

This function returns the correction values for a given row and column.

See Also

- *AerAxisCal2DSetPoint*
4.25. AerAxisCal2DSetData

AERERR_CODE AerAxisCal2DSetData ( HAERCTRL hAerCtrl, AXISINDEX iIn1, AXISINDEX iIn2, DWORD dwIncr1, DWORD dwIncr2, AXISINDEX iOut1, AXISINDEX iOut2, AXISINDEX iOut3);

Declare Function AerAxisCal2DSetData Lib “AERSYS.DLL” ( ByVal hAerCtrl As Long, ByVal iIn1 As Long, ByVal iIn2 As Long, ByVal dwIncr1 As Long, ByVal dwIncr2 As Long, ByVal iOut1 As Long, ByVal iOut2 As Long, ByVal iOut3 As Long ) As Long

Parameters

hAerCtrl Handle to the controller.
iIn1 Input axis 1 (see constants).
iIn2 Input axis 2 (see constants).
dwIncr1 Increment distance between samples for axis 1.
dwIncr2 Increment distance between samples for axis 2.
iOut1 Output axis 1 (see constants).
iOut2 Output axis 2 (see constants).
iOut3 Output axis 3 (see constants).

This function sets up all the necessary input parameters for the 2D calibration table.

C Language and LabView Constants

AXISINDEX_1 to AXISINDEX_16

VB Constants

aerAxisIndex1 to aerAxisIndex16

See Also

AerAxisCal2DGetData
4.26. **AerAxisCal2DSetMode**

AERERR_CODE AerAxisCal2DSetMode (HAERCTRL hAerCtrl, WORD wMode);

Declare Function AerAxisCal2DSetMode Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal wMode As Integer ) As Long

**Parameters**

- *hAerCtrl*  Handle to the controller.
- *wMode*  Set mode of 2D calibration table (0-disabled, 1-enabled).

This function sets the current state of the 2D Calibration Table.

**See Also**

- *AerAxisCal2DGetMode*
4.27. AerAxisCal2DSetPoint

AERERR_CODE AerAxisCal2DSetPoint ( HAERCTRL hAerCtrl, WORD wRow,
    WORD wCol, DWORD dwX, DWORD dwY, DWORD dwZ);

Declare Function AerAxisCal2DSetPoint Lib "AERSYS.DLL" ( ByVal hAerCtrl As
Long, ByVal wRow As Integer, ByVal wCol As Integer, ByVal dwX As
Long, ByVal dwY As Long, ByVal dwZ As Long ) As Long

Parameters

  * **hAerCtrl**: Handle to the controller.
  * **wRow**: Which row in calibration table to retrieve.
  * **wCol**: Which column in calibration table to retrieve.
  * **dwX**: Correction value for axis 1.
  * **dwY**: Correction value for axis 2.
  * **dwZ**: Correction value for axis 3.

This function sets the correction values for a given row and column.

See Also

  * *AerAxisCal2DGetPoint*
4.28. Related Axis Parameters

**DRIVE** - This parameter enables and disables the motor’s torque associated with an axis. A zero disables the drive, while a 1 enables it.

**REVERSALMODE** - This parameter allows the user to specify the number of machine steps required to compensate for any backlash present in the system. Backlash occurs when the ball screw changes direction and moves a fixed distance before the stage begins to move in the new direction. This parameter specifies the distance in machine steps. Zero counts disable backlash compensation.

**REVERSALVALUE** - This parameter is the current correction value (in machine counts) for the reversal mode (backlash compensation).

See the U600MMI.hlp file for more information on these parameters.
CHAPTER 5: ELECTRONIC CAMMING FUNCTIONS

5.1. Introduction

Electronic camming (master/slave) functions allow the user to command master/slave motion. Master/slave motion is the most general form of motion allowing the user to command a single axis to move with virtually any position or velocity profile. Furthermore, by synchronizing multiple slave axes to a common master, the user can move multiple axes in fully synchronized motion. Master/slave motion, originally developed for grinding cam shafts, has many other uses like forcing an axis to follow a handwheel. For historical reasons, master/slave motion is alternately referred to as “camming.” However, there is a drawback to the master/slave motion, the programmer must understand and do more to achieve the proper results.

Two axes are involved in any master/slave motion, the axis that executes the desired motion is called the "slave" axis and a user provided "master" axis is used to direct the slave. The slave is commanded by the master axis' commanded position (refer to AerConfigMaster).

For each master/slave relationship, the programmer must provide a table of coordinates (a cam table) that specifies slave axis positions for each master axis position. Refer to Figure 5-1.

See the U600MMI.hlp file for more information on camming.
When the current position of the master axis is a value specified in one of the above points, then the slave axis position will take on the corresponding slave value of the point. If the current master position lies in between points specified in the table, then the slave position is found by either linear or spline interpolation of the adjacent points. (see the SYNC command). However, if the current position of the master is outside the range of the master values provided by the user, then the current master position will be “wrapped” so as to fall within the specified values in the table. For example, if the master points in the table range from 0 to 10, and the master axis is currently at position 12, then the master point for value 2 will be used. Therefore, if the master axis is not rotary, the user should provide points for the entire range of potential master motion.

The slave axis can act as a master to another axis, allowing the user to direct multiple axes in master/slave motion. In addition, the programmer can specify slave axis velocities for given master axis positions (see AerCamTableSetMode).

One can also add an "infeed" onto the slave while it is camming. The infeed is specified as a position and speed, (see AerMoveInfeedSlave function) and the required motion produced by the infeed will be added onto the cam directed motion.
5.1.1. **Required Camming Steps**

The five steps needed to direct master/slave motion are listed below with the function(s) detailing that step.

1. Allocate cam table  
   Refer to `AerCamTableAllocate` function.
2. Load cam table  
   Refer to `AerCamTable` functions.
3. Configure master/slave axis  
   Refer to `AerConfigMaster` function.
4. Synchronize an axis  
   Refer to `AerCamTableSetMode` function.
5. Initiate master motion  
   Refer to Parameters section.

---

The axis needs to be enabled before step 4 in the above procedure.

For help on important related issues see the following sections: Note that some of this information is contained in other chapters.

- Interpolation  
  Refer to `AerCamTableCalcCoeff`.
- Table entry/exit  
  Refer to `AerCamTableSetMode` and all related axis parameters.
- Infeeding  
  Refer to `AerMoveInfeedSlave`.
- Master position  
  Refer to `AerConfigMaster` and all related axis parameters.

---

Care should be taken when engaging cam table motion as its incorrect application may result in abrupt changes in slave velocity. Make sure that the master axis is stationary when engaging or disengaging cam table motion for best results.

---

Example

Samples\LIB\AexCam\AexCam.c
5.2. AerCamTableAllocate

AERERR_CODE AerCamTableAllocate (HAERCTRL hAerCtrl, WORD wTable, DWORD dwSize);

Declare Function AerCamTableAllocate Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal wTable As Integer, ByVal dwSize As Long) As Long

Parameters
  hAerCtrl Handle to the axis processor card.
  wTable Cam table number.
  dwSize Number of points in the table.

This function allocates memory on the axis processor card in order to store a cam table. Table numbers may range from 0 through 99 and can not be dynamically reallocated - they must be freed before reallocating to a different size. Each cam table point occupies 24 bytes of memory on the axis processor and all points allocated will be filled with zeros after allocation.

See Also
  AerCamTableFree

Example
  Samples\LIB\AexCam\AexCam.c
5.3. AerCamTableCalcCoeff

AERERR_CODE AerCamTableCalcCoeff (HAERCTRL hAerCtrl, WORD wTable, DWORD dwWait);

Declare Function AerCamTableCalcCoeff Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal wTable As Integer, ByVal dwWait As Long) As Long

Parameters

- **hAerCtrl** Handle to the axis processor card.
- **wTable** Cam Table number.
- **dwWait** Maximum amount of time in milliseconds to wait until the coefficients are calculated.

This function computes the coefficients necessary to interpolate between the points specified in the cam table. Coefficients must be calculated before performing master/slave motion because each point has its own coefficients. The user can specify two types of interpolation: linear or spline for each point. The type of interpolation is specified when adding the point to the table (refer to AerCamTableSetPoint).

Coefficients cannot be calculated for a table that has not been allocated or has an axis currently synchronized to it, (refer to AerCamTableSetMode). The interpolation can be linear or cubic spline (set by AerCamSetPoint) where cubic spline is used for maximum "smoothness," and linear for minimum "rippling." The splines are 3rd order, but are not true Bsplines. In true Bsplines, an entire n by n matrix (where n is the number of points) must be solved to obtain values for all the points. Instead, for speed and memory optimization, a 2 by 2 matrix for each pair of points is solved, carrying the resultant derivatives over to the adjacent pair as the boundary conditions for the next splining point. The results are nearly identical to true Bsplines and in some cases are preferable.

Computing coefficients can take up a significant amount of axis processor time. The dwWait parameter specifies whether the caller wants to wait for computation completion before returning. If dwWait is set to zero, the function returns immediately and the programmer must later call AerCamTableGetStatus to insure that the computation is done. If dwWait is non-zero, the function will wait a maximum of dwWait milliseconds for the coefficient calculations to complete. If the calculation has not completed within this time, the function will return AERERR_CAMTABLE_TIMEOUT. In C language, this define can be used. However, in VB or C, the AerErrGetMessage function may be called to translate an error number to a text error message.

The cam table coefficients for each point are zeroed when the point is loaded into the table.

See Also

- AerCamTableSetMode
- AerCamTableGetStatus
- AerCamTableSetPoint

Example

Samples\LIB\AexCam\AexCam.c
5.4. AerCamTableFileDownload

AERERR_CODE AerCamTableFileDownload (HAERCTRL hAerCtrl, DWORD dwTable, AXISINDEX iMasterAxis, AXISINDEX iSlaveAxis, WORD wInterpolationType, LPTSTR pszFileName);

AERERR_CODE AerCamTableFileDownload Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal dwTable As Long, ByVal iMasterAxis As Long, ByVal iSlaveAxis As Long, ByVal wInterpolationType As Integer, ByVal pszFileName As String) As Long

Parameters

- **hAerCtrl**: Handle to the Axis Processor card.
- **iMasterAxis**: Axis index of master axis (see constants).
- **iSlaveAxis**: Axis index of slave axis (see constants).
- **dwTable**: Table number.
- **wInterpolationType**: Interpolation type (see constants).
- **pszFileName**: Pointer to file containing cam table data.

This function reads an ASCII file containing cam table data, forms a cam table, and configures the master of the given slave axis to the given master axis. The master axis is the axis from which the lookup position that is used in the table is obtained. Once the position for the slave axis has been determined by table lookup, this value is the “command” position of the slave axis. However, camming is only activated or in place immediately after the AerCamTableSetMode call (use AerCamTableSetMode to turn camming off). Therefore, the programmer should first use AerCamTableFileDownload as a setup action before running any programs and only use AerCamTableSetMode when the desire is to actually do cam motion. Check whether camming is “synced” for a particular axis, by checking the sync mode bit of the axis status for that axis (use AerStat for this).

The format of the ASCII file is described in the following paragraphs.

The file must consist of data lines and comment lines. A line is defined as a string of characters terminated by any number of consecutive line terminator characters. A line terminator character is a carriage return or a line feed.

Data lines are those whose first "non-whitespace" character is a "number" character. All other lines are comment lines. Whitespace characters are the tab, formfeed, comma, or space character. A number character is a digit, decimal point, or minus sign.

After the first data line is read-in, the program will continue reading-in data lines until an End of File or comment line is seen. Then the file is closed.
COMMENT LINES
The file must contain comment lines, preceding the first data line, providing three pieces of information:

- Number of points in the file
- Units of the master position values
- Units of the slave position values

The number of target points must be preceded by the keyword “Number of Points.” The first number character after this keyword will begin the number read-in as the number of target positions. The program must read-in exactly the number of data lines given by the “Number of Target Positions.”

The units of the position values must be preceded by the keyword “Master Units.” The first non-whitespace character that is not “(” after the keyword, begins the master position units descriptor. Units descriptors can be “nanometers,” “mm,” “in,” “microns,” “millimeters,” or “inches.” Identification of the master position units descriptor is case insensitive.

The units and scale factor of the slave position values must be preceded by the keyword: “Slave Units(.” The first non-whitespace character that is not “(” after the keyword, begins the slave scale factor (the slave scale factor is optional and if not specified assumed to be one). The first non-whitespace character after the slave scale factor must be the slave position units descriptor. Slave position units descriptors can be “nanometers,” “mm,” “in,” “microns,” “millimeters,” or “inches.” Identification of the slave position units descriptor is case insensitive.

DATA LINES
Each data line must have three numbers, the first of which must be an integer. All text following the first three numbers is ignored. The numbers must be separated by one or more whitespace characters.

The first number is the point number. It must be a positive integer. These numbers are not used.

The second number is the position value. This value is assumed to be in the units specified in the comment lines at the beginning of the file. Position values must be monotonically increasing or decreasing throughout the file but need not be evenly spaced.

The third number is the compensation value that goes with the position value. It is assumed to be in the units specified in the comment lines at the beginning of the file (i.e. if the value read-in is -2, and the compensation read-in was .25 microns, then the actual compensation value is -.5 microns). Compensation values are always measured relative to the zero compensation value, that is, the position command that would exist if no compensation were being performed.

All table entries consist of a master position and a slave value, where no two table entries have the same master position (see AerCamTableSetPoint for more details on table entries). If the current master position is equal to a table entry master position, then the slave value for that table entry is used as the current slave value. If, however, the current master position lies in-between two master positions listed in the table, then either linear or spline interpolation (see wInterpolationType parameter) is used to find a slave value between the two slave values listed in the table. However, if the current position of the master is outside the range of the master values provided by the user, then the current
master position will be “wrapped” so as to fall within the specified values in the table. For example, if the master points in the table range from 0 to 10, and the master axis is currently at position 12, then the master point for value 2 will be used. Therefore, if the master axis is not rotary, the user should provide points for the entire range of potential master motion.

Master position values listed in the file must be in ascending order of position (after the position is translated into counts, they can have a descending order of position in the file if the CntsPerInch/CntsPerDeg machine parameter is negative).

C Language and LabView Constants

AXISINDEX_1 to AXISINDEX_16

AERCAM_XXXX

VB Constants

aerAxisIndex1 to aerAxisIndex16

aerCamXXXX

See Also

AerCamTableAllocate
AerCamTableGetStatus
AerCamTableFree
AerCamTableSetPoint
AerCamTableGetPoint
AerCamTableSetMode
AerCamTableGetMultPoints
AerCamTableGetMode

Example

Samples\LIB\AexCam\AexCam.c
5.5. AerCamTableFree

AERERR_CODE AerCamTableFree (HAERCTRL hAerCtrl, WORD wTable);

Declare Function AerCamTableFree Lib "AERSYS.DLL" (ByVal hAerCtrl As Long, ByVal wTable As Integer) As Long

Parameters

hAerCtrl Handle to the axis processor card.
wTable Cam Table number.

This function de-allocates memory on the axis processor card that was used to store a cam table.

See Also

AerCamTableSetMode
AerCamTableAllocate

Example

Samples\LIB\AexCam\AexCam.c
5.6. **AerCamTableGetMode**

AERERR_CODE AerCamTableGetMode (HAERCTRL hAerCtrl, AXISINDEX iAxis, PWORD pwTable, PWORD pwMode);

Declare Function AerCamTableGetMode Lib "AERSYS.DLL" (ByVal hAerCtrl As Long, ByRef iAxis As Long, ByRef pwTable As Integer, ByRef pwMode As Integer) As Long

**Parameters**

- **hAerCtrl** Handle to the axis processor card.
- **iAxis** A physical axis index, representing the slave axis (see constants).
- **wTable** Cam table number.
- **wMode** Pointer to WORD to receive desired mode (see constants).

This function returns the current synchronization mode of the slave axis. If the specified axis is not synchronized to a table, then zero is returned for both the table and the mode.

Refer to **AerCamTableSetMode** for a more detailed description of the parameters to this function.

**C Language and LabView Constants**

- **AXISINDEX_1**
- **AXISINDEX_16**
- **AERCAM_XXXX**

**VB Constants**

- **aerAxisIndex1**
- **aerAxisIndex16**
- **aerCamXXXX**

**See Also**

- **AerCamTableSetMode**

**Example**

Samples\LIB\AexCam\AexCam.c
5.7. AerCamTableGetMultPoints

AERERR_CODE AerCamTableGetMultPoints (HAERCTRL hAerCtrl, WORD wTable,
                DWORD dwStart, DWORD dwCount, PAER_CAM_GETPOINT pPoint);

Parameters

    hAerCtrl  Handle to the axis processor card.
    wTable   Cam Table number.
    dwStart  Starting point number.
    dwCount  Number of points to set.
    pPoint   Pointer to array of structures to receive point data.

This function allows the user to read multiple points simultaneously and is faster than
single reads, since it requires less calling overhead. The pPoint must point to an array of
AER_CAM_GETPOINT structures, each filled with data for one point. The points will
be retrieved consecutively starting with the point indicated by dwStart. The caller must
insure that dwCount number of structures exist, pointed to by pPoint. The
AER_CAM_GETPOINT structure is identical to the arguments described in the
AerCamGetPoint function description. Refer to Appendix C: Structures.

This function is identical to AerCamTableGetPoint.

See Also

    AerCamTableGetPoint
    AerCamTableSetMultPoints

Example

        Samples\LIB\AexCam\AexCam.c
5.8. AerCamTableGetPoint

AERERR_CODE AerCamTableGetPoint (HAERCCTRL hAerCtrl, WORD wTable, DWORD dwPoint, PLONG plMaster, PLONG plSlave, PLONG plCoeffA, LONG plCoeffB, PLONG plCoeffC, PWORD pwStatus);

Declare Function AerCamTableGetPoint Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal wTable As Integer, ByVal dwPoint As Long, ByRef plMaster As Long, ByRef plSlave As Long, ByRef plCoeffA As Long, ByRef plCoeffB As Long, ByRef plCoeffC As Long, ByRef pwStatus As Integer) As Long

Parameters
  hAerCtrl Handle to the axis processor card.
  wTable Cam Table number.
  dwPoint Point number.
  plMaster Pointer to Master position.
  plSlave Pointer to Slave position.
  plCoeffA Pointer to "A" coefficient.
  plCoeffB Pointer to "B" coefficient.
  plCoeffC Pointer to "C" coefficient.
  pwStatus Pointer to status of point. Is equal to 256 if that point has been loaded, else it is 0.

This function returns pointers to all relevant information concerning a point in a cam table. Points can be read from a table that has an axis currently synchronized to it (see AerCamTableSetMode). Points can be specified from -1 to the number-of-cam-table-points+1, where the number-of-cam-table-points is the number provided in the AerCamTableAllocate function.

For the meaning of any of the values returned in the Pointers (all arguments after dwPoint) see AerCamTableSetPoint.

Interpretation of the coefficients is as follows: y = Ax^3 + Bx^2 + Cx; where x is the current master position and y the interpolated slave position. For linear interpolation, the A and B coefficients are set to zero.

See Also
  AerCamTableSetPoint
  AerCamTableSetMode

Example
  Samples\LIB\AexCam\AexCam.c
5.9. **AerCamTableGetStatus**

AERERR_CODE AerCamTableGetStatus (HAERCTRL hAerCtrl, WORD wTable, PDWORD pdwSize, PWORD pwStatus);

Declare Function AerCamTableGetStatus Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal wTable As Integer, ByRef pdwSize As Long, ByRef pwStatus As Integer) As Long

**Parameters**

- **hAerCtrl**: Handle to the axis processor card.
- **wTable**: Cam Table number.
- **pdwSize**: Pointer to DWORD to receive cam table size.
- **pwStatus**: Pointer to word to receive the mask of the status (see constants).

This function returns the size of a cam table and its current status. The size is the number provided in the AerCamTableAllocate function. If the given table was never allocated, or is out of range (more than 99), then a zero is returned for both the size and the status. The pwStatus value returned is a mask of AERCAM_STAT_xxxx values. The possible status values that can be returned and their meanings are defined below, with the constants.

This function will not inform the user if a table has an axis synchronized to it. Use AerCamTableGetMode for this. The user cannot synchronize an axis to a table (refer to AerCamTableSetMode) unless the table is in relative mode.

**C Language and LabView Constants**

- **AERCAM_STAT_ALLOCATED**: Table Allocated.
- **AERCAM_STAT_COEFFS_DONE**: Coefficients have been calculated.
- **AERCAM_STAT_COEFFS_BUSY**: Coefficients being calculated.

**VB Constants**

- **aerCamStatAllocated**: cam table allocated  
  - aerCamPointLinear
- **aerCamStatCoeffsDone**: cam coefficients done and ready  
  - aerCamPointCubic
- **aerCamStatCoeffsBusy**: coefficients being calculated  
  - aerTaskMask1
- **aerCmplrDefault**: Uses object files, only recompile  
  - aerTaskMask2
- **aerCmplrPreprocOnly**: Only run preprocessor (use with MAKE_LISTING)  
  - aerTaskMask3
- **aerCmplrNoReadObj**: Always recompile it from source.  
  - aerTaskMask4
- **aerCmplrNoUseSrc**: Read object (no source code available)  
  - aerCamModeOff
- **aerCmplrMakeListing**: Puts preprocessor output in file  
  - aerCamModeRelative
- **aerCmplrSrcWithObj**: Reads source too, when reading from object  
  - aerCamModeAbsolute
- **aerCmplrNoWriteObj**: Never writes objects  
  - aerCamModeVelocity

**See Also**

- AerCamTableCalcCoeff
- AerCamTableSetMode

**Example**

- Samples\LIB\AexCam\AexCam.c
5.10. AerCamTableGetTrack

AERERR_CODE AerCamTableGetTrack (HAERCTRL hAerCtrl, AXISINDEX iAxis,
PDWORD pdwMStart, PDWORD pdwMAccel, PDWORD pdwSAccel);

Declare Function AerCamTableGetTrack Lib “AERSYS.DLL” (ByVal hAerCtrl As
Long, ByVal iAxis As Long, ByRef pdwMStart As Long, ByRef pdwMAccel As Long, ByRef pdwSAccel As Long) As Long

Parameters

  hAerCtrl          Handle to axis processor card.
  iAxis             The axis that is to begin tracking the master’s position (see
                    constants).
  pdwMStart         Starting master position.
  pdwMAccel         Distance in master axis counts.
  pdwSAccel         Distance in slave axis counts.

This function returns the current values of the parameters used for the Track command.

C Language and LabView Constants

  AXISINDEX_1 to
  AXISINDEX_16

VB Constants

  aerAxisIndex1 to
  aerAxisIndex16

See Also

  AerCamTableSetTrack
5.11. AerCamTableSetMode

AERERR_CODE AerCamTableSetMode (HAERCTRL hAerCtrl, AXISINDEX iAxis, WORD wTable, WORD wMode);

Declare Function AerCamTableSetMode Lib "AERSYS.DLL"(ByVal hAerCtrl As Long, ByVal iAxis As Long, ByVal wTable As Integer, ByVal wMode As Integer) As Long

Parameters

- **hAerCtrl**: Handle to the axis processor card.
- **iAxis**: A physical axis index, representing the slave axis (see constants).
- **wTable**: Cam Table number.
- **wMode**: Desired mode (see constants).

This function activates or deactivates the synchronization of a master to a slave. Prior to synchronizing, the cam table is inactive (master motion does not cause slave motion). Immediately after synchronizing, master motion will cause slave motion. In addition, this function defines the mode in which the slave values are interpreted. The **iAxis** parameter is an axis index that specifies a physical axis. If the synchronization modes are changed while the master or slave is moving, abrupt changes in motion may occur. The user is cautioned that synchronizing an axis and the camming behavior can be complex and the following description along with the description of the related axis parameters must be fully understood to produce the desired results.

After synchronizing a slave to a master, the slave motion is linked to master motion through the specified cam table. After a slave axis synchronization mode is deactivated, the slave motion no longer follows the master axis. The user must provide a synchronization mode. AERCAM_MODE_OFF deactivates the synchronization mode of a slave axis from a cam table. AERCAM_MODE_RELATIVE engages relative cam table mode. The slave axis is assumed to be in the correct position relative to the current master axis position. AERCAM_MODE_ABSOLUTE engages absolute cam table mode. The current slave position is checked against the slave position specified by the current master position. If they are different, then an axis move is generated to place the slave position in alignment with the corresponding master position as defined in the cam table. Camming begins immediately and the move is made at the speed specified by the **SYNCSPEED** parameter. The motion generated by this move will be added to the generated cam table motion.

AERCAM_MODE_VELOCITY is velocity table mode that specifies a slave axis velocity in counts/sec vs. a master axis position. AERCAM_MODE_VELOCITY also provides "smoothing" when synchronizing to a table (refer to the **MAXCAMACCEL** parameter at the end of this chapter, or in the U600MMI.hlp file).

The axis needs to be enabled before engaging sync mode.
C Language and LabView Constants
AXISINDEX_1
to
AXISINDEX_16
AERCAM_MODE_XXXX

VB Constants
aerAxisIndex1
to
aerAxisIndex16
aerCamModeXXXX

See Also
AerCamTableGetMode

Example
Samples\LIB\AexCam\AexCam.c
5.12. AerCamTableSetMultPoints

AERERR_CODE AerCamTableSetMultPoints (HAERCTRL hAerCtrl, WORD wTable,
        DWORD dwStart, DWORD dwCount, PAER_CAM_SETPOINT
        pPoint);

Parameters

        hAerCtrl  Handle to the axis processor card.
        wTable   Cam Table number.
        dwStart  Starting point number.
        dwCount  Number of points to set.
        pPoint   Pointer to array of structures containing point data.

This function allows the user to set a number of points simultaneously, assigning to points
consecutively increasing, starting with the point indicated by dwStart. This is faster than
single sets, since it requires less calling overhead. The pPoint must point to an array of
AER_CAM_SETPOINT structures, each containing data specifying one point. The caller
must insure that dwCount number of structures exist, pointed to by pPoint.

The AER_CAM_SETPOINT structure is identical to the arguments described in the
AerCamSetPoint function, refer to Appendix C: Structures for its description. This
function is similar to AerCamSetPoint.

See Also

        AerCamTableSetPoint

Example

        Samples\LIB\AexCam\AexCam.c
5.13. **AerCamTableSetPoint**

```c
AERERR_CODE AerCamTableSetPoint (HAERCTRL hAerCtrl, WORD wTable,
                                 DWORD dwPoint, LONG lMaster, LONG lSlave, WORD wType);
```

Declare Function AerCamTableSetPoint Lib “AERSYS.DLL” (ByVal hAerCtrl As Long,
ByVal wTable As Integer, ByVal dwPoint As Long, ByVal lMaster As Long,
ByVal lSlave As Long, ByVal wType As Integer) As Long

**Parameters**
- **hAerCtrl** Handle to the axis processor card.
- **wTable** Cam Table number.
- **dwPoint** Point number.
- **lMaster** Master position.
- **lSlave** Slave position.
- **wType** Interpolation type (see constants).

This function loads a cam table point into a cam table. Points cannot be specified for a
table which has not been allocated or currently has an axis synchronized to it (refer to
**AerCamTableSetMode**). Points can be specified in the range between -1 to the number of
cam table points+1, where the number of cam table points is the number provided in the
**AerCamTableAllocate** function. The **lSlave** and **lMaster** values define the master and
slave position relationship. The interpolation type must be one of the constants below.

The -1 and (cam_table_points +1) points are only used in spline mode to determine the
slope of the spline curve at the end points of the curve (see curve in Figure 4-1). If these
points are not provided, then the slope is assumed 0 through the end points (this is the
case in Figure 4-1).

The master coordinate values must be in a monotonically increasing sequence. Use the
**CAMOFFSET** axis parameter to change the alignment between the master position and
the slave position specified in the table.

**C Language and LabView Constants**
```
AERCAM_POINT_XXXX
```

**VB Constants**
```
aerCamPointXXXX
```

**See Also**
- AerCamTableAllocate
- AerCamTableCalcCoeff
- AerCamTableGetPoint
- AerCamTableSetMode

**Example**
```
Samples\LIB\AexCam\AexCam.c
```
5.14. AerCamTableSetTrack

AERERR_CODE AerCamTableSetTrack (HAERCTRL hAerCtrl, AXISINDEX iAxis,
DWORD dwMStart, DWORD dwMAccel, DWORD dwSAccel);

Declare Function AerCamTableSetTrack Lib “AERSYS.DLL” (ByVal hAerCtrl As
Long, ByVal iAxis As Long, ByVal dwMStart As Long, ByVal dwMAccel As Long, ByVal
dwSAccel As Long ) As Long

Parameters

hAerCtrl Handle to axis processor card.
iAxis The axis that is to begin tracking the master’s position (see constants).
dwMStart Starting master position.
dwMAccel Distance in master axis counts (see below)
dwSAccel Distance in slave axis counts (see below)

The iAxis parameter specifies the axis that is to begin tracking its master’s position. The
axis must not be moving when this command is executed (the master axis can be moving).

The dwMStart parameter specifies the starting master position, in master axis encoder
counts, to begin tracking. If the current master position exceeds dwMStart when
dwMAccel is positive, or is less than dwMStart when dwMAccel negative, then tracking
will begin immediately.

The dwMAccel parameter specifies the distance, in master axis encoder counts, over
which the master will travel as the slave accelerates from zero speed to the desired
tracking ratio. This distance can be positive or negative. The sign of the number
determines the direction of travel of the master axis.

The dwSAccel parameter specifies the distance, in slave axis encoder counts, over which
the slave will travel as it accelerates from zero speed to the desired tracking ratio.

This command is used to have a stationary slave axis begin tracking a moving master axis.
Typical applications include gantry type axes (slave axis) that must begin tracking the
motion of a moving conveyor belt (master axis). The command specifies the starting
master position to begin tracking along with the master and slave distance over which the
slave axis will accelerate. The ratio of the acceleration distances defines the ratio of the
final speed of the slave axis. The axis final velocity can be determined by the equation:

\[ \text{SLAVE}_\text{VEL} = \frac{[(\text{MASTR}_\text{VEL}) \times \text{dwSAccel}^2]}{\text{dwMAccel}} \]

Therefore, if the slave axis is to match the master axis velocity then dwSAccel should be
one-half dwMAccel (assuming the master and slave axes have the same resolution). This
relationship arises from the fact that while the master axis is moving at constant speed and
the slave axis accelerates from zero velocity to match the master velocity, the master axis
will have traveled twice as far as the slave. Note that all distances are specified in
encoder counts and not in inches or millimeters.

C Language and LabView Constants

AXISINDEX_1 to aerAxisIndex1
AXISINDEX_16 to aerAxisIndex16

See Also

AerCamTableGetTrack
5.15. Related Axis Parameters

*CAMADVANCE* - This parameter is used as a master offset advance that is a function of the velocity of the master axis. The units are in counts/(counts/millisecond).

*MAXCAMACCEL* - Mode 3 offers acceleration/deceleration protection that can be used when synchronizing on the fly (without desynchronizing in-between with the SYNC mode). If this parameter is a non-zero, the slave axis will not exceed the specified acceleration while camming. This acceleration is in units of user-units per millisecond squared. This parameter is not used in synchronization modes 1 and 2. To deactivate this feature, set the parameter value to 0.

*CAMOFFSET* - This parameter is added to the master position before doing the table lookup. For example, if the table covers master positions from 0° to 360°, the actual master position is 2° and *CAMOFFSET* is 3°, then the CNC will use the value of 5° as the master position to look up in the table.

*MATERLENGTH* - If upon synchronizing the actual master position lies outside of the master coordinates in the table, the table master coordinates will be pushed up or down in units of *MASTERLENGTH* until it lies within the table. For example, if the table covers from 0° to 359° (the master is rotational) and the current master coordinate is 361°, the axis processor uses the 1° entry in the table to direct the slave.

The *MASTERLENGTH* parameter should not be used if the master axis position will exceed the *MASTERLENGTH* parameter. Doing so causes the slave axis to abruptly reverse position to maintain alignment with the master position.

*MASTERPOS* - This parameter is the master position, as seen by the slave. This is the actual value used to look up in the table. The *MASTERPOS* parameter can only be set while the axis is not in sync mode. Once the axis is in sync mode, the *MASTERPOS* is maintained relative to the master axis position by integrating the master axis velocity and adding it to the initial *MASTERPOS* parameter. Note that the user can direct either the master axis' actual position, or the master axis' commanded position be used to derive the slave’s *MASTERPOS* parameter. (see the *AerConfigMaster* function). If the *MASTERLEN* is non-zero, then every time the *MASTERPOS* exceeds the *MASTERLEN* value, it is reset to 0. Any changes in the master axis's position will still be reflected in the slave axis's *MASTERPOS* value, but the *MASTERPOS* value will now be *MASTERLEN* lower than the master axis's position. This jump in *MASTERPOS* causes a jump in the cam table output for the slave, which may cause abrupt motion in the slave. To understand how this parameter works, the reader must be familiar with the operation of synchronized motion through the use of CAM tables on the U600 Series controller. While operating in this mode, axis motion relates directly to motion on the master axis (the axis designated by the user). The basis of this relationship is dependent on the currently active CAM table. Each CAM table entry contains two position types: a master position and a slave position. As the master axis approaches the positions found within the CAM table, the slave axis moves to the corresponding slave position. Interpolation occurs between the CAM points.
The first CAM table entry for the master position must be a zero. Two rules apply to all master positions following the first entry: they must always increase and they must never repeat.

See the U600MMI.hlp file for more information on these parameters.
6.1. Introduction

The Aerotech CNC Compiler allows the programmer to compile and download CNC lines and programs containing G codes, etc. These codes do not cause the controller to execute the programs or CNC line. The AerTaskXXXX functions are used for that purpose (see Chapter 22). These functions also allow the programmer to obtain data on CNC program errors, monitor the status of CNC compiling/downloading, and obtain CNC program information (i.e., compile warnings, time of compile, number of lines compiled). Please see the UNIDEX 600 Series CNC Programming Manual, P/N EDU158 for details on valid CNC program text.

The following remarks apply to all AerCompiler function calls. See Samples\LIB\AexProg\AexProg.c for a C example on compiling and running a program.

- All line, character, and error indices passed in and returned are zero-based. This means the first line is indicated by the line number “0”, and so on. However, counts returned are one-based. For example, if a CNC program has two errors, a call to the function “AerCompilerGetNumOfErrs” will yield “2”. To get information on the two errors, the programmer must call the function “AerCompilerGetErrData,” passing it the indices “0” and “1” respectively.
- Line and character numbers returned as the constant -1 indicate a NULL, or invalid number.
- Functions with "Get" in their name write data into a location indicated by a pointer that is passed in the argument list. For example, the function “AerCompilerGetErrText” returns error text information in a “char*” pointer or string variable passed in the argument list. If the passed pointer is detected as being bad (NULL or points out of the user sector) the routine returns AERCMPLR_GENERAL_BAD_PASSED_BUFFER, which can be tested for in C language. In VB and C, the error return code can be passed to AerErrGetMessage function, which will return a text message for the error code. In this case, do not use
the passed pointer, or the program may crash. The user should never get this error, unless an error was made using or writing the program.

- Error data consists of two parts, the error condition data, and error location data. Error condition data describes the error. Error location data consists of the line number, char number, file, etc., where the error occurs. See the description of the Compile_Error_Data structure for more details (in Appendix C: Structures). Both location and condition error data can be returned in formatted strings, or as values.

Either a file or a string can be compiled as a program and once the compilation is complete, the program can be downloaded as a regular program, or an infinite-size queued program. After downloading, you must associate before running the program (see Samples\LIB\AexProg\AexProg.c, or the beginning of the AerTask chapter). The compile object holds data concerning the program text, the program object (its compiled form), and data concerning compile errors or warnings in memory. After a compile, the user can make calls to the compile object to obtain text lines, or error data. Object files can be written or read to the hard drive, based on the mode bits passed. In the default mode, the compiler automatically manages the object files (.ogm), only recompiling when the source code changes.

A pseudo-code example of the steps required to compile a CNC program is shown below. Please see the Samples\LIB\AexProg\AexProg.c file for more detailed examples, including examples on how to compile a single line, and how to download program queues.

AerCompilerCreate() ; 1. Initialize the compiler
AerCompilerLoadAxisNames() ; Load user axis names
AerCompilerAutoIncludeEx() ; Add include files (AerParam.pgm and
                                ; Mcode.pgm)
AerCompilerCompileFile() ; 2. Compile the program
If ( ERRORS ) ; any errors?
{
    AerCompilerErrsGetNumOf() ; 3. Get number of compiler errors
    for ( ALL_ERRORS ) ; show all compile errors (optional)
    {
        AerCompilerErrGetCode() ; Get error code
        AerCompilerErrGetText() ; Get error text
        AerCompilerErrGetData() ; Get error data
        AerCompilerErrGetLocText() ; Get bad CNC text string
        AerCompilerGetLineText() ; Get bad CNC program line
        ; display the error message
    }
}
else ; compile is okay!
{
    AerCompilerDownload() ; 4. Load program into axis processor
}
AerCompilerDestroy() ; 5. Kill the compiler (DON'T FORGET
                        ; THIS, OR YOU GET MEMORY LEAKS)
See “Task Functions” to execute programs.

Some compiler features that may be of interest include the following:
1. Automatic detection of out-of-date objects, including checking of all included files.
2. High-speed “one pass” operation (compiles from source in a few milliseconds per line).
3. “Infinite program size” compilation (see AerCompilerSetQueueMode).

All string variables in MS Visual Basic, passed by reference (ByRef), must be declared as fixed length strings within your program, long enough to hold the string value returned by the function. Also, those string variables which are passed, with another parameter indicating the length of the string variable, must also be fixed length strings, otherwise, you would not be able to pass the length of the string. For example, to declare a fixed length string:

```
DIM sGlobStr as STRING * 50 ; 50 characters long
```
6.2. AerCompilerAbort

AERERR_CODE AerCompilerAbort (HCOMPILER hCompile);

Declare Function AerCompilerAbort Lib “AERSYS.DLL” (ByVal hCompile As Long)
As Long

Parameters

hCompile Handle to an Aerotech compiler session.

This function aborts any current compile or download for this compile session.
6.3. AerCompilerAddDefinesFile

AERERR_CODE AerCompilerAddDefinesFile (HCOMPILER hCompile, LPCTSTR pszFileName);

Declare Function AerCompilerAddDefinesFile Lib “AERSYS.DLL” (ByVal hCompile As Long, ByVal pszFileName As String) As Long

Parameters
   hCompile Handle to an Aerotech compiler session.
   pszFileName Path and name of file to add.

This function adds a “defines file” to the compiler. A defines file is a file that consists of all #define statements. This gives the capability to teach the compiler new commands. If a file is added to the compiler session with this statement, then it is not necessary to include files directly in the program with the #include statement.

The following two files are normally “added” via this function:

The AERPARAM.PGM is a standard file that Aerotech distributes to handle all parameter definitions.

The MCODE.PGM is a standard file that Aerotech distributes to handle all M Code definitions.

All string variables in MS Visual Basic, passed by reference (ByRef), must be declared as fixed length strings within your program, long enough to hold the string value returned by the function. Also, those string variables which are passed, with another parameter indicating the length of the string variable, must also be fixed length strings, otherwise, you would not be able to pass the length of the string. For example, to declare a fixed length string;

   DIM sGlobStr as STRING * 50 ; 50 characters long

Example
   Samples\Lib\VisualBasic\RunPgm.vbp
6.4. **AerCompilerAutoIncludeEx**

AERERR_CODE AerCompilerAutoIncludeEx (HCOMPILER hCompile, TASKMASK mTask, LPCTSTR pszAutomationFile);

Declare Function AerCompilerAutoIncludeEx Lib "AERCMPLR.DLL" (ByVal hCompile As Long, ByVal mTask As Long, ByVal pszAutomationFile As String) As Long

AERERR_CODE AerCompilerAutoInclude (HCOMPILER hCompile, LPCTSTR pszAutomationFile);

Declare Function AerCompilerAutoInclude Lib "AERCMPLR.DLL" (ByVal hCompile As Long, ByVal pszAutomationFile As String) As Long

**Parameters**

- **hCompile** Handle to an Aerotech compiler session.
- **mTask** Task mask that species which task(s) to use (see constants).
- **pszAutomationFile** Program Automation File (May be NULL)

This function loads the Automation File and calls AerCompilerAddDefinesFile on the hCompile for each program that is specified as “Auto Include”.

If pszAxisCfgFile is NULL, the program automation file is automatically retrieved with pszAutomationFile, normally: \U600\Ini\U600Auto.Ini

All string variables in MS Visual Basic, passed by reference (ByRef), must be declared as fixed length strings within your program, long enough to hold the string value returned by the function. Also, those string variables which are passed, with another parameter indicating the length of the string variable, must also be fixed length strings, otherwise, you would not be able to pass the length of the string. For example, to declare a fixed length string:

```
DIM sGlobStr as STRING * 50 ; 50 characters long
```

The AerCompilerAutoInclude is maintained for backward compatibility and calls AerCompilerAutoIncludeEx as follows:

```
AerCompilerAutoIncludeEx (hCompile, TASK_MASK, pszAutomationFile);
```
The name of the Automation file can be retrieved with `AerRegGetFileName`.

**C Language and LabView Constants**

\[ \text{TASKMASK}_1 \]
\[ \text{to} \]
\[ \text{TASKMASK}_4 \]

**VB Constants**

\[ \text{aerTaskMask1} \]
\[ \text{to} \]
\[ \text{aerTaskMask4} \]

**See Also**

`AerAutoProgAddProgram`
`AerCompilerAddDefinesFile`
`AerCompilerAutoRun`
`AerRegGetFileName`
6.5. **AerCompilerAutoRun**

AERERR_CODE AerCompilerAutoRun (HCOMPILER hCompile, HAERCTRL hAerCtrl, LPCTSTR pszAutomationFile);

Declare Function AerCompilerAutoRun Lib “AERCMPLR.DLL” (ByVal hCompile As Long, ByVal hAerCtrl As Long, ByVal pszAutomationFile As String) As Long

**Parameters**

- **hCompile**: Handle to an Aerotech compiler session (may be NULL).
- **hAerCtrl**: Handle to the axis processor card.
- **PszAutomationFile**: Program Automation File (may be NULL).

This function loads the Automation file (\U600\Ini\U600Auto.Ini) plus downloads and executes all necessary programs.

If **hCompile** is NULL, a default compiler handle is created and then discarded.

If **pszAxisCfgFile** is NULL, the axis configuration file is automatically retrieved with **pszAutomationFile** (\U600\Ini\U600AxisCfg.Ini).

---

All string variables in MS Visual Basic, passed by reference (ByRef), must be declared as fixed length strings within your program, long enough to hold the string value returned by the function. Also, those string variables which are passed, with another parameter indicating the length of the string variable, must also be fixed length strings, otherwise, you would not be able to pass the length of the string. For example, to declare a fixed length string:

```vbnet
DIM sGlobStr as STRING * 50 ; 50 characters long
```

The code for the function follows:

```c
AERERR_CODE AERCMPL_DLLENTRY AerCompilerAutoRun (HCOMPILER hCompile, 
HAERCTRL hAerCtrl, 
LPCTSTR pszFile  )
{

    AFX_MANAGE_STATE (AfxGetStaticModuleState() );

    TCHAR szFile[MAX_TEXT_LEN];
    TCHAR szProg[MAX_TEXT_LEN];
    TCHAR szHandle[MAX_PROG_NAME_LEN];
    TCHAR szAuto[MAX_TASKS][512];
    TCHAR szFarcall[MAX_TEXT_LEN];
    TASKINDEX iTask;
    TASKMASK mTask;
    DWORD dwSystem;
    DWORD dwType;
    DWORD dwNumPrograms;
    DWORD iProg;
    DWORD mBit;
```
BOOL     bCloseCompiler = FALSE;
AERERR_CODE eRc;
if( (pszFile == NULL) || (*pszFile == '\0') )
{
    eRc = ::AerRegGetFileName( AER_UNIDEX_DEFAULT, AER_CARD_DEFAULT,
                   AERREGID_AutomationFile, szFile );
}
else
{
    STRCPY( szFile, pszFile );
}

// kill all motion on the tasks
for( iTask = TASKINDEX_1; iTask < MAX_TASKS; iTask++ )
{
    AerTaskProgramAbort( hAerCtrl, iTask );
    AerTaskReset( hAerCtrl, iTask );
    *szAuto[iTask] = '\0';
}

// Create a compiler if none was passed to us
if( hCompile == NULL )
{
    // Create a new compiler
    eRc = AerCompilerOpen( &hCompile );
    RETURN_ON_ERR( eRc );
    // we need to close the compiler since we're creating it
    bCloseCompiler = TRUE;
    // make sure we're using axis names
    eRc = AerCompilerLoadAxisNames( hCompile, NULL );
    JUMP_ON_ERR( eRc, problems );

    // Add auto include information to new compiler
    eRc = AerCompilerAutoInclude( hCompile, szFile );
    JUMP_ON_ERR( eRc, problems );
}

// Get the number of programs
eRc = AerAutoProgGetNumPrograms( szFile, &dwNumPrograms );
RETURN_ON_ERR( eRc );
for( iProg = 0; iProg < dwNumPrograms; iProg++ )
{
    // get program info from ini file
    eRc = AerAutoProgGetProgram( szFile, iProg, &dwSystem, szProg,
                                 &dwType, &mTask );
    BREAK_ON_ERR( eRc );

    // if it is an auto-include get next automation file
if( dwType == AUTOPROG_TYPE_INCLUDE )
{
    continue;
}

// compile the file
eRc = AerCompilerCompileFile( hCompile, szProg,
    AERCMPLR_DEFAULT );
BREAK_ON_ERR( eRc );

switch( dwType )
{
    case AUTOPROG_TYPE_RUN_IMMEDIATE:
    {
        mBit = 0x01;
        for( iTask = TASKINDEX_1; iTask < MAX_TASKS; iTask++ )
        {
            if( mBit & mTask )
            {
                eRc = runImmediate( hAerCtrl, hCompile, iTask );
                BREAK_ON_ERR( eRc );
            }
            mBit <<= 1;
        }
    }
    break;

case AUTOPROG_TYPE_DOWNLOAD_ONLY:
    case AUTOPROG_TYPE_RUN:
    case AUTOPROG_TYPE_RUN_SILENT:
    case AUTOPROG_TYPE_LOAD:
    {
        AerCompilerFileNameToHandle( szProg, szHandle );
        eRc = AerCompilerDownload( hCompile, hAerCtrl, szHandle, 0 );
        BREAK_ON_ERR( eRc );
        if( dwType == AUTOPROG_TYPE_DOWNLOAD_ONLY )
        {
            break;
        }
        mBit = 0x1;
        for( iTask = TASKINDEX_1; iTask < MAX_TASKS; iTask++ )
        {
            if( mBit & mTask )
            {
                if( dwType == AUTOPROG_TYPE_LOAD )
                {
                    eRc = AerTaskProgramAssociate( hAerCtrl, iTask,
                        (PAER_PROG_HANDLE) szHandle );
                    JUMP_ON_ERR( eRc, problems );
                }
                else
                {
                    SPRINTF( szFarcall, "FARCALL "%s" \r\n",
                        szHandle );
                    STRCAT( szAuto[iTask], szFarcall );
                }}}}
mBit <<= 1;
}
break;
}

for( iTask = TASKINDEX_1; iTask < MAX_TASKS; iTask++ )
{
  if( *szAuto[iTask] != '\0' )
  {
    sprintf( szHandle, "AUTOPROG%d", iTask + 1 );
    eRc = AerCompilerCompileLine( hCompile, szAuto[iTask],
          AERCMPLR_DEFAULT );
    BREAK_ON_ERR( eRc );
    eRc = AerTaskProgramDeAssociate( hAerCtrl, iTask );
    BREAK_ON_ERR( eRc );
    eRc = AerCompilerDownload( hCompile, hAerCtrl, szHandle, 0 );
    BREAK_ON_ERR( eRc );
    eRc = AerTaskProgramAssociate( hAerCtrl, iTask,
          (PAER_PROG_HANDLE) szHandle );
    BREAK_ON_ERR( eRc );
    eRc = AerTaskProgramExecute( hAerCtrl, iTask, TASKEXEC_RUN );
    BREAK_ON_ERR( eRc );
  }
}

problems:
  if( bCloseCompiler )
  {
    AerCompilerClose( hCompile );
  }
  return( eRc );

See Also

  AerAutoProgAddProgram
  AerCompilerAddDefinesFile
  AerCompilerAutoIncludeEx
  AerRegGetFileName
6.6.  AerCompilerClose (AerCompilerDestroy)

AERERR_CODE AerCompilerClose (HCOMPILER hCompiler);
Declare Function AerCompilerClose Lib “AERCMPLR.DLL” (ByVal hCompiler As Long) As Long

Parameters

hCompiler  Handle to an Aerotech Compiler session.

This function was formerly AerCompilerDestroy. The function frees all memory associated with a compiler session. The compile handle is no longer valid after a call to this function.

This function does not free memory on the axis processor that may be associated with a program downloaded from an AerCompilerDownload call. If a program has been downloaded successfully with this hCompiler handle, then this program will remain on the axis processor after a call to AerCompilerClose with the hCompiler handle.

See AerProgramFree to unload the CNC program and free the memory on the controller.

See Also

AerCompilerOpen

Example

Samples\Lib\VisualBasic\RunPgm.vbp
6.7. **AerCompilerCompileFile**

AERERR_CODE AerCompilerCompileFile (HCOMPILER hCompiler, LPCTSTR pszFileName, DWORD dwMode);

Declare Function AerCompilerCompileFile Lib “AERCMPLR.DLL” (ByVal hCompiler As Long, ByVal pszFileName As String, ByVal dwMode As Long) As Long

**Parameters**
- *hCompiler*: Handle to an Aerotech compiler session.
- *pszFileName*: Pointer to a null-terminated character string, where the string is the name of the file to compile.
- *dwMode*: Mode (see constants).

This function compiles a file containing CNC program text. Refer to the *UNIDEX 600 Series CNC Programming Manual, P/N EDU158* for a description of valid CNC syntax. The file must be in ASCII format and the following pair of consecutive characters must terminate each line: CR and LF.

This function does not download the compiled file (.ogm) to the axis processor. The compiled program can be downloaded to the axis processor via a call to *AerCompilerDownload*. *AerCompilerCompileFile* may or may not write an object file (.ogm) containing the compiled text to the hard drive depending on the mode parameters set.

*AerCompilerCompileFile* returns either AERERR_NOERR or AERCMPLR_FILE_GENERIC_ERR. Sometimes the compiler generates warnings, but it still returns AERERR_NOERR if the program has warnings and no errors. Information on all errors and warnings can be obtained from the *AerCompilerErrxxxx* functions.

The *dwMode* parameter is a mask that controls certain features of the compile process. The user can set (using the logical ‘or’ operator) any combination of the following bits into the mode. If the user passes zero, all of these bits are off.

The compile consists of a precompile phase and a compile phase that executes in sequence. The precompiler is responsible for performing the “#define” and “#include” statements. The output of the precompiler is given to the compiler and if the compiler completes with no error, then it generates an object file. The object file is the binary program that can be downloaded to the axis processor.

If AERCMPLR PREPROC_ONLY is set, only the precompilation is done. The user can then look at the precompiler output via *AerCompilerGetLineText* and *AerCompilerGetNumOfLines* calls. Or the user can deliver the listing to a file by specifying the AERCMPLR_MAKE_LISTING bit in the mode parameter. However, the user cannot download the program, since it was not fully compiled.

The compiler normally writes the object file to disk using the same name as the filename passed, but with the file extension “OGM.” If *AerCompilerCompileFile* is called at a later time, the compiler compares the dates of the object file (if any) and the program files - (there may be multiple files if there are include statements). If the object file was written later than the entire program files, the compiler uses the object file instead of recompiling the source files. Using the object file results in a significant saving of time - compilation is often five or more times faster. However, object files waste disk space. An object file takes up five to twenty times the disk space as the original source code.
If AERCMPLR_NO_READ_OBJ is set, the compiler will not read an object file. In this mode, the compiler has to recompile the file every time AerCompilerCompileFile is called so there is no timesavings with repeated compiles. Also, this bit has no effect if AERCMPLR_PREPROC_ONLY is set.

If AERCMPLR_NO_WRITE_OBJ is set, the compiler will not write an object file. Also, this bit has no effect if AERCMPLR_PREPROC_ONLY is set.

If AERCMPLR_NO_USE_SRC is set, the compiler will only look for an object file, regardless of whether a source file is present or not. This bit has no effect if AERCMPLR_PREPROC_ONLY is also set.

If AERCMPLR_MAKE_LISTING is set, then the output of the precompiler will be dumped to a disk file. The file will have the same name as the source file, but with a file extension of “.LGM.” Normally, when the compiler reads an object file, it does not look for and read any associated source text. Subsequent calls to AerCompilerGetLineText will return the error “No Source Available.”

Set AERCMPLR_SRC_WITH_OBJ to force the compiler to read in source code when reading in the object file. Keep in mind that this increases compile time by a factor of ten when reading object files. Note that error data is always available, regardless of this flag setting.

C Language and LabView Constants

AERCMPLR_PREPROC_ONLY ; Only runs the preprocessor. No compile
AERCMPLR_NO_READ_OBJ ; Forced recompile, never reads an object file
AERCMPLR_NO_USE_SRC ; Uses only the object file, not the source on disk
AERCMPLR_MAKE_LISTING ; Make a listing file
AERCMPLR_SRC_WITH_OBJ ; Read source when reading object
AERCMPLR_NOWRITE_OBJ ; Never writes an object
AERCMPLR_XXXX

VB Constants

aerComplrPreprocOnly
aerComplrNoReadObj
aerComplrNoUseSrc
aerComplrMakeListing
aerComplrSrcWithObj
aerComplrNoWriteObj
aerComplrXXXX

See Also

AerCompilerCompileLine
AerCompilerDownload

Example

Samples\Lib\VisualBasic\RunPgm.vbp
6.8. AerCompilerCompileLine

AERERR_CODE AerCompilerCompileLine (HCOMPILER hCompiler, LPCTSTR pszLine, DWORD dwMode);

Declare Function AerCompilerCompileLine Lib "AERCMPLR.DLL" (ByVal hCompiler As Long, ByVal pszLine As String, ByVal dwMode As Long) As Long

Parameters

- **hCompiler**: Handle to an Aerotech Compiler session.
- **pszLine**: Pointer to CNC text.
- **dwMode**: Mode (refer to the following description).

This function compiles a string containing CNC program text. Please see the UNINDEX 600 Series CNC Programming Manual, P/N EDU158 for a description of valid CNC syntax. The string must be null terminated and can contain multiple lines separated by a CR/LF pair.

This function behaves similarly to the AerCompilerCompileFile function, except that the text input is from a string, not a file. Therefore, this documentation will only discuss the differences between AerCompilerCompileFile and AerCompilerCompileLine. Please see AerCompilerCompileFile for a full description.

The compiler will not generate an object file in this function, nor will it read an object file. The AERCMPLR_NOOBJ_READ and AERCMPLR_NOOBJ_WRITE bits are ignored if set in the mode parameter. Otherwise, the mode bits behave as described in AerCompilerCompileFile.

It is important to emphasize that compilation of a line is identical to that of a file, except for the source. For example, if the user compiles the line, "#include a.pgm," then the resulting object would be identical to the user doing an AerCompilerCompileFile of the file "a.pgm."

See Also

- AerCompilerCompileFile
- AerCompilerDownload

Example

Samples\Lib\VisualBasic\RunPgm.vbp
6.9. AerCompilerDownload

AERERR_CODE AerCompilerDownload (HCOMPILER hCompile, HAERCTRL hAerCtrl, LPCTSTR psz960Name, DWORD dwFirstUserLineNumber);

Declare Function AerCompilerDownload Lib “AERSYS.DLL” (ByVal hCompile As Long, ByVal hAerCtrl As Long, ByVal psz960Name As String, ByVal dwFirstUserLineNumber As Long) As Long

Parameters

- **hCompile** Handle to an Aerotech Compiler session.
- **hAerCtrl** Handle to the axis processor card.
- **psz960Name** Pointer to a null-terminated character string, which is the name under which the axis processor will store the downloaded program.
- **dwFirstUserLineNumber** Line number of the first user line number to download.

This function is included for backward compatibility. This function calls the AerCompilerDownloadEx function as follows:

```
AerCompilerDownloadEx (hCompile, hAerCtrl, psz960Name, 0, dwFirstUserLineNumber, (DWORD) -1, (PDWORD) NULL, (PDWORD) NULL)
```

This function can be used for downloading programs in normal (non-queue) mode. The **dwFirstUserLineNumber** is typically 0.

See Also

- AerCompilerDownloadEx
- AerCompilerSetQueueMode
- AerCompilerGetQueueMode
- AerCompilerRunImmediate
- AerCompilerCompileFile

Example

```
Samples\Lib\VisualBasic\RunPgm.vbp
```
### 6.10. AerCompilerDownloadEx

AERERR_CODE AerCompilerDownloadEx (HCOMPILER hCompile, HAERCTRL hAerCtrl, LPCSTR psz960Name, DWORD dwStartPacket, DWORD dwStartUserLine, DWORD dwNumPackets, PDWORD pdwNumDownloaded, PDWORD pdwUserLine);

Declare Function AerCompilerDownloadEx Lib “AERSYS.DLL” (ByVal hCompile As Long, ByVal hAerCtrl As Long, ByVal psz960Name As String, ByVal dwStartPacket As Long, ByVal dwStartUserLine As Long, ByVal dwNumPackets As Long, ByRef pdwNumDownloaded As Long, ByRef pdwUserLine As Long) As Long

**Parameters**

- **hCompile**: Handle to an Aerotech compiler session.
- **hAerCtrl**: Handle to the axis processor card.
- **psz960Name**: Name (program handle) for the downloaded file.
- **dwStartPacket**: Which compiler (object) packet to start downloading.
- **dwStartUserLine**: Beginning user line for **dwStartPacket**.
- **dwNumPackets**: Number of packets to download (-1 for all).
- **pdwNumDownloaded**: Total number of packets actually downloaded (can be NULL).
- **pdwUserLine**: Last user line downloaded (can be NULL).

This function downloads a compiled program that was created by a call to AerCompilerCompileFile or AerCompilerCompileLine. It does not know or care about the original program source. The function takes the compiled program and downloads it to the axis processor.

The user can provide a name for the compiled file that will be used by the axis processor to identify the file (referred to as a “program handle”) with **psz960Name**. If **psz960Name** is NULL, then the compiler will use a default name. If the source originated from a file, then the default name is the name of the original source code file, without a path specification. If the source was compiled from a string, then the default name is “?”.

For normal downloads (non-queue mode), the typical call is as follows:

```
AerCompilerDownloadEx (hCompile, hAerCtrl, psz960Name, 0, 1, (DWORD) -1, (PDWORD) NULL, (PDWORD) NULL)
```

Sometimes a program cannot fit into memory on the axis processor card or the CNC program is being fed across a communications link to the controller as it is running. In these cases it is necessary to place the compiler in “queue mode” or “infinite download mode.” This can be accomplished with the AerCompilerSetQueueMode function. When a program is placed in queue mode, only a specified number of lines are allocated. Therefore, in most cases, the whole program cannot be downloaded at once. An error code is returned in this case: AER960RET_PROG_QUEUE_FULL. It is the user’s responsibility to continue to download lines to the axis processor as the queue empties.
Generally, downloading lines is accomplished by the following sequence of events:

```c
...  
// at some point our compiler is set into Queue mode  
eRc = AerCompilerSetQueueMode ( hCompile, TRUE, 1000, 0, FALSE )  
...  
// at some point the downloading process is begun  
dwStartPacket = 0;  
dwStartUserLine = 1;  
dwNumPackets = -1;  // try to download everything  

while ( TRUE )  
{  
eRc = AerCompilerDownloadEx( hCompile, hAerCtrl, psz960Name,  
dwStartPacket, dwStartUserLine, dwNumPackets,  
&dwNumDownloaded, &dwUserLine );  

if( eRc == AERERR_NOERR )  
{  
  break;  // we’re done downloading  
}  

if( eRc != AER960RET_PROG_QUEUE_FULL )  
{  
  break;  // we’ve encountered an unknown error  
}  

// our queue is full, so we will just wait and try again with more lines  
dwStartPacket = dwStartPacket + dwNumDownloaded;  
dwStartUserLine = dwUserLine;  

// hold off execution for a bit, do some other process and comeback  
// Typically the download process is on a separate thread so that we can just  
// sleep,  
// give up our task slice, and try again later  
Sleep(500);  
}  
```

It is also important to know that since the download does not care where the source file existed, `AerCompilerDownloadEx` can download multiple files or compiled text to the same program queue. This is accomplished by specifying the same `psz960Name` for different compiler handles or downloading a newly compiled file/text with the same compile handle.

See Also

- `AerCompilerDownload`
- `AerCompilerSetQueueMode`
- `AerCompilerGetQueueMode`
- `AerCompilerRunImmediate`
- `AerCompilerCompileFile`
6.11. AerCompilerDownloadImmediate

AERERR_CODE AerCompilerDownloadImmediate (HCOMPILER hCompile, HAERCTRL hAerCtrl, TASKINDEX iTask, PDWORD pdwLastLineLoaded);

Declare Function AerCompilerDownloadEx Lib “AERCMPLR.DLL” (ByVal hCompile As Long, ByVal hAerCtrl As Long, ByVal iTask As Long, ByRef pdwLastLineLoaded As Long) As Long

Parameters

- **hCompile**: Handle to an Aerotech compiler session.
- **hAerCtrl**: Handle to the axis processor card.
- **iTask**: Task to download immediate command (see constants).
- **pdwLastLineLoaded**: Last user line downloaded (can be NULL).

This function downloads a compiled program that was created by a call to AerCompilerCompileFile or AerCompilerCompileLine. It does not know or care about the original program source. The function takes the compiled program and downloads it to the axis processor. The program should only contain only Immediate Mode commands: Variable/Parameter/IO assignments, Asynchronous motion commands, offset commands, and spindle commands. Immediate commands are further defined in the U600MMI.hlp file.

For normal downloads, the typical call is as follows:

```
AerCompilerDownloadImmediate( hCompile, hAerCtrl, iTask, (PDWORD) NULL )
```

Generally, an immediate command is executed immediately and another command can be sent right away. However, the DWELL (G4) and WAIT commands are exceptions to this rule. To properly handle this situation the `pdwLastLineLoaded` parameter must be used.
Ordinarily, immediate downloading lines is accomplished by with the following sequence of events:

```c
....
dwLastLineLoaded = 0;
while( TRUE )
{
    eRc = AerCompilerDownloadImmediate( hCompiler, hAerCtrl, GetTask(),
&dwLastLineLoaded );
    if( eRc == AERERR_NOERR || IsStopRequested() )
    {
        break;
    }
    if( eRc == AER960RET_TASKIMMED_EXECUTING )
    {
        // We must be able to execute first one
        if( dwLastLineLoaded != 0 )
        {
            eRc = AERERR_NOERR;
        }
    }
    BREAK_ON_ERR( eRc );
    // hold off execution for a bit, do some other process and comeback
    // Typically the download process is on a separate thread so that we can just sleep,
    // give up our task slice, and try again later
    Sleep(250);
}
```

**C Language and LabView Constants**

- `TASKINDEX_1` to `TASKINDEX_4`

**VB Constants**

- `aerTaskIndex1` to `aerTaskIndex4`

**See Also**

- `AerCompilerDownloadEx`
- `AerCompilerCompileFile`
- `AerCompilerCompileLine`

AERERR_CODE AerCompilerErrGetCode (HCOMPILER hCompiler, DWORD dwErrNum, PDWORD pdwErrorCode);

Declare Function AerCompilerErrGetCode Lib “AERCMPLR.DLL” (ByVal hCompiler As Long, ByVal dwErrNum As Long, ByRef pErrorCode As Long) As Long

Parameters
   hCompiler Handle to an Aerotech compiler session.
   dwErrNum Index of the error (zero based).
   pdwErrorCode Pointer to receive the error code.

This function returns the error code for a particular compiler error. The AerCompilerErrsGetNumOf returns the number of compiler errors. These errors are zero based, if there are five errors, the error numbers are 0-4. Each of these error numbers would be passed to this function, which returns an error code. If there are no program compiler errors, this function returns an error code. The dwErrNum parameter is an index to the compiler error and can range from zero to one less than the value returned by AerCompilerErrGetNumof function.

After receiving the error code, the user can use the AerErrGetMessage function to obtain the error text. However, this text will not have any substitutions in the text line because only the error code is known. We recommend the user use AerCompilerErrGetText to get the complete text of the error. For example, if the user tries to compile the nonexistent file “fake.pgm”, the compile error is AERERR_FILE_NOT_FOUND and a call to AerCompilerErrGetCode, followed by a AerErrGetMessage call on the result will yield the string: “File not found: %s”. However, a call to AerCompilerErrGetText will yield the more informative string: “File not found: fake.pgm”.

The value returned by this function is the error code for this function (if any), not the compiler error.

Example
   Samples\Lib\AexProg.C
6.13. AerCompilerErrGetData

AERERR_CODE AerCompilerErrGetData( HCOMPILER hCompiler, DWORD dwErrNum, PAER_COMPILE_ERROR_DATA pData );

Parameters

- **hCompiler**: Handle to an Aerotech compiler session.
- **dwErrNum**: Index of the error (zero based).
- **pData**: Pointer to structure to receive the error data.

This function returns the full set of data that the compiler maintains concerning a particular error.

If no program has been compiled, the function returns an error code. The **dwErrNum** parameter is an index to the compiler error and can range from zero to one less than the value returned by **AerCompilerErrGetNumof** function.

The data returned is in the form of a structure, whose meaning is described in the Appendix C: Structures and Data Types (under AER_COMPILE_ERROR_DATA).

The value returned by this function is the error code for this function (if any), not the compiler error.

See Also

- **AerCompilerErrGetCode**

Example

- Samples\Lib\AexProg.C

AERERR_CODE AerCompilerErrGetLocText (HCOMPILER hCompiler, DWORD dwErrNum, LPTSTR pszBuffer, DWORD dwBufferSize);

Declare Function AerCompilerErrGetLocText Lib “AERCMPLR.DLL” (ByVal hCompiler As Long, ByVal dwErrNum As Long, ByVal pszBuffer As String, ByVal dwBufferSize As Long) As Long

Parameters

- **hCompiler**: Handle to an Aerotech compiler session.
- **dwErrNum**: Index of the error (zero based).
- **pszBuffer**: Pointer to a character array to receive a copy of the error location text.
- **dwBufferSize**: Size of **pszBuffer**.

All string variables in MS Visual Basic, passed by reference (ByRef), must be declared as fixed length strings within your program, long enough to hold the string value returned by the function. Also, those string variables which are passed, with another parameter indicating the length of the string variable, must also be fixed length strings, otherwise, you would not be able to pass the length of the string. For example, to declare a fixed length string:

```
DIM sGlobStr as STRING * 50 ; 50 characters long
```

This function returns a text string indicating the location of the offending string that caused the error. This string will contain text showing the file, line in the file, and range of characters on the line that caused the error. The line number and character indexes in the text are one-based. If no program has been compiled, the user receives an error. The **dwErrNum** parameter is an index to the error, it can range from zero to one less than the value returned by AerCompilerErrGetNumof function.

The value returned by this function is the error code for this function (if any), not the compiler error. The maximum number of characters copied to **pszBuffer** is 196 characters.

See Also

- AerCompilerErrGetData

Example

```
Samples\Lib\AexProg.C
```
6.15. AerCompilerErrsGetNumOf

AERERR_CODE AerCompilerErrsGetNumOf (HCOMPILER hCompiler, DWORD dwLowestSeverityOf, PDWORD pdwNErrs);

Declare Function AerCompilerErrsGetNumOf Lib “AERCMLPR.DLL” (ByVal hCompiler As Long, ByVal dwLowestSeverityOf As Long, ByRef pdwNErrs As Long) As Long

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCompiler</td>
<td>Handle to an Aerotech compiler session.</td>
</tr>
<tr>
<td>dwLowestSeverityOf</td>
<td>Returns number of all errors “more serious” than this.</td>
</tr>
<tr>
<td>pdwNErrs</td>
<td>Pointer to double word to receive the number of errors.</td>
</tr>
</tbody>
</table>

This function returns the number of errors from a compiler session supplied by the user using the `dwLowestSeverityOf` parameter that instructs what kind of errors to count. The severities are listed in the constants section below in ascending order or seriousness.

For example, providing `dwLowestSeverityOf` as AERERR_TYPE_WARN counts everything but messages, while a value of AERERR_TYPE_MSG counts all of them. By far the most useful setting for `dwLowestSeverityOf` is AERERR_TYPE_ERROR that counts the number of errors that can abort the compile. With `dwLowestSeverityOf` equal to AERERR_TYPE_ERROR and `dwNErrs` is returned as zero, then the compile succeeded and the user can now download the program. Otherwise, if `dwNErrs` is returned as non-zero the compile was unsuccessful. If the compile is unsuccessful, then the user can use the AerCompilerErrGetText or AerCompilerErrGetData functions to get more data on the errors.

If no program has been compiled, the user receives an error. The value returned by this function is the error code for this function (if any), not the compiler error.

C Language and LabView Constants

- `AERERR_TYPE_MSG`
- `AERERR_TYPE_WARN`
- `AERERR_TYPE_ERROR`
- `AERERR_TYPE_NONE`

VB Constants

- `aerErrTypeMsg`
- `aerErrTypeWarn`
- `aerErrTypeError`
- `aerErrTypeNone`

See Also

- AerCompilerErrGetData

Example

- Samples\Lib\AexProg.C
6.16. AerCompilerErrGetText

AERERR_CODE AerCompilerErrGetText (HCOMPILER hCompiler, DWORD dwErrNum, LPTSTR pszBuffer, DWORD dwBufferSize);

Declare Function AerCompilerErrGetText Lib “AERCMPLR.DLL” (ByVal hCompiler As Long, ByVal dwErrNum As Long, ByVal pszBuffer As String, ByVal dwBufferSize As Long) As Long

All string variables in MS Visual Basic, passed by reference (ByRef), must be declared as fixed length strings within your program, long enough to hold the string value returned by the function. Also, those string variables which are passed, with another parameter indicating the length of the string variable, must also be fixed length strings, otherwise, you would not be able to pass the length of the string. For example, to declare a fixed length string:

    DIM sGlobStr as STRING * 50 ; 50 characters long

Parameters

hCompiler Handle to an Aerotech Compiler session.
dwErrNum Index of the error (zero based).
pszBuffer Pointer to a character array to receive a copy of the error text.
dwBufferSize Size of pszBuffer.

This function returns a copy of the error text for a particular compiler error. The dwErrNum parameter is an index to the error number, not the error code, and can range from zero to one less than the value returned by AerCompilerErrGetNumof function.

The value returned by this function is the error (if any) in obtaining the compiler error data - the return has no relationship to any compiler error. The maximum number of characters copied to pszBuffer is determined by, MAX_TEXT_LEN.

C Language and LabView Constants

MAX_TEXT_LEN

VB Constants

aerMaxTextLength

See Also

AerCompilerErrGetData

Example

Samples\Lib\AexProg.C
6.17. AerCompilerFileNameToHandle

AERERR_CODE AerCompilerFileNameToHandle (LPCTSTR pszFileName, LPTSTR pszHandle);

Declare Function AerCompilerFileNameToHandle Lib “AERCMPLR.DLL” (ByVal pszFileName As String, ByVal pszHandle As String) As Long

Parameters

- pszFileName: Filename (may include fully qualified path)
- pszHandle: Pointer to the value to hold the Handle name

All string variables in MS Visual Basic, passed by reference (ByRef), must be declared as fixed length strings within your program, long enough to hold the string value returned by the function. Also, those string variables which are passed, with another parameter indicating the length of the string variable, must also be fixed length strings, otherwise, you would not be able to pass the length of the string. For example, to declare a fixed length string:

```
DIM sGlobStr as STRING * 50 ; 50 characters long
```

This function converts a filename to a “short handle” name. It simply strips all path info from the filename and returns the NAME.EXT.

For Example:

- `c:\u600\programs\test.pgm` ➔ TEST.PGM
- `\programs\test2.pgm` ➔ TEST2.PGM
- `test3.pgm` ➔ TEST.PGM
6.18. AerCompilerGetLineText

AERERR_CODE AerCompilerGetLineText (HCOMPILER hCompiler, DWORD dwLineNum, LPTSTR pszBuffer, DWORD dwBufferSize);

Declare Function AerCompilerGetLineText Lib “AERCMPLR.DLL” (ByVal hCompiler As Long, ByVal dwLineNum As Long, ByVal pszBuffer As String, ByVal dwBufferSize As Long) As Long

Parameters

- **hCompiler**: Handle to an Aerotech compiler session.
- **dwLineNum**: Index of the line (zero based).
- **pszBuffer**: Pointer to a character array to receive a copy of the line.
- **dwBufferSize**: Size of `pszBuffer`.

All string variables in MS Visual Basic, passed by reference (ByRef), must be declared as fixed length strings within your program, long enough to hold the string value returned by the function. Also, those string variables which are passed, with another parameter indicating the length of the string variable, must also be fixed length strings, otherwise, you would not be able to pass the length of the string. For example, to declare a fixed length string:

```
DIM sGlobStr as STRING * 50 ; 50 characters long
```

This function returns a copy of the text of a compiled line. If no program has been compiled, the user receives an error. The maximum number of characters that will be copied to `pszBuffer` is indeterminate, since the compiler has no limitations on the length of a text line. It is the user’s responsibility to insure that `pszBuffer` is large enough to hold the largest file line.

The lines returned are those after preprocessing (“#defines” and “#includes” processed), but before any compiling.

The first line’s index is 0. The line numbering does not count those lines in included files, nor does it count define statements.

**Example**

```
Samples\Lib\AexProg.C
```
6.19. AerCompilerGetNumOfLines

AERERR_CODE AerCompilerGetNumOfLines (HCOMPILER hCompiler, PDWORD pdwNLines);

Function AerCompilerGetNumOfLines Lib “AERCMPLR.DLL” (ByVal hCompiler As Long, ByRef pdwNLines As Long) As Long

Parameters

- **hCompiler**  
  Handle to an Aerotech compiler session.

- **pdwNLines**  
  Pointer to double word to receive the number of lines in the compiled text.

This function returns the number of lines in the file. The lines returned are those after preprocessing (“#defines” and “#includes” processed), but before compiling.

This function does not count lines in include files, nor does it count defines from preprocessor statements.

If no program has been compiled, the user receives an error.

**Example**

Samples\Lib\AexProg.C
6.20. **AerCompilerGetProgVarByName**

AERERR_CODE AerCompilerGetProgVarByName (HCOMPILE hCompile, LPCTSTR pszVarName, PLONG plProgVarNum, PLONG plProgVarSize);

Declare Function AerCompilerGetProgVarByName Lib "AERSYS.DLL" (ByVal hCompile As Long, ByVal pszVarName As String, ByRef plProgVarNum As Long, ByRef plProgVarSize As Long) As Long

**Parameters**

- **hCompile** Handle to an Aerotech compiler session.
- **pszVarName** Name of variable to retrieve.
- **plProgVarNum** Pointer to return the number associated with the specified program variable.
- **plProgVarSize** Pointer to return the size allocated for the variable (greater than 1 if dimensioned as an array).

This function retrieves the number that is used to reference the user defined (DVAR command) program variable. In normal circumstances, plProgVarSize is 1. If the variable was dimensioned as an array, then plProgVarSize will return the size of the array. If the name cannot be found, then plProgVarSize is –1. See AerCompilerGetProgVarTotal for a description of program variables.

**See Also**

AerCompilerGetProgVarTotal
6.21. **AerCompilerGetProgVarByNumber**

AERERR_CODE AerCompilerGetProgVarByNumber (HCOMPILER hCompile, LONG lProgVarNum, LPTSTR pszVarName);

Declare Function AerCompilerGetProgVarByNumber Lib “AERSYS.DLL” (ByVal hCompile As Long, ByVal lProgVarNum As Long, ByRef pszVarName As String) As Long

**Parameters**

- **hCompile**
  - Handle to an Aerotech compiler session.

- **lProgVarNum**
  - The Number of the specified program variable to lookup.

- **pszVarName**
  - Pointer to string to hold the variable name.

All string variables in MS Visual Basic, passed by reference (ByRef), must be declared as fixed length strings within your program, long enough to hold the string value returned by the function. Also, those string variables which are passed, with another parameter indicating the length of the string variable, must also be fixed length strings, otherwise, you would not be able to pass the length of the string. For example, to declare a fixed length string:

```vbnet
dim sGlobStr as STRING * 50 ; 50 characters long
```

Given the program variable number, this function will lookup the name of the variable. See **AerCompilerGetProgVarTotal** for a description of program variables.

**See Also**

- **AerCompilerGetProgVarTotal**
6.22. AerCompilerGetProgVarTotal

AERERR_CODE AerCompilerGetProgVarTotal (HCOMPIER hCompile, PLONG plTotalVars);

Declare Function AerCompilerGetProgVarTotal Lib “AERSYS.DLL” (ByVal hCompile As Long, ByRef plTotalVars As Long) As Long

Parameters

- hCompile: Handle to an Aerotech compiler session.
- plTotalVars: Total number of variables defined in compiled program.

This function returns the total number of program variables (DVAR command) that were allocated in a specified program. Included are only those variables that were allocated with the DVAR CNC statement.

Program variables are stored on the axis processor by number. To set or get a program variable on the card, the AerVarProgramGetDouble / AerVarProgramSetDouble functions need to be used and these only accept the program variable number.

Arrays are always stored consecutively. If a variable called “array” was allocated with 10 elements (i.e. DVAR $array[10]) and the “array” begins at program variable 10, then the array elements within “array” begin at variable 10 and end at 19.

See Also

- AerVarProgramGetDouble
- AerVarProgramSetDouble
6.23. AerCompilerGetQueueMode

AERERR_CODE AerCompilerGetQueueMode (HCOMPILER hCompile, PBOOL pbQueue, PBOOL pbAllocated, PDWORD pdwQueueSize, PDWORD pdwQueueRetain);

Declare Function AerCompilerGetQueueMode Lib “AERSYS.DLL” (ByVal hCompile As Long, ByRef pbQueue As Long, ByRef pbAllocated As Long, ByRef pdwQueueSize As Long, ByRef pdwQueueRetain As Long) As Long

Parameters
   hCompile       Handle to an Aerotech compiler session.
   pbQueue        Is the compiler session setup for queue mode?
   pbAllocated    Has the queue been allocated on the axis processor yet by this compiler session?
   pdwQueueSize   The Number of lines in queue.
   pdwQueueRetain Number of lines to retain in the queue (typically 0).

This function returns the current state of queue-specific variables for the compiler session. See AerCompilerSetQueueMode for more details. The pbQueue and pbAllocated parameters return TRUE or FALSE.

See Also
   AerCompilerSetQueueMode
6.24. **AerCompilerGetStatus**

```c
AERERR_CODE AerCompilerGetStatus (HCOMPILER hCompiler,
                                PAER_COMPILE_STATUS_DATA pData);
```

**Parameters**

- `hCompiler` Handle to an Aerotech compiler session.
- `pData` Pointer to structure to receive the status data.

The data returned is in the form of a structure, whose meaning is described under the structure documentation for `AER_COMPILE_STATUS_DATA`.

**Example**

Samples\Lib\AexProg.C
6.25.  **AerCompilerGetUniqueID**

AERERR_CODE AerCompilerGetUniqueID (HCOMPILER hCompile, PDWORD pdwID);

Declare Function AerCompilerGetUniqueID Lib "AERSYS.DLL" (ByVal hCompile As Long, ByRef pdwID As Long) As Long

**Parameters**

- `hCompile` Handle to an Aerotech compiler session.
- `pdwID` Pointer to hold the compiler ID.

Whenever a file is compiled, a unique ID is stored in the object file. This ID is also downloaded to the axis processor with the program. This is a way to determine if the file on the host PC matches the file that is downloaded on the card. The ID for the file can be retrieved from the axis processor with the `AerProgramGetHeader` function.

**See Also**

- `AerProgramGetHeader`
6.26. AerCompilerLoadAxisNames

AERERR_CODE AerCompilerLoadAxisNames (HCOMPILER hCompile, LPCTSTR pszAxisCfgFile);

Declare Function AerCompilerLoadAxisNames Lib “AERCMPLR.DLL” (ByVal hCompile As Long, ByVal pszAxisCfgFile As String) As Long

Parameters

    hCompile Handle to an Aerotech compiler session.
    pszAxisCfgFile Program Axis Configuration File (may be NULL)

This function loads the Axis Configuration file and assigns the names of the axes to the compiler session. If pszAxisCfgFile is NULL, the axis configuration file is automatically retrieved with AerRegGetFileName.

The name of the Automation file can be retrieved with AerRegGetFileName.

See Also

    AerRegGetFileName
6.27. **AerCompilerOpen (AerCompilerCreate)**

AERERR_CODE AerCompilerOpen (HCOMPILER phCompile);

Declare Function AerCompilerOpen Lib “AERCMPLR.DLL” (ByRef phCompile As Long) As Long

**Parameters**

*phCompile* Pointer to a handle to an Aerotech compiler session.

This function was formerly *AerCompilerCreate*. It initializes a compile session, but does not compile anything - it merely returns a pointer to a compiler object (called a handle). *AerCompilerOpen* must be called prior to any compiling and the object pointer returned by this function must be passed to every other *AerCompiler* function call as the first parameter.

| Each *AerCompilerOpen* call must be paired with a call to *AerCompilerClose* or memory leaks will result. |

**See Also**

*AerCompilerClose*  
*AerCompilerOpenEx*

**Example**

Samples\Lib\VisualBasic\RunPgm.vbp
6.28. **AerCompilerOpenEx**

AERERR_CODE AerCompilerOpenEx (HCOMPILER hCompile);

Declare Function AerCompilerOpenEx Lib “AERSYS.DLL” (ByVal hCompile As Long) As Long

**Parameters**

*hCompile*  Handle to an Aerotech compiler session.

This function increments a reference count for the compiler session. The *hCompile* will not actually be destroyed by a call to *AerCompilerClose* unless its reference count is 0, indicating all other processes (threads) are also done with the compiler session. In multithreaded applications, where each thread has a copy of the *hCompile* handle, the *AerCompilerOpenEx* function can be called within each thread to prevent any one thread from destroying the data associated with *hCompile*.

---

All calls to *AerCompilerOpenEx* should be matched by a call to *AerCompilerClose*.

**See Also**

*AerCompilerClose*

*AerCompilerOpen*
6.29. **AerCompilerRemoveDefinesFile**

AERERR_CODE AerCompilerRemoveDefinesFile (HCOMPILER hCompile, LPCTSTR pszFileName);

Declare Function AerCompilerRemoveDefinesFile Lib “AERSYS.DLL” (ByVal hCompile As Long, ByVal pszFileName As String) As Long

**Parameters**

- `hCompile` Handle to an Aerotech compiler session.
- `pszFileName` Name of file to remove.

This function removes the file from the specified compiler session. See `AerCompilerAddDefinesFile` for more information.

**See Also**

- `AerCompilerAddDefinesFile`
6.30. **AerCompilerRunImmediate**

AERERR_CODE AerCompilerRunImmediate (HCOMPILER hCompiler, HAERCTRL hAerCtrl, LPTSTR pszTextLine, DWORD iTask);

Declare Function AerCompilerRunImmediate Lib “AERCMPLR.DLL” (ByVal hCompiler As Long, ByVal hAerCtrl As Long, ByVal pszTextLine As String, ByVal iTask As Long) As Long

**Parameters**

- **hCompiler** Handle to an Aerotech compiler session.
- **iTask** Task number to run program on.
- **hAerCtrl** Handle to the axis processor card.
- **pszTextLine** Text to compile.

This function compiles the program text and runs it on the given task in “immediate mode”. The string must be null terminated and contain multiple lines, each separated by a CR/LF pair. The axis processor does not retain any information on the program after it executes. The program is executed by the axis processor one line at a time. This call will not download any labels or program variable definitions to the axis processor related to the program.

Immediate mode commands are full defined in the U600MMI.hlp file.

Immediate Mode is intended to allow the user to execute single commands while a program is running on the same task. However, it can be used with no program currently running. Immediate mode also has many limitations. Immediate mode will not allow you to run G1, G2, or G0 motion. When using this function, you don not need to go through the multiple steps of compile, download, and associate – everything is done in one step.

Immediate commands are guaranteed to be executed each “task cycle.” These are variable in execution time (see the “AvgPollTimeSec” global parameter for more info.).

Many CNC statements are illegal in this mode. This function returns an error if the program contains such a statement. For example, program jumps and synchronous motion commands are illegal in this mode, but assignments are legal. Assignments to CNC program variables must be treated with caution, because if the task is currently associated to a different program, then the assignments passed down will refer to other program variables, not the program variables of the current program (because no program variables are downloaded in this mode). Synchronous motion commands are defined as the commands that the axis processor waits for to complete. Asynchronous commands are the commands the axis processor begins execution and continues with the next program block, such as the axis STRM CNC command.
C Language and LabView Constants

    TASKINDEX_1
    to
    TASKINDEX_4

VB Constants

    aerTaskIndex1
    to
    aerTaskIndex4

See Also

    AerCompilerDownload

Example

    Samples\Lib\AexProg.C
6.31. AerCompilerSetQueueMode

AERERR_CODE AerCompilerSetQueueMode (HCOMPILER hCompile, BOOL bQueue, DWORD dwQueueSize, DWORD dwQueueRetain, BOOL bForceQueueAllocate);

Declare Function AerCompilerSetQueueMode Lib “AERSYS.DLL” (ByVal hCompile As Long, ByVal bQueue As Long, ByVal dwQueueSize As Long, ByVal dwQueueRetain As Long, ByVal bForceQueueAllocate As Long) As Long

Parameters
- hCompile: Handle to an Aerotech compiler session.
- bQueue: TRUE to set queue mode, FALSE otherwise.
- dwQueueSize: Number of queue lines to allocate on axis processor.
- dwQueueRetain: Number of lines to retain in the queue (typically 0).
- bForceQueueAllocate: Should allocation of the queue on the axis processor be forced?

Used in conjunction with AerCompilerDownloadEx, this function allows for executing programs of unlimited size. Any program can be downloaded as a queue, independent of whether other programs are loaded as a queue or not.

The main difference between queue and non-queue operation is that in queue mode the axis processor discards lines as it executes them. If lines are loaded at the same rate that they are executed, the user can execute programs of infinite size. See AerCompilerDownloadEx for the sequence of events to download a program in queue mode.

If bQueue is set to TRUE, the program will be downloaded in queue mode. The dwQueueSize parameter specifies the maximum number of lines to allocate on the axis processor. The dwQueueRetain parameter is typically 0.

If the bForceQueueAllocate is set to TRUE, the program queue will be reallocated. If this parameter is FALSE and the program was already allocated as a queue, no allocation takes place. This is useful if more than one program is feeding the same queue.

The bQueue can be set to FALSE to turn off queue mode downloading. The next download will then overwrite all program information on the axis processor. In this case, all other arguments are ignored. The non-queue mode is the default, so the user does not need to call AerCompilerSetQueueMode with FALSE, unless AerCompilerSetQueueMode was previously called with TRUE for that program.

As a program in queue mode executes each line, the line is discarded from the queue. The dwQueueRetain effects this behavior. This parameter keeps up to dwQueueRetain lines in the beginning of the queue. This allows for jumping backward within a queued program. CNC statements which could cause a jump are IF, WHILE, REPEAT, JUMP, etc.
There are side effects to jumping within a queued program. If the line that is being
jumped to is not currently in the queue, then a task fault is generated. Generally, queued
programs should not contain jumps. If they do, then the distance of the jump should be
minimal or very predictable (i.e., simple IF..THEN..ELSE..ENDIF statements or tight
REPEAT/WHILE loops.) A queued program cannot execute a CALL statement
(FARCALL is possible).

See Also

AerCompilerDownload
AerCompilerGetQueueMode

Example

Samples\Lib\AexProg.C

∇ ∇ ∇
CHAPTER 7: AXIS CONFIGURATION FUNCTIONS

7.1. Introduction

Configuring an axis assigns an output command channel (DAC), configures a feedback channel, and assigns them to an axis. After configuration, the servo loop software will use the specified feedback (along with other parameters) to generate a current (or velocity) command. Each axis can have both feedback and Digital-to-Analog Converter (D/A or DAC) information defined. Configuration essentially defines the feedback inputs and the command outputs used by each axis, similar to the functionality provided by the axis configuration wizard in the MMI600 or the SetupWiz utility.

The heart of axis configuration is the AER_CFG_PACKET structure that contains all the data required to configure an axis. Any configuration can be achieved by properly filling one of these structures and calling AerConfig. Likewise, any axis configuration can be read by calling AerConfigGet and examining the returned AER_CFG_PACKET structure. For the user’s convenience, some specialized configuration functions are provided that only accepts the data necessary (i.e., encoder, Hall effect, or resolver).

Both feedback and D/A configuration require a channel number. The channels on the U600 board correspond to channels 1 through 4. Additional channels are dependent upon the board number of the encoder expansion card. Expansion card 1 would be channels 5 through 8, card 2 would be 9 through 12, etc., through all 16 channels.

Axis-specific I/O (CCW/CW, encoder fault, drive fault, AUX out, and drive enable) for encoder-based axes, is associated with the specified encoder channel number. For resolver axes, the axis I/O are associated with the specified D/A channel. The following are examples.

1. Axis 3 is configured to use encoder channel 2 and D/A channel 1. The limits, drive fault, AUX output, and drive enable would be connected to channel 2 (the specified encoder channel) on the BB500/BB501/DR500.

2. Axis 3 is configured to use resolver channel 1 and D/A channel 4. The limits, drive fault, AUX output, and drive enable would be connected to channel 4 (the specified D/A channel) on the BB500/BB501/DR500.

3. Axis 3 is configured for NULL feedback and D/A channel 2 (open loop spindle operation). There would be no encoder feedback or limit switch, etc. inputs. The axis FAULTMASK axis parameter will need to have these bits cleared to prevent erroneous faults. See the U600MMI.hlp file for more information on faults or FAULTMASKS.
7.1.1. Encoder and Resolver/Inductosyn Board Resolution

All of Aerotech’s equipment electronically multiplies the effective line resolution of the encoder (rotary or linear) signal 4 times, additionally, multiplied by any multiplication provided by a MX / MXH multiplier. All parameters specifying encoder lines per revolution should be specified appropriately after the multiplication. For example, Aerotech’s BM brushless motors typically have a 1000 line per revolution rotary encoder mounted on the back of the motor. Any references to this encoder would be as though it were a 4000 line per revolution encoder. This does not apply to resolver (R/D) boards.

Aerotech’s resolver cards may be software configured for 10, 12, 14, or 16 bit resolution, which employ 8,192, 16,384, 32,768 or 65,536 steps per revolution of the resolver. However, the board must also have the respective hardware for that axis configured for the proper resolution by installing the proper RCN (Resistor - Capacitor Network) values.

The UNIDEX 600/620 controllers have a 16 bit D/A converter. Each controller has this analog signal from the DAC scaled to provide a +/- 10 volt command to the drive.

Proper motor and feedback device phasing for UNIDEX 600/620 is achieved when a negative command from the DAC produces clockwise (CW) rotation of the motor (as viewed looking into the front motor shaft). This causes the position and velocity feedback from the motor’s feedback device to produce a positive increase in position as the shaft rotates CW. Meeting these two conditions guarantees a properly phased servo loop that does not provide positive feedback, causing an unsafe run-away condition. The ICMDPOLARITY axis parameter can be used to invert the current command generated by the servo loop. For more information, refer to the controller’s hardware manual.

All of the axis AerConfigxxx functions require an axis index. Use the AXISINDEX_n C constants - the first AXISINDEX is AXISINDEX_1, (or the VB constants – the first axisindex is aerAxisIndex1).

7.1.2. Axis Configuration File Format

Aerotech’s axis configuration file is a text-based INI file. Two available functions to read and write the configuration file are AerConfigReadPacket and AerConfigWritePacket. The configuration file is a text file and can hold as many configurations as desired. The format of the file is as follows (below).
Note that each group shows one feedback type, so not all of these are present simultaneously.

[AxConfig.{AXISINDEX_xxxx+1}]

FBType={Null,Encoder,EncoderHall,Resolver,ResolverHall}
IOType={Null,D2A}
Sp1Type={Null,Encoder,Resolver}
Sp2Type={Null}

:If FBType is Null Then the following values would be present
FBType.Null.Lines=

:If FBType is Encoder Then the following values would be present
FBType.Encoder.Channel=
FBType.Encoder.Lines=
FBType.Encoder.Bounded=

:If FBType is EncoderHall Then the following values would be present
FBType.EncoderHall.Channel=
FBType.EncoderHall.Lines=
FBType.EncoderHall.HallLines=
FBType.EncoderHall.CommOffset=
FBType.EncoderHall.CommChannel=
FBType.EncoderHall.Bounded=

:If FBType is Resolver Then the following values would be present
FBType.Resolver.Channel=
FBType.Resolver.Resolution=
FBType.Resolver.Poles=
FBType.Resolver.CommOffset=
FBType.Resolver.Bounded=

:If FBType is ResolverHall Then the following values would be present
FBType.ResolverHall.Channel=
FBType.ResolverHall.Resolution=
FBType.ResolverHall.HallLines=
FBType.ResolverHall.CommOffset=
FBType.ResolverHall.CommChannel=
FBType.ResolverHall.Bounded=

:If IOType is D2A Then the following values would be present
IOType.D2A.Channel=

:There are no values for Sp1Type Null
:Sp1Type.Null

:If Sp1Type is Resolver Then the following values would be present
Sp1Type.Resolver.Type=5 (AER_SPARETYPE1_RESOLVER)
Sp1Type.Resolver.Channel=
Sp1Type.Resolver.Resolution=
Sp1Type.Resolver.Poles=
Sp1Type.Resolver.CommOffset=
Sp1Type.Resolver.CommOnly=

;If Sp1Type is Encoder Then the following values would be present
Sp1Type.Encoder.Type=3 (AER_SPARETYPE1_ENCODER),
4 (AER_SPARETYPE1_ENCODER_SLAVE)
Sp1Type.Encoder.Channel=
Sp1Type.Encoder.Lines=
Sp1Type.Encoder.VelHomeFlag= BIT MASK: 0x01→Use velocity marker for home.
0x02→Use limits from velocity channel.

;There are no values for Sp2Type Null
;Sp2Type.Null

Although the configuration file can be edited manually with a text editor, it is
strongly suggested that the user use the utilities provided to edit and maintain the axis
configuration information.

/// example /////
/// AxisCfg.ini ////

[AxisConfig.1]
FBType=ResolverHall
IOType=D2A
Sp1Type=Null
Sp2Type=Null
FBType.ResolverHall.Channel=1
FBType.ResolverHall.Resolution=14
FBType.ResolverHall.HallLines=125000
FBType.ResolverHall.CommOffset=0
FBType.ResolverHall.CommChannel=4
FBType.ResolverHall.Bounded=1
IOType.D2A.Channel=1

[AxisConfig.2]
FBType=Encoder
IOType=D2A
Sp1Type=Null
Sp2Type=Null
FBType.Encoder.Channel=2
FBType.Encoder.Lines=4000
FBType.Encoder.Bounded=1
IOType.D2A.Channel=2
7.2. AerConfig

AERERR_CODE AerConfig (HAERCTRL hAerCtrl, AXISINDEX iAxis,
PAER_CFG_PACKET pCfg);

Parameters

hAerCtrl Handle to the axis processor card.
iAxis Axis index to specify which axis to configure. (AXISINDEX_xxxx constants).
pCfg Pointer to AER_CFG_PACKET which contains the configuration information.

This function configures the axis specified in the axis index to the configuration passed in structure pCfg. For the meaning of the configuration packet members, refer to AER_CFG_PACKET in the structures chapter, or see the specific items under the more specialized functions below (see AerConfigEncoder for description of items needed for configuring encoders, AerConfigResolver for resolvers, etc.).

See Also
AerConfigGet

Example

samples\lib\AexCfg.C
7.3. AerConfigDownloadFile

AERERR_CODE AerConfigDownloadFile (HAERCTRL hAerCtrl, LPCTSTR pszFile, AXISMASK mAxis);

Declare Function AerConfigDownloadFile Lib "AERSYS.DLL" (ByVal hAerCtrl As Long, ByVal pszFile As String, ByVal mAxis As Long) As Long

Parameters

hAerCtrl Handle to axis processor card.
pszFile File to download.
mAxis Mask of the axis parameters to download.

This function downloads the axis configuration file to the motion control card. The filename currently in use by the user can be retrieved with the AerRegGetFileName function. Note that it is recommended that the AerSysInitSystem function be used instead of this one.

C Language and LabView Constants

AXISMASK_1 to AXISMASK_16

VB Constants

aerAxisMask1 to aerAxisMask16

See Also

aerRegGetFileName
7.4. AerConfigEncoder

AERERR_CODE AerConfigEncoder (HAERCTRL hAerCtrl, AXISINDEX iAxis,
WORD wEncoderChannel, WORD wD2AChannel, DWORD dwLines,
WORD wBounded);

Declare Function AerConfigEncoder Lib "AERSYS.DLL" (ByVal hAerCtrl As Long,
ByVal iAxis As Long, ByVal wEncoderChannel As Integer, ByVal wD2AChannel As Integer,
ByVal dwLines As Long, ByVal wBounded As Integer) As Long

Parameters

  hAerCtrl  Handle to the axis processor card.
  iAxis     Axis index to specify which axis to configure
           (see constants below).
  wEncoderChannel    Channel number for encoder feedback.
  wD2AChannel     Channel this axis receives its current/velocity command from.
  dwLines        Number of lines per revolution on the encoder after x4 (and
                 MX/MXH multiplication).
  wBounded       1 to activate software travel limits, 0 to disable them.

AerConfigEncoder configures an axis with encoder feedback. The wBounded parameter
enables software limits that generate a fault if the axis is commanded outside the software
limits defined by the CWEOT and CCWEOT axis parameters. In addition, the
SOFTLIMITMODE axis parameter defines the mode of the software limits. Axis limits,
drive faults, drive enable, and AUX I/O are associated with the specified wEncoderChannel.

C Language and LabView Constants

  AXISINDEX_1 to AXISINDEX_16

VB Constants

  aerAxisIndex1 to aerAxisIndex16

See Also

  aerConfigGetEncoder

Example

  samples\lib\AexCfg.C
## 7.5. AerConfigEncoderHall

AERERR_CODE AerConfigEncoderHall (HAERCTRL hAerCtrl, AXISINDEX iAxis, WORD wEncoderChannel, WORD wD2AChannel, DWORD dwLines, WORD wCommChannel, DWORD dwCycleLines, WORD wCommOffset, WORD wBounded);

Declare Function AerConfigEncoderHall Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal iAxis As Long, ByVal wEncoderChannel As Integer, ByVal wD2AChannel As Integer, ByVal dwLines As Long, ByVal wCommChannel As Integer, ByVal dwCycleLines As Long, ByVal wCommOffset As Integer, ByVal wBounded As Integer) As Long

### Parameters
- **hAerCtrl** Handle to the axis processor card.
- **iAxis** Axis index to specify which axis to configure (see constants).
- **wEncoderChannel** Channel number for encoder feedback.
- **wD2AChannel** Channel this axis receives its current/velocity command from.
- **dwLines** Number of lines per revolution on the encoder (after \(x4\) and MX/MXH multiplication).
- **wCommChannel** Channel this axis looks for Hall effect feedback.
- **dwCycleLines** Number of encoder lines per electrical cycle (Hall effect) after all multiplication.
- **wCommOffset** Commutation offset.
- **wBounded** 1 to activate software travel limits, 0 to disable them.

The *AerConfigEncoderHall* configures a brushless motor axis with encoder and Hall effect feedback (Hall effect is for commutation only) feedback. The *dwLines* parameter specifies the number of lines per revolution of the motor (after multiplication). The *dwCycleLines* parameter specifies the number of encoder lines (after multiplication) that is equivalent to one electrical cycle of the brushless motor. The *wBounded* parameter enables software limits to generate a fault if the axis is commanded outside the software limits defined by the CWEOT and CCWEO'T axis parameters. Also, the SOFTLIMITMODE axis parameter defines the mode of the software limits. Axis limits, drive faults, drive enable, and AUX I/O are associated with the specified *wEncoderChannel*. The *wCommOffset* parameter specifies a commutation offset in counts \((1,024 \text{ counts} = 360°)\) that is added to the motor commutation angle, to align the motor’s rotor to the hall sequence specified in the U600MMI.hlp file.

### C Language and LabView Constants
- **AXISINDEX_1** to **AXISINDEX_16**

### VB Constants
- **aerAxisIndex1** to **aerAxisIndex16**

### See Also
- AerConfigGetEncoderHall

### Example
- Samples\Lib\AexCfg.C
7.6. **AerConfigGet**

AERERR_CODE AerConfigGet (HAERCTRL hAerCtrl, AXISINDEX iAxis, PAER_CFG_PACKET pCfg);

**Parameters**
- **hAerCtrl** Handle to the axis processor card.
- **iAxis** Axis on which to obtain data on (AXISINDEX_**** constants).
- **pCfg** Pointer to AER_CFG_PACKET which will return the configuration information.

This function returns the configuration data structure for a given axis. For the meaning of the configuration packet members, refer to AER_CFG_PACKET in Appendix C: Structures, or see the specific items under the more specialized function below (refer to AerConfigEncoder for description of items used for configuring encoders, AerConfigResolver for resolvers, etc.)

**See Also**
- AerConfig

**Example**
- samples\lib\AexCfg.C
7.7. **AerConfigGetEncoder**

AERERR_CODE AerConfigGetEncoder (HAERCTRL hAerCtrl, AXISINDEX iAxis, PWORD pwEncoderChannel, PWORD pwD2AChannel, PDWORD pdwLines, PWORD pwBounded);

Declare Function AerConfigGetEncoder Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal iAxis As Long, ByRef pwEncoderChannel As Integer, ByRef pwD2AChannel As Integer, ByRef pdwLines As Long, ByRef pwBounded As Integer ) As Long

**Parameters**
- **hAerCtrl** Handle to the axis processor card.
- **iAxis** Axis on which to obtain data on (see constants).
- **pwEncoderChannel** Pointer to channel number for encoder feedback.
- **pwD2AChannel** Pointer to channel this axis receives its current/velocity command from.
- **pdwLines** Pointer to Number of lines per revolution on the encoder after MX/MXH and controller \( \times 4 \) multiplication.
- **pwBounded** 1 to activate software travel limits, 0 to disable them.

The **AerConfigGetEncoder** function obtains configuration data on an axis with encoder feedback. The **wBounded** parameter enables software limits that generate a fault if the axis is commanded outside the software limits defined by the **CWEOT** and **CCWEOT** axis parameters. Also, the **SOFTLIMITMODE** axis parameter defines the mode of the software limits.

**C Language and LabView Constants**
- **AXISINDEX_1** to **AXISINDEX_16**

**VB Constants**
- **aerAxisIndex1** to **aerAxisIndex16**

**See Also**
- **AerConfigEncoder**

**Example**
Samples\lib\AexCfg.C
7.8. AerConfigGetEncoderHall

AERERR_CODE AerConfigGetEncoderHall (HAERCTRL hAerCtrl, AXISINDEX iAxis, PWORD pwEncoderChannel, PWORD pwD2AChannel, PWORD pdwLines, PWORD pwCommChannel, PWORD pdwCycleLines, PWORD pwCommOffset, PWORD pwBounded);

Declare Function AerConfigGetEncoderHall Lib "AERSYS.DLL" (ByVal hAerCtrl As Long, ByVal iAxis As Long, ByRef pwEncoderChannel As Integer, ByRef pwD2AChannel As Integer, ByRef pdwLines As Long, ByRef pwCommChannel As Integer, ByRef pdwCycleLines As Long, ByRef pwCommOffset As Integer, ByRef pwBounded As Integer) As Long

Parameters

hAerCtrl Handle to the axis processor card.
iAxis Axis on which to obtain data on (see constants).
pwEncoderChannel Pointer to channel number for encoder feedback.
pwD2AChannel Pointer to channel this axis receives its current/velocity command from.
pdwLines Pointer to Number of lines per revolution of the encoder after after MX / MXH and controller x4 multiplication.
pwCommChannel Pointer to channel this axis looks for Hall effect feedback.
pdwCycleLines Pointer to Number of counts per electrical cycle (Hall effect) after MX / MXH and controller x4 multiplication.
pwCommOffset Pointer to commutation offset.
pwBounded Pointer to bound, 1 to activate software travel limits, 0 to disable them.

The AerConfigGetEncoderHall obtains configuration data on an axis with encoder and Hall effect (for motor commutation only) feedback. The wBounded parameter enables software limits that generate a fault if the axis is commanded outside the software limits defined by the CWEOT and CCWEOT axis parameters. Also, the SOFTLIMITMODE axis parameter defines the mode of the software limits.

C Language and LabView Constants

AXISINDEX_1
AXISINDEX_16

VB Constants

aerAxisIndex1
toaerAxisIndex16

See Also

aerConfigEncoderHall

Example

Samples\Lib\AexCfg.C
7.9. AerConfigGetHallPolePairs

AERERR_CODE AerConfigGetHallPolePairs( HAERCTRL hAerCtrl, AXISINDEX iAxis, PWORD pwChannel, PWORD pwD2AChannel, PDWORD pdwLines, PWORD pwCommChannel, PDWORD pdwLinesRev, PWORD pwPolePairs, PWORD pwCommOffset, PWORD pwBound );

Declare Function AerConfigGetHallPolePairs Lib "AerSys.Dll" ( ByVal hAerCtrl as Long, ByVal iAxis as Long, ByRef pwChannel as Short, ByRef pwD2AChannel as Short, ByRef pdwLines as Short, ByRef pwCommChannel as Short, ByRef pdwLinesRev as Long, ByRef pwPolePairs as Short, ByRef pwCommOffset as Short, ByRef pwBound as Short) as Long

Parameters
- hAerCtrl: Handle to an Aerotech control
- iAxis: Axis Index whose configuration to read (See constants below)
- pwChannel: Encoder feedback channel this axis is configured for
- pwD2AChannel: DAC channel for the command output this axis is configured for
- pdwLines: Returns the number of lines per revolution of the encoder after multiplication by the MXH and/or controller.
- pwCommChannel: Returns the channel number for the Hall effect feedback signals
- pdwLinesRev: Returns the number of lines per electrical cycle of the motor after multiplication by the MXH and/or controller.
- pwPolePairs: Returns the number of pairs of poles of the motor
- pwCommOffset: Returns the commutation offset, where; 360°=1,024
- pwBound: Returns 1 if software limits are enabled, 0 if they are disabled

This function will return the configuration of an axis with encoder and Hall effect (Hall effect is for commutation only) feedback, where the number of lines per electrical cycle is not an integer. The dwLines parameter specifies the number of lines per revolution of the motor (after the X4 multiplication of the controller). The dwLinesRev parameter specifies the number of encoder lines (after X4 multiplication of the controller) that is equivalent to one revolution of the motor. The wBounded parameter enables software limits that generate a fault if the axis is commanded outside the software limits defined by the CWEOT and CCWEOT axis parameters. Also, the SOFTLIMITMODE axis parameter defines the mode of the software limits. Axis limits, drive faults, drive enable, and AUX I/O are associated with the specified wChannel. The wCommOffset parameter specifies a commutation offset, where 1024=360 degrees, that is added to the motor commutation angle.

C and LabView Constants
- AXISINDEX#

VB Constants
- aerAxisIndex#

See Also
- AerConfigHallPolePairs
7.10. AerConfigGetMaster

AERERR_CODE AerConfigGetMaster (HAERCTRL hAerCtrl, AXISINDEX iAxis, PAER_CFG_MASTER_PACKET pMaster);

Parameters
- **hAerCtrl** Handle to the axis processor card.
- **iAxis** Axis on which to obtain data on (AXISINDEX_xxxx constants).
- **pMaster** Pointer to AER_CFG_MASTER PACKET which contains the master/slave information.

The AerConfigGetMaster obtains master configuration data for the specified axis. This function is the same as the AerConfigGetMasterAxis function, except this function uses a structure as opposed to individual parameters.

See Also
- AerConfigGetMasterAxis
- AerConfigMaster
- AerConfigMasterAxis
- AerCamTablexxxx

Example
- samples\lib\AexCam.C
7.11. AerConfigGetMasterAxis

AERERR_CODE AerConfigGetMasterAxis( HAERCTRL hAerCtrl, AXISINDEX iAxis, PWORD pwType, PWORD pwChannel, PDWORD pdwData );

Declare Function AerConfigGetMasterAxis Lib "AERSYS.DLL" ( ByVal hAerCtrl As Long, ByVal iAxis As Long, ByRef pwType As Integer, ByRef pwChannel As Integer, ByRef pdwData As Long ) As Long

Parameters
  hAerCtrl    Handle to the axis processor card.
  iAxis       Axis index to specify which axis to configure (see constants below).
  pwType      Pointer to type of feedback (resolver, encoder, virtual or null, see constants below).
  pwChannel   Pointer to feedback channel number (1, 2, …16).
  pdwData     Pointer to feedback resolution (lines/rev or 10, 12, 14, 16 resolver bits).

The AerConfigGetMasterAxis obtains master configuration data for the specified axis. This function is the same as the AerConfigGetMaster function, except this function uses individual parameters as opposed to a structure for retrieving the master axis configuration. See the appendices for Constants and Data Types and Appendix C: Structures for the definitions of the pwType parameter.

C Language and LabView Constants
  AXISINDEX_1 to AXISINDEX_16
  AER_MFBTYPE_XXXX

VB Constants
  aerAxisIndex1 to aerAxisIndex16
  aerMFBTypexxxx

See Also
  AerConfigGetMaster
  AerConfigMaster
  AerConfigMasterAxis
  AerCamTablexxxx

Example
  Samples\Lib\AexCfg.C
7.12. AerConfigGetResolver

AERERR_CODE AerConfigGetResolver (HAERCTRL hAerCtrl, AXISINDEX iAxis,
PWORDDVAR pwResolverChannel, PWORDDVAR pwD2AChannel, PWORDDVAR
pwResolution, PWORDDVAR pwPoles, PWORDDVAR pwCommOffset, PWORDDVAR
pwBounded);

Declare Function AerConfigGetResolver Lib “AERSYS.DLL” (ByVal
hAerCtrl As Long,
ByVal iAxis As Long, ByRef pwResolverChannel As Integer, ByRef
pwD2AChannel As Integer, ByRef pwResolution As Integer, ByRef
pwPoles As Integer, ByRef pwCommOffset As Integer, ByRef
pwBounded As Integer) As Long

Parameters

hAerCtrl Handle to the axis processor card.
iAxis Axis on which to obtain data on
(see constants).
pwResolverChannel Pointer to channel number for resolver feedback.
pwD2AChannel Pointer to channel this axis receives its
current/velocity command from.
pwResolution Pointer to resolution of resolver (number of bits in
R/D converter, will be 10, 12 14, 16, based on R/D
cnverter hardware used).
pwPoles Pointer to number of motor poles (must be even, use
0 for a DC motor).
pwCommOffset Pointer to commutation offset (1,024 = 360°).
pwBounded Pointer to bounded, 1 to activate software travel
limits, 0 to disable them.

AerConfigGetResolver obtains configuration data on an axis with resolver feedback. The
wBounded parameter enables software limits that generate a fault if the axis is
commanded outside the software limits defined by the CWEOT and CCWEOT axis
parameters. In addition, the SOFTLIMITMODE axis parameter defines the mode of the
software limits.

C Language and LabView Constants

AXISINDEX_1
to
AXISINDEX_16

VB Constants

aerAxisIndex1
to
aerAxisIndex16

See Also

AerConfigResolver
AerConfigResolverHall
AerConfigGetResolverHall

Example

Samples\Lib\AexCfg.C
7.13. AerConfigGetResolverHall

AERERR_CODE AerConfigGetResolverHall (HAERCTRL hAerCtrl, AXISINDEX iAxis, PWORD pwResolverChannel, PWORD pwD2AChannel, PWORD pwResolution, PWORD pwCommChannel, PDWORD pdwCycleLines, PWORD pwCommOffset, PWORD pwBounded);

Declare Function AerConfigGetResolverHall Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal iAxis As Long, ByRef pwResolverChannel As Integer, ByRef pwD2Achannel As Integer, ByRef pwResolution As Integer, ByRef pwCommChannel As Integer, ByRef pdwCycleLines As Long, ByRef pwCommOffset As Integer, ByRef pwBounded As Integer) As Long

Parameters

- **hAerCtrl**: Handle to the axis processor card.
- **iAxis**: Axis on which to obtain data on (see constants).
- **pwResolverChannel**: Pointer to channel number for resolver feedback.
- **pwD2AChannel**: Pointer to channel this axis receives its current/velocity command from.
- **pwResolution**: Pointer to resolution of resolver (number of bits in R/D converter, must be 10, 12, 14, 16, based on R/D converter hardware used).
- **pwCommChannel**: Pointer to channel. This axis looks for Hall effect feedback.
- **pdwCycleLines**: Pointer to number of lines per electrical cycle (Hall effect) after MX/MXH and controller ×4 multiplication.
- **pwCommOffset**: Pointer to commutation offset.
- **pwBounded**: Pointer to bounded, 1 to activate software travel limits, 0 to disable them.

The **AerConfigGetResolverHall** obtains configuration data on an axis with resolver and Hall effect (commutation only) feedback. The **pwBounded** parameter enables software limits that generate a fault if the axis is commanded outside the software limits defined by the **CWEOT** and **CCWEOT** axis parameters. Also, the **SOFTLIMITMODE** axis parameter defines the mode of the software limits.

C Language and LabView Constants

AXISINDEX_1 to AXISINDEX_16

VB Constants

- **aerAxisIndex1** to aerAxisIndex16

See Also

**AerConfigResolverHall**

Example

Samples\Lib\AexCfg.C
7.14. AerConfigHallPolePairs

AERERR_CODE AerConfigHallPolePairs( HAERCTRL hAerCtrl, AXISINDEX iAxis, 
WORD wChannel, WORD wD2AChannel, DWORD dwLines, WORD wCommChannel, DWORD dwLinesRev, WORD wPolePairs, WORD wCommOffset, WORD wBound);

Declare Function AerConfigHallPolePairs Lib "AerSys.Dll" ( ByVal hAerCtrl as Long, 
ByVal iAxis as Long, ByVal wChannel as Short, ByVal wD2AChannel as Short, ByVal dwLines as Long,  ByVal wCommChannel as Short, 
ByVal dwLinesRev Long, ByVal wPolePairs as Short, ByVal wCommOffset as Short, ByVal wBound as Short) as Long

Parameters
hAerCtrl Handle to an Aerotech control
iAxis Axis Index to specify the axis to configure (See constants below)
wChannel Encoder feedback channel for this axis
wD2Achannel DAC channel for the command output to the axis
dwLines Number of lines per revolution of the encoder after multiplication by the MXH and/or controller.
wCommChannel The channel number for the Hall effect feedback signals
wLinesRev The number of lines per revolution of the motor after multiplication by the MXH and/or controller.
wPolePairs The number of pairs of poles of the motor
wCommOffset The commutation offset, where; 360 degrees=1024
wBound 1 to enable software limits, 0 disables them

This function will configures an axis with encoder and Hall effect (Hall effect is for commutation only) feedback, where the number of lines per electrical cycle is not an integer. The dwLines parameter specifies the number of lines per revolution of the motor (after the X4 multiplication of the controller). The dwLinesRev parameter specifies the number of encoder lines (after X4 multiplication of the controller) that is equivalent to one revolution of the motor. The wBounded parameter enables software limits that generate a fault if the axis is commanded outside the software limits defined by the CWEOT and CCWEOT axis parameters. Also, the SOFTLIMITMODE axis parameter defines the mode of the software limits. Axis limits, drive faults, drive enable, and AUX I/O are associated with the specified wChannel. The wCommOffset parameter specifies a commutation offset, where 1,024=360°, that is added to the motor commutation angle.

C and LabView Constants
AXISINDEX#

VB Constants
aerAxisIndex#

See Also
AerConfigGetHallPolePairs
7.15. AerConfigMaster

AERERR_CODE AerConfigMaster (HAERCTRL hAerCtrl, AXISINDEX iAxis, PAER_CFG_MASTER_PACKET pMaster);

There is no VB function for AerConfigMaster, see Section 7.16. AerConfigMasterAxis (page 7-19) for VB information.

**Parameters**

- **hAerCtrl** Handle to the axis processor card.
- **iAxis** Axis index to specify which axis (slave) is to have its master configured (AXISINDEX_xxxx constants).
- **pMaster** Pointer to AER_CFG_MASTER PACKET which contains the master/slave information.

This function is used in a number of contexts, such as cam table motion, auxiliary tables, and strip charts. It defines how a particular axis will fill its “master position.” Unless this function is called, an axis’ master position remains at zero. After this function is called, the master position will be updated each servo cycle to be the actual or commanded position of a specified axis. The axis (where the position is derived from) may be the same as or different than that specified by the **iAxis** parameter.

By configuring **iAxis** as virtual, the commanded rather than actual position is used as the master position.

For cam tables, the master position is used as the command for the slave to follow. The slave (**iAxis**) interprets this command in the same way that a “normal” axis interprets its own command. Therefore, similar to the “normal” axis, the slave needs to know the channel to obtain the feedback and the conversion information needed to convert the feedback into a position. This data must be placed into the structure pointed to by **pMaster**.

It is possible to define a master configuration for a slave, different than the master axis’ actual configuration. Normally, one would call AerConfigGet to retrieve the master axis’ configuration and copy the required data into the master packet passed to AerConfigMaster.

This function is also used by auxiliary tables to specify which axis gets positioned and monitored for firing of the auxiliary output. In this case, the master configuration would normally be set to the same setting as **iAxis**. Thus, the master position of **iAxis** will be the same as the position of **iAxis**.

**See Also**

- AerAuxTablexxxx
- AerCamTablexxxx
- AerConfigGetMaster
- AerConfigGetMasterAxis
- AerConfigMasterAxis

**Example**

Samples\Lib\ AexCam.C
### 7.16. AerConfigMasterAxis

**AERERR_CODE** AerConfigMasterAxis (HAERCTRL hAerCtrl, AXISINDEX iAxis, WORD wType, WORD wChannel, DWORD dwData);

Declare Function AerConfigMasterAxis Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal iAxis As Long, ByVal wType As Integer, ByVal wChannel As Integer, ByVal dwData As Long) As Long

**Parameters**

- **hAerCtrl** Handle to the axis processor card.
- **iAxis** Axis index to specify which axis (slave) to configure its master (see constants).
- **wType** Type of feedback (resolver, encoder, virtual or null, or see constants).
- **wChannel** Feedback channel number (1, 2,…, 16) or axis index of master, if wType = virtual.
- **dwData** Resolution of feedback device (lines/rev. after MX/MXH and controller x4 multiplication or 10, 12, 14, or 16 resolver bits).

This function is used in a number of contexts, such as cam table motion, auxiliary tables, and strip charts. Its purpose is to define how this axis is to fill its “master position.” Unless this function is called, an axis’ master position remains at zero. After this function is called, the master position will be updated each servo cycle to be the actual or commanded position of a specified axis. The axis (where the position is derived from) may be the same as or different than that specified by the iAxis parameter.

By configuring iAxis as virtual, the commanded rather than actual position is used as the master position.

For cam tables the master position is used as the command for the slave to follow. The slave (iAxis) interprets this command in the same way that a “normal” axis interprets its own command. Therefore, similar to the “normal” axis, the slave needs to know the channel to obtain the feedback and the conversion information needed to convert the feedback into a position. This is indicated by the wChannel parameter.

It is possible to define a master configuration for a slave different than the master axis’s actual configuration. Normally, one would call AerConfigGet to retrieve the master axis’s configuration and copy the required data into the master packet passed to AerConfigMaster.

This function is used by auxiliary tables to specify which axis to monitor its position to fire the auxiliary output. In this case, the master configuration would normally be set to the same setting as iAxis. Thus, the master position of iAxis will be the same as the position of iAxis.
C Language and LabView Constants
AXISINDEX_1
to
AXISINDEX_16

AER_MFBTYPE_XXXX

VB Constants
aerAxisIndex1
to
aerAxisIndex16
aerMFBTypexxxx

See Also
AerAuxTablexxxx
AerCamTablexxxx
AerConfigGetMaster
AerConfigGetMasterAxis
AerConfigMaster

Example
samples\lib\AexCfg.C
7.17. AerConfigMasterAuto

AERERR_CODE AerConfigMasterAuto (HAERCTRL hAerCtrl, BOOL bUseCommand,
AXISINDEX iMaster, AXISINDEX iSlave);

Declare Function AerConfigMasterAuto Lib “AERSYS.DLL” (ByVal hAerCtrl As Long,
ByVal bUseCommand As Long, ByVal iMaster As Long, ByVal iSlave As Long) As Long

Parameters
  hAerCtrl    Handle to the axis processor card.
  bUseCommand Always equal to FALSE.
  iMaster     Axis index to specify the master axis (see constants).
  iSlave      Axis index to specify the slave axis (see constants).

This function is used to define how the slave axis will obtain its “master position” from
the master axis.

If bUseCommand is equal to true, the command rather than the actual position is used
as the master position.

If the master axis is configured as a stepper or virtual, the commanded position is
used as the master position.

C Language and LabView Constants
  AXISINDEX_1
to
  AXISINDEX_16

VB Constants
  aerAxisIndex1
to
  aerAxisIndex16
7.18. AerConfigReadPacket

AERERR_CODE AerConfigReadPacket (LPCTSTR pszFile, AXISINDEX iAxis,
PAER_CFG_USER_INFO pUserCfg);

**Parameters**

- *pszFile* Name of axis configuration file.
- *iAxis* Axis index to retrieve axis configuration info (AXISINDEX_xxxx constant).
- *pUserCfg* Pointer to AER_CFG_USER_INFO which will hold the configuration information.

This function reads the specified configuration file and returns the configuration packet for the given axis. The AER_CFG_USER_INFO is compatible with an AER_CFG_PACKET. It can be passed to any function that accepts AER_CFG_PACKET. See Section 7.1.2.: Axis Configuration File Format, for details on the layout of the file.

**See Also**

- AerConfig
- AerConfigWritePacket
- AerConfigReadPacketEx
7.19. AerConfigReadPacketEx

AERERR_CODE AerConfigReadPacketEx (LPCTSTR pszFile, BOOL bTemplate,
AXISINDEX iTemplateAxis, AXISINDEX iAxis,
PAER_CFG_USER_INFO pUserCfg);

Parameters

pszFile Name of axis configuration file.
bTemplate Determines if the configuration packet should be read back as a template.
iTemplateAxis Axis to use as a template (AXISINDEX_xxxx constant).
iAxis Axis index to retrieve axis configuration information (AXISINDEX_xxxx constant).
pUserCfg Pointer to AER_CFG_USER_INFO which will hold the configuration information.

This function reads the specified configuration file and returns the configuration packet for the given axis. If bTemplate is TRUE, then the axis configuration specified by iTemplateAxis is used. In this situation, any channel information defaults to the axis specified by iAxis. This is a way to make copies of an existing axis configuration from a file.

The AER_CFG_USER_INFO is compatible with an AER_CFG_PACKET. It can be passed to any function that accepts an AER_CFG_PACKET. See Section 7.1.2.: Axis Configuration File Format for details on the layout of the file.

See Also

AerConfigReadPacket
7.20. AerConfigResolver

AERERR_CODE AerConfigResolver (HAERCTRL hAerCtrl, AXISINDEX iAxis, WORD wResolverChannel, WORD wD2AChannel, WORD wResolution, WORD wPoles, WORD wCommOffset, WORD wBounded);

Declare Function AerConfigResolver Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal iAxis As Long, ByVal wResolverChannel As Integer, ByVal wD2AChannel As Integer, ByVal wResolution As Integer, ByVal wPoles As Integer, ByVal wCommOffset As Integer, ByVal wBounded As Integer) As Long

Parameters
  hAerCtrl Handle to the axis processor card.
  iAxis Axis index to specify which axis to configure (see constants below).
  wResolverChannel Channel number for resolver feedback.
  wD2AChannel Channel this axis receives its current/velocity command from.
  wResolution Resolution of resolver (number of bits in R/D converter, must be 10, 12, 14, 16, based on R/D converter hardware used).
  wPoles Number of poles (Must be even - use 0 for DC motor).
  wCommOffset Commutation offset (1,024 = 360°).
  wBounded 1 to activate software travel limits, 0 to disable them.

AerConfigResolver configures an axis with a resolver. The wBounded parameter enables software limits that generate a fault if the axis is commanded outside the software limits defined by the CWEOT and CCWEOT axis parameters. In addition, the SOFTLIMITMODE axis parameter defines the mode of the software limits. Axis limits, drive fault, drive enable, and AUX I/O are associated with the specified wD2AChannel. The wCommOffset parameter is subtracted from the current resolver value before the resolver position is used to commutate the motor.

C Language and LabView Constants
  AXISINDEX_1 to AXISINDEX_16

VB Constants
  aerAxisIndex1 to aerAxisIndex16

See Also
  AerConfigGetResolver
  AerConfigResolverHall
  AerConfigGetResolverHall

Example
  Samples\Lib\AexCfg.C
7.21. AerConfigResolverHall

AERERR_CODE AerConfigResolverHall (HAERCTRL hAerCtrl, AXISINDEX iAxis, WORD wResolverChannel, WORD wD2AChannel, WORD wResolution, WORD wCommChannel, DWORD dwCycleLines, WORD wCommOffset, WORD wBounded);

Declare Function AerConfigResolverHall Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal iAxis As Long, ByVal wResolverChannel As Integer, ByVal wD2AChannel As Integer, ByVal wResolution As Integer, ByVal wCommChannel As Integer, ByVal dwCycleLines As Long, ByVal wCommOffset As Integer, ByVal wBounded As Integer) As Long

Parameters

hAerCtrl       Handle to the axis processor card.
iAxis          Axis index to specify which axis to configure (see constants below).
wResolverChannel Channel number for resolver feedback.
wD2AChannel    Channel this axis receives its current/velocity command from.
wResolution    Resolution of resolver (number of bits in R/D converter, must be 10, 12, 14, 16, based on R/D converter hardware used).
wCommChannel   Hall effect feedback channel number.
dwCycleLines   Number of lines per electrical cycle.
wCommOffset    Commutation offset.
wBounded       1 to activate software travel limits, 0 to disable them.

The AerConfigResolverHall configures an axis with resolver and Hall effect (commutation only) feedback. The wBounded parameter enables software limits that will generate a fault if the axis is commanded outside the software limits defined by the CWEOT and CCWEOT axis parameters. Also, the SOFTLIMITMODE axis parameter defines the mode of the software limits. Axis limits, drive fault, drive enable, and AUX I/O are associated with the specified wD2AChannel. The wCommOffset parameter specifies a commutation offset in counts (1,024 counts = 360°) that is added to the motor commutation angle.

C Language and LabView Constants

AXISINDEX_1 to AXISINDEX_16

VB Constants

aerAxisIndex1 to aerAxisIndex16

See Also

AerConfigGetResolverHall

Example

Samples\Lib\AexCfg.C
7.22. AerConfigWritePacket

AERERR_CODE AerConfigWritePacket (LPTSTR pszFile, AXISINDEX iAxis,
PAER_CFG_PACKET pCfg);

**Parameters**

- **pszFile**  
  Name of file that holds configuration information.

- **iAxis**  
  Axis on which to save information (AXISINDEX xxxx constants).

- **pCfg**  
  Pointer to AER_CFG_PACKET which holds the configuration information.

This function writes the specified configuration packet to the specified configuration file for the given axis. See Section 7.1.2.: Axis Configuration File Format for details on the layout of the file.

**See Also**

- AerConfig
- AerConfigReadPacket
CHAPTER 8: DATA CENTER FUNCTIONS

In This Section:
- Introduction ................................................. 8-1
- AerDCGetAxisDirectEx ............................. 8-2
- AerDCGetTaskDirect.................................. 8-3

8.1. Introduction

The Data Center functions are a convenient way of retrieving blocks of axis and task data.
8.2. AerDCGetAxisDirectEx

AERERR_CODE AerDCGetAxisDirectEx (HAERCTRL hAerCtrl, AXISMASK mAxis, PAER_AXIS_DATA_EX pData);

Parameters

- **hAerCtrl**
  - Handle to axis processor card.

- **mAxis**
  - Axis mask of axes to retrieve.

- **pData**
  - Pointer to an array of AER_AXIS_DATA_EX structures.

This function retrieves axis information for all axes specified in `mAxis`. The `pData` parameter must be allocated for the axes that need to be retrieved.

C Language and LabView Constants

AXISMASK_1 to AXISMASK_16
8.3. AerDCGetTaskDirect

AERERR_CODE AerDCGetTaskDirect (HAERCTRL hAerCtrl, TASKMASK mTask, PAER_TASK_DATA pData);

**Parameters**
- **hAerCtrl** Handle to axis processor card.
- **mTask** Task mask of task data to retrieve.
- **pData** Pointer to an array of AER_TASK_DATA structures.

This function retrieves axis information for all tasks specified in `mTask`. The `pData` parameter must be allocated for the axes that need to be retrieved.

**C Language and LabView Constants**

```
AXISMASK_1
```

to

```
AXISMASK_16
```

\[ \n\]
9.1. Introduction

The design of these routines is almost foolproof. There are very few cases that cause them to crash. A number of exceptions can occur during error string retrieval (i.e., a bad sprintf substitution). In these cases, the routine attempts to continue anyway (for a bad substitution it simply returns the string with no substitution). However, in some cases there is no rational continuation (pointer to buffer to receive string is NULL). In these cases, a message box appears explaining the problem and writes to the error log if it is open.
9.2. AERERRGETMESS

LPTSTR AERERRGETMESS (DWORD dwMessID, LPSTR pszStr, WORD wSeverity,
WORD wlastArg, BOOL bLongForm);

Parameters
---

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dwMessID</td>
<td>Error code to return message string for.</td>
</tr>
<tr>
<td>pszStr</td>
<td>Pointer to buffer to receive text corresponding to error code.</td>
</tr>
<tr>
<td>wSeverity</td>
<td>Severity of error message (see constants below).</td>
</tr>
<tr>
<td>wlastArg</td>
<td>Name of last fixed-argument in calling routines argument list.</td>
</tr>
<tr>
<td>bLongForm</td>
<td>If TRUE the &quot;long&quot; form of the message will be returned.</td>
</tr>
</tbody>
</table>

AERERRGETMESS is a macro that must be called if the user wants to call AerErrGetMessage or AerErrGetMessage from a function that receives a variable number of arguments. However, any function can call it.

If it cannot write to pszStr for any reason, it displays a message box indicating the problem, and the file and line in which the offending AERERRGETMESS call was made.

This macro has no return value and is similar to AerErrGetMessage.

C Language and LabView Constants

- AERERR_TYPE_MSG
- AERERR_TYPE_WARN
- AERERR_TYPE_ERROR
- AERERR_TYPE_NONE

VB Constants

- aerErrTypeMsg
- aerErrTypeWarn
- aerErrTypeError
- aerErrTypeNone

See Also

- AerErrGetMessage

Example

samples\lib\AexSys.C
9.3. AerErrGetMessage and AerErrGetMessageEx

LPTSTR AerErrGetMessage (DWORD dwMessID, LPTSTR pszStr, LONG lStrSize, 
BOOL bLongForm,...);

Declare Function AerErrGetMessageEx Lib “AERERR.DLL” (ByVal dwMessID As 
Long, ByVal pszStr As String, ByVal lStrSize As Long, ByVal 
bLongForm As Long) As String

Parameters

dwMessID Error code to retrieve string for.
pszStr Pointer to buffer to receive text corresponding to error code.
lStrSize Size of passed pszStr buffer.
bLongForm If TRUE, returns the "long" form of the message.
... Extra arguments (optional), to be substituted into the string (C 
language only).

All string variables in MS Visual Basic, passed by reference (ByRef), must be 
declared as fixed length strings within your program, long enough to hold the string 
value returned by the function. Also, those string variables which are passed, with 
another parameter indicating the length of the string variable, must also be fixed 
length strings, otherwise, you would not be able to pass the length of the string. For 
example, to declare a fixed length string:

```
DIM sGlobStr as STRING * 50 ; 50 characters long
```

These functions retrieve error strings when given an error code. The C functions take a 
variable number of arguments, and perform "sprintf-type" substitutions of any additional 
parameters passed (following the bLongForm argument) into the error strings. For 
example, if the message is, "Here it is: %s", and a string is passed after the 
bLongForm argument, then %s is replaced with the passed string. If the number or type of extra 
parameters passed does not match those in the string, then the substitution may not be 
performed, or "junk" values may appear within the substitution’s - it will not crash. If the 
user does not pass any extra parameters, no substitution occurs.

This function echoes the string to the error log, if an error log is open. If successful, it 
returns a pointer to the pszStr argument. If unsuccessful, it returns NULL. In most failures 
such as a message corresponding to the error code), it places a string describing the error 
in the passed pszStr. In the worst case, when it cannot write to pszStr it displays a message 
box describing the failure.

If bLongForm is TRUE, it will prefix the string with information indicating the passed 
severity, the error number, and the Aerotech name. If bLongForm is false, the user will 
only receive in the string the text of the message and the severity. However, if the error is 
AERERR_NOERR, (and bLongForm is FALSE), the user will not receive the severity.

See Also

AERERRGETMESS 
aerERRLogFileOpen

Example

Samples\Lib\AexSys.C 
Samples\Lib\VisualBasic\RunPgm.vbp
9.4. **AerErrGetSeverity**

LPTSTR AerErrGetSeverity (AERERR_CODE dwCode);

Declare Function AerErrGetSeverity Lib “AERERR.DLL” (ByVal dwCode As Long) As String

**Parameters**

dwCode Error code passed in.

*AerErrGetSeverity* returns the severity for an error code. See the AERERR>Type_xxxx constants in the appendices, for details on the types of severity.

**C Language and LabView Constants**

AERERR_TYPE_xxxx

**VB Constants**

aerErrType_xxxx

**See Also**

AerErrGetMessage

**Example**

samples\lib\AexSys.C
9.5. **AerErrLogError**

AERERR_CODE AerErrLogError (LPCTSTR pszStr);

Declare Function AerErrLogError Lib “AERERR.DLL” (ByVal pszStr As String) As Long

**Parameters**

- **pszStr**  
  The error string to write to the file.

*AerErrLogError* writes the given string to the error file. If successfully written to the file, the routine returns *AERERR_NOERR*. If it fails, it returns an appropriate error code.

| All calls to *AerErrGetMessage* echo the returned string into the log file. |

**C Language and LabView Constants**

- **AERERR_NOERR**

**VB Constants**

- **aerNoErr**

**See Also**

- *AerErrLogFileOpen*
- *AerErrLogFileClose*

**Example**

- `samples\lib\AexSys.C`
9.6. AerErrLogFileClose

void AerErrLogFileClose (void);

Declare Function AerErrLogFileClose Lib “AERERR.DLL” (void) As Void

Parameters

NONE

AerErrLogFileClose closes the error log file named "AERERR.LOG" in the current directory. There is no return code and if no log file is open, it does nothing.

See Also

AerErrLogFileOpen
AerErrLogError

Example

samples\lib\AexSys.C
9.7. AerErrLogFileOpen

LPTSTR AerErrLogFileOpen (void);

Declare Function AerErrLogFileOpen Lib "AERERR.DLL" (void) As String

Parameters
NONE

AerErrOpenLogFile opens up the log file named "AERERR.LOG" in the current directory, and activates error logging. All subsequent calls to AerErrGetMessage echo the message to that file. If the log file is opened successfully, the routine returns NULL. If it fails, it returns a pointer to the appropriate error string.

See Also
    AerErrLogFileClose
    AerErrLogError

Example
    samples\lib\AexSys.C
9.8. AerErrMessageBox

void AerErrMessageBox (DWORD dwMessID, WORD wSeverity, ...);

Declare Function AerErrMessageBox Lib “AERERR.DLL” (ByVal dwMessID As Long, ByVal wSeverity As Integer) As String

Parameters
   dwMessID       Error code passed in.
   wSeverity      Severity of error message (see constants below).
   ...            String substitution parameters, as required (C language only).

AerErrMessageBox retrieves an error string for the specified error code and displays a message box indicating the error. If WIN95 is not defined it will write the message string to the error log (if it is open). It requires a variable number of arguments, and performs "sprintf-type" substitutions of any additional parameters passed (past the wSeverity) into the error strings. Refer to AerErrGetMessage for details on formatting.

It is identical to AerErrGetMessage, except that instead of returning a pointer to the message, it displays the message in a message box. If errors are encountered while retrieving the error message or allocating space, then a message box describing the problem appears.

C Language and LabView Constants

   AERERR_TYPE_MSG
   AERERR_TYPE_WARN
   AERERR_TYPE_ERROR
   AERERR_TYPE_NONE

VB Constants

   aerErrTypeMsg
   aerErrTypeWarn
   aerErrTypeError
   aerErrTypeNone

See Also

   AerErrGetMessage
   AerErrLogFileOpen

Example

   samples\lib\AexSys.C

∇ ∇ ∇
CHAPTER 10: EVENT FUNCTIONS

In This Section:
- Introduction .......................................................... 10-1
- AerEventCloseEvent ........................................... 10-2
- AerEventCreateEvent ........................................... 10-3
- AerEventGenerateInt ............................................. 10-5
- AerEventTest ...................................................... 10-6

10.1. Introduction

Events are the method by which the UNIDEX 600 Series controllers communicate back to the application running on the host PC. This communication is done via ISA bus interrupts (for UNIDEX 600, the interrupt is defined in the registry and through jumpers on the card, refer to AerRegxxxx functions, see Chapter 19 for additional information).

The use of events requires the user to be able to use the Win32 calls for defining threads and blocking (waiting) on threads (see the ExEvent.c example).

When the axis processor generates the interrupt, the device driver receives the interrupt, determines its associated event, and the event that is set for that particular interrupt is “pulsed”. The state is changed from non-signaled to signaled to non-signaled. Therefore, any application thread that is “blocking” on this event is released.

When the axis processor wants to generate an interrupt, it writes to a specific location in its memory map, generates the PC interrupt, and continues processing. It does not wait to see if the device driver has received it. The device driver receives the interrupt via the PC bus and signals any events “blocking” for the given interrupt notifying the axis processor that it received the interrupt. However, if the axis processor generates another interrupt before the device driver acknowledges the first, the axis processor will delay the interrupt until the device driver does acknowledge the first. This “interrupt queuing” will not queue more than one interrupt of the same type, the later interrupts will be discarded. The types of interrupts include both axis and task numbers (i.e., if two faults are generated on axis 1 and one on axis 2, only the second axis 1 fault will be discarded).

Shown below are the four basic steps for creating an event.

1. Create event (AerEventCreateEvent)
2. Spawn a thread (Win32 function Create).
3. Block thread (Win32 function WaitForSingleObject [multiple]).
4. Close event.
10.2. **AerEventCloseEvent**

AERERR_CODE AerEventCloseEvent( HAERCTRL hAerCtrl, HANDLE hEvent, DWORD dwEvent, DWORD dwNum );

Declare Function AerEventCloseEvent Lib "AERSYS.DLL" ( ByVal hAerCtrl As Long, ByVal hEvent As Long, ByVal dwEvent As Long, ByVal dwNum As Long) As Long

**Parameters**
- `hAerCtrl` Handle to the axis processor card.
- `hEvent` Handle to event (obtained from an `AerEventCreateEvent` call).
- `dwEvent` AER_EVENT_xxxx constant.
- `dwNum` Specifies an axis or task number.

This function closes the event associated with the specified event handle. The `dwNum` argument is only used if the specified interrupt is an axis or task interrupt.

**C Language and LabView Constants**

```
AER_EVENT_XXXX
```

**VB Constants**

```
aerEventxxxx
```

**See Also**

`AerEventCreateEvent`
10.3. AerEventCreateEvent

AERERR_CODE AerEventCreateEvent (HAERCTRL hAerCtrl, DWORD dwEvent, DWORD dwNum, LPCTSTR pszEventName, PHANDLE phEvent);

Declare Function AerEventCreateEvent Lib "AERSYS.DLL" (ByVal hAerCtrl As Long, ByVal dwEvent As Long, ByVal dwNum As Long, ByVal pszEventName As String, ByRef phEvent) As Long

Parameters

- **hAerCtrl**: Handle to the axis processor card.
- **dwEvent**: AER_EVENT_xxxx constant (see constants below).
- **dwNum**: Axis or task number (see constants below).
- **pszEventName**: Name of event or Null.
- **phEvent**: Pointer to variable to hold a Win32 event handle.

The **dwNum** argument is only used if the specified interrupt is an axis or task interrupt. Table 10-1 shows the interrupts that can be generated by the axis processor.

**Table 10-1. Axis Processor Generated Interrupts**

<table>
<thead>
<tr>
<th>Type</th>
<th>AER_EVENT_xxxx code</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task Fault</td>
<td>TASK_FAULT</td>
<td>A task related fault has occurred (i.e. bad CNC line encountered).</td>
</tr>
<tr>
<td>Axis Fault</td>
<td>AXIS_FAULT</td>
<td>An axis related fault has occurred (i.e. position error).</td>
</tr>
<tr>
<td>Interrupt Failure</td>
<td>UNKNOWN_EVENT</td>
<td>Interrupt occurred, but no data found.</td>
</tr>
<tr>
<td>Task Callback</td>
<td>TASK_CALLBACK</td>
<td>A task callback has occurred.</td>
</tr>
<tr>
<td>Joystick</td>
<td>JOYSTICK</td>
<td>A joystick event has occurred.</td>
</tr>
</tbody>
</table>

The Unknown event interrupt occurs when a process other than the axis processor triggers an interrupt. The callback interrupt is a “miscellaneous” category for operations that the axis processor requires an operation from the host processor. A ‘task callback’ includes file operations (axis processor wants to read/write to a file), display operations (axis processor wants to display a Window under control of the users CNC program). The secondary loop interrupt allows the user to define a variable timer interrupt to the “front-end” application.

Some of the faults have additional data associated with them. It is guaranteed that when the interrupt is received, that the associated data exists. The way to access the additional data varies with the type of interrupt. When an axis or task interrupt is triggered, the user can check the FAULT parameter for the axis or task respectively, to determine the fault code. Interrupt failures have no additional data.

The user can attach multiple interrupts to the same event. Just use the same pszEventname and the interrupts will be added.

Multiple executables can also monitor the same interrupt. However, each executable must create its own event and use the same name to reference that event. Even if some of the processes creating the event close, the remaining events will be triggered by the interrupt.
The user may also reference an event at the Win32 level. Under Windows 95/NT the event is created via the Win32 function `CreateEvent`. A code fragment for the `AerEventCreateEvent` function follows:

```c
// Create a Win32 Manual-Reset Event that is Non-Signaled
hEvent = CreateEvent( NULL, TRUE, FALSE, pszEventName );
// pass handle on to device driver
....
// return event handle to user
*phEvent = hEvent;
```

In this case, if the above process closes, the other processes monitoring the event will not see the close.

**C Language and LabView Constants**

- `AER_EVENT_XXXX`
- `TASKINDEX_1` to `TASKINDEX_4`
- `AXISMASK_1` to `AXISMASK_16`

**VB Constants**

- `aerEventxxxx`
- `aerTaskIndex1` to `aerTaskIndex4`
- `aerAxisMask1` to `aerAxisMask16`

**See Also**

- `AerEventCloseEvent`
10.4. **AerEventGenerateInt**

```c
AERERR_CODE AerEventGenerateInt (HAERCTRL hAerCtrl, DWORD dwEvent, DWORD dwNum);
```

Declare Function AerEventGenerateInt Lib "AERSYS.DLL"(ByVal hAerCtrl As Long, ByVal dwEvent As Long, ByVal dwNum As Long) As Long

**Parameters**
- **hAerCtrl** Handle to the axis processor card.
- **dwEvent** AER_EVENT_xxxx constant (see constants below).
- **dwNum** Axis or task number.

This function will manually generate the specified interrupt on the axis processor. Normally, the axis processor would generate an interrupt at the proper time. The **dwNum** argument is only used if the specified interrupt is an axis or task interrupt. This function is intended for test purposes only.

**C Language and LabView Constants**

AER_EVENT_XXXX

**VB Constants**

aerEventxxxx

**See Also**

AerEventTest
10.5. AerEventTest

AERERR_CODE AerEventTest (HAERCTRL hAerCtrl, DWORD dwEvent, DWORD dwNum, DWORD dwWaitMSec);

Declare Function AerEventTest Lib "AERSYS.DLL"(ByVal hAerCtrl As Long, ByVal dwEvent As Long, ByVal dwNum As Long, ByVal dwWaitMSec As Long) As Long

Parameters

- **hAerCtrl** Handle to the axis processor card.
- **dwEvent** AER_EVENT_xxxx constant.
- **dwNum** Specifies an axis or task number.
- **dwWaitMSec** Time-out value to wait for interrupt.

This function is supplied as a means to test the event and interrupt operation. This function will create the desired event, generate an interrupt, start a separate thread of execution and block on that event for the specified time period or until the interrupt occurs, and close the event. If the interrupt is properly received, it returns AERERR_NOERR. Otherwise, it returns an error code indicating the problem. The *dwNum* argument is only used if the specified interrupt is an axis or task interrupt.

C Language and LabView Constants

- **AER_EVENT_XXXX**
- **AERERR_NOERR**

VB Constants

- **AerEventxxxx**
- **aerNoErr**

See Also

- **AerEventGenerateInt**
CHAPTER 11: MEMORY FUNCTIONS

In This Section:
- Introduction ................................................. 11-1
- AerMemCheck .............................................. 11-2
- AerMemGetFree ............................................ 11-3

11.1. Introduction

These functions return the total amount of memory on the axis processor card and the amount of free memory available.
11.2. AerMemCheck

AERERR_CODE AerMemCheck (HAERCTRL hAerCtrl, PDWORD pdwMemSize);

Declare Function AerMemCheck Lib "AERSYS.DLL" (ByVal hAerCtrl As Long, ByRef pdwMemSize As Long) As Long

Parameters

hAerCtrl Handle to the axis processor card.
pdwMemSize Pointer to returned size of memory.

This function returns the size of the installed memory on the axis processor card in the pdwMemSize pointer. The firmware will occupy some of this memory, see AerMemGetFree for the amount of free memory.

See Also

AerSysDownload
AerMemGetFree

Example

samples\lib\AexSys.C
11.3. AerMemGetFree

AERERR_CODE AerMemGetFree (HAERCTRL hAerCtrl, PDWORD pdwTotal, PDWORD pdwLargest)

Declare Function AerMemGetFree Lib "AERSYS.DLL" (ByVal hAerCtrl As Long, ByRef pdwTotal As Long, ByRef pdwLargest As Long) As Long

Parameters

- **hAerCtrl** Handle to the axis processor card.
- **pdwTotal** A pointer to the total free memory on the axis processor card.
- **pdwLargest** A pointer to the largest contiguous block of free memory.

This function returns the amount of free memory currently available on the Unidex 600 series axis card. The total free memory is returned by *pdwTotal* and the largest contiguous free memory block is returned by *pdwLargest*.

See Also

- AerMemCheck

Example

```
samples\lib\AexSys.C
```

∇  ∇  ∇
CHAPTER 12: MOVE FUNCTIONS

In This Section:
- Introduction ................................................................. 12-1
- AerJogGetMode ............................................................... 12-3
- AerJogSetMode ............................................................... 12-4
- AerMoveAbort, AerMoveMAbort ........................................ 12-5
- AerMoveAbsolute, AerMoveMAbsolute ............................. 12-6
- AerMoveAxis .................................................................... 12-7
- AerMoveFeedhold, AerMoveMFeedhold ............................. 12-9
- AerMoveFreerun, AerMoveMFreerun ................................. 12-10
- AerMoveHalt, AerMoveMHalt ........................................... 12-11
- AerMoveHome, AerMoveMHome ...................................... 12-12
- AerMoveHomeNoLimit, AerMoveMHomeNoLimit ............... 12-13
- AerMoveHomeQuick, AerMoveMHomeQuick ...................... 12-14
- AerMoveIncremental, AerMoveMIncremental .................... 12-17
- AerMoveInfeedSlave, AerMoveMInfeedSlave ...................... 12-18
- AerMoveLinear ................................................................ 12-19
- AerMoveMulti ................................................................... 12-20
- AerMoveOscillate, AerMoveMOscillate ............................. 12-22
- AerMoveQueueAbsolute, AerMoveMQueueAbsolute ............ 12-23
- AerMoveQueueFlush, AerMoveMQueueFlush ...................... 12-24
- AerMoveQueueHold .......................................................... 12-25
- AerMoveQueueIncremental, AerMoveMQueueIncremental .... 12-26
- AerMoveQueueRelease, AerMoveMQueueRelease ................ 12-27
- AerMoveRelease, AerMoveMRelease ................................. 12-28
- AerMoveWaitDone ............................................................ 12-29
- AerMoveWaitDoneMulti .................................................... 12-30

12.1. Introduction

The AerMove and AerJog library functions implement basic motion, jogging, and homing functions. Homing is the process of seeking an accurate, absolute reference point for a system that uses incremental position feedback. The various motion functions provide absolute, incremental, queued, linear, and axes freerun motion, similar to that provided by the asynchronous move command (STRM, MOVETO, HOME, etc.) and G1 CNC commands. See the U600MMI.hlp file for more information.

AerMove functions DO NOT wait until the motion is done; the function will return immediately after the motion is started. You must use the “AerMoveWaitDone” functions to wait until motion is actually completed.

Most of the move functions in this chapter allow the user to specify movement on a single axis, or a group of axes. Functions with an “AerMoveM” prefix accept an axis mask that specifies a set of axes to move. Functions with an “AerMove” prefix can only generate movement on the single axis number provided (except AerMoveLinear() which can move multiple axes at once).
The *AerMove* and *AerMoveM* functions are implemented as macros of *AerMoveAxis* and *AerMoveMulti* functions.

For the Visual Basic programmer, the *AerMoveAxis* and *AerMoveMulti* functions must be used directly.

All functions in this chapter require speeds with distances given in machine steps. Therefore, the programmer must do the necessary conversion from inches, millimeters, or degrees to machine counts. For this reason, it is recommended that these functions not be used, since the machine, global and task parameters have no effect, only the axis parameters do. See the *AerTaskxxxx* or *AerProgramxxxx* functions instead.
12.2. AerJogGetMode

AERERR_CODE AerJogGetMode (HAERCTRL hAerCtrl, AXISINDEX iAxis, PWORD pwMode, PDWORD pdwSpeed, PWORD pwEnableBit, PWORD pwDirBit);

Declare Function AerJogGetMode Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal iAxis As Long, ByRef pwMode As Integer, ByRef pdwSpeed As Long, ByRef pwEnableBit As Integer, ByRef pwDirBit As Integer) As Long

Parameters

- **hAerCtrl**: Handle to the axis processor card.
- **iAxis**: Axis index (see constants).
- **pwMode**: Pointer returning jog state, enabled/disabled (1/0) for the specified axis.
- **pdwSpeed**: Pointer returning velocity in machine steps per second.
- **pwEnableBit**: Pointer to the I/O bit number that will jog the axis.
- **pwDirBit**: Pointer to the I/O bit number controlling the direction of the axis.

This function returns the state of the jog mode that allows the axes to be jogged from an external device connected to digital I/O of the controller. The **iAxis** parameter specifies the axis number. The **pwEnableBit** and **pwDirBit** parameters return the I/O bit numbers that are used as the enable and direction inputs. These I/O numbers are virtual bit numbers for UNIDEX 600/620.

C Language and LabView Constants

<table>
<thead>
<tr>
<th><strong>AXISINDEX_1</strong> to <strong>AXISINDEX_16</strong></th>
</tr>
</thead>
</table>

VB Constants

<table>
<thead>
<tr>
<th><strong>aerAxisIndex1</strong> to <strong>aerAxisIndex16</strong></th>
</tr>
</thead>
</table>

See Also

- AerConfig
- AerJogSetMode

Example

Samples\Lib\AexMove.C
12.3. AerJogSetMode

AERERR_CODE AerJogSetMode (HAERCTRL hAerCtrl, AXISINDEX iAxis, WORD wMode, DWORD dwSpeed, WORD wEnableBit, WORD wDirBit);

Declare Function AerJogSetMode Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal iAxis As Long, ByVal wMode As Integer, ByVal dwSpeed As Long, ByVal wEnableBit As Integer, ByVal wDirBit As Integer) As Long

Parameters

- **hAerCtrl**: Handle to the axis processor card.
- **iAxis**: Axis index (see constants below).
- **wMode**: Enable/disable (1/0) jog mode for the specified axis.
- **dwSpeed**: Velocity in machine steps per second.
- **wEnableBit**: I/O bit number that will jog the axis.
- **wDirBit**: I/O bit number that controls the direction of the axis.

This function allows the axes to be jogged from an external device connected to the user inputs of the controller. The axis may not be in the sync mode or a programming error will occur. The *iAxis* parameter specifies the axis. The I/O bit numbers representing the *wEnableBit* and *wDirBit* parameters, are specified as virtual bit numbers for UNIDEX 600/620. A logic level one on the direction bit will result in positive (CW) motor rotation.

**C Language and LabView Constants**

- AXISINDEX_1 to AXISINDEX_16

**VB Constants**

- aerAxisIndex1 to aerAxisIndex16

See Also

- AerConfig
- AerJogGetMode

Example

- Samples\Lib\AexMove.C
12.4. AerMoveAbort, AerMoveMAbort

AERERR_CODE AerMoveAbort (HAERCTRL hAerCtrl, AXISINDEX iAxis);

The AerMoveMAbort function is identical to AerMoveAbort, except that it operates over a set of axes, rather than just one axis.

AERERR_CODE AerMoveMAbort (HAERCTRL hAerCtrl, AXISMASK mAxes);

For Visual Basic programming, see AerMoveAxis and AerMoveMulti.

Parameters

- *hAerCtrl* Handle to the axis processor card.
- *iAxis* Axis index (AXISINDEX_xxxx constant).
- *mAxes* Axis mask (combination of AXISMASK_xxxx constants).

This function will command the specified axis to come to an abrupt stop by setting the commanded position equal to the current position. The commanded deceleration will be instantaneous. However, the actual deceleration will be some finite value, based on the inertia of the system and therefore, in reality, there will be some position overshoot.

See Also

- AerMoveAxis
- AerMoveHalt

Example

- Samples\Lib\AexMove.C
12.5. AerMoveAbsolute, AerMoveMAbsolute

AERERR_CODE AerMoveAbsolute (HAERCTRL hAerCtrl, AXISINDEX iAxis, LONG lTarg, DWORD dwSpeed);

AerMoveMAbsolute is identical to AerMoveAbsolute, except that it operates over a set of axes, rather than just one axis.

AERERR_CODE AerMoveMAbsolute (HAERCTRL hAerCtrl, AXISMASK mAxes, PLONG plMoveArray, PDWORD pdwSpeedArray);

For Visual Basic programming, see AerMoveAxis and AerMoveMulti.

Parameters

- *hAerCtrl* Handle to the axis processor card.
- *iAxis* Axis index (AXISINDEX_xxxx constant).
- *mAxes* Axis mask (combination of AXISMASK_xxxx constants).
- *lTarg* Absolute position (+/-) in machine steps to move to.
- *dwSpeed* Speed in machine steps per second.
- *plMoveArray* Array of move values specified in machine steps.
- *pdwSpeedArray* Array of speeds in machine steps.

This function will move the specified axis to the desired absolute position at the given velocity. Target and speed are in machine units. The axis will accelerate and decelerate using the current accel/decel axis parameters (ACCEL, DECEL, etc.). Any motion command currently executing on the specified axis/axes will be aborted and the axis will immediately begin moving to the specified absolute position. This will allow the destination to be changed during the move, calling this function while an absolute move from a previous function call is in progress. If motion is begun on a different axis, the current motion will not be affected. The specified drive must be enabled and the axis must not be in the sync mode or a programming error will occur.

With the AerMoveMAbsolute function, unpredictable results occur if the number of elements in either array is less than the number of bits set in mAxes. The specified drives must be enabled and the axes must not be in the sync mode or a programming error will occur.

See Also

- AerMoveAbsolute
- AerMoveAxis
- AerConfig
- AerMoveMulti
- AerMoveIncremental
- AerMoveQueueAbsolute
- AerMoveQueueIncremental

Example

Samples\Lib\AexMove.C
12.6. AerMoveAxis

AERERR_CODE AerMoveAxis (HAERCTRL hAerCtrl, AXISINDEX iAxis, DWORD dwMoveCmd, LONG lDistOrTarget, DWORD dwSpeed);

Declare Function AerMoveAxis Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal iAxis As Long, ByVal dwMoveCmd As Long, ByVal lDistOrTarget As Long, ByVal dwSpeed As Long) As Long

Parameters

hAerCtrl Handle to the axis processor card.
iAxis Index of axis to move (see constants below).
dwMoveCmd Type of move command (see constants below).
lDistOrTarget Depends on the dwMoveCmd. - Distances/targets are specified in machine steps. Direction is either (+/-) 1 for positive/negative direction.
dwSpeed Speed is specified in machine steps per second.

The AerMoveAxis function implements basic asynchronous motion and homing functions. Homing is the process of seeking an accurate absolute reference point for a system that uses incremental position feedback. The various motion functions provide absolute, incremental, queued, linear, and axes freerun motion. Most of the AerMove functions are implemented as macros of AerMoveAxis or AerMoveMulti. See Table 12-1. This function will not wait until motion is complete (see AerMoveWaitDone() for that).

Table 12-1. AerMove Functions

<table>
<thead>
<tr>
<th>MoveCmd (VB see below)</th>
<th>Macro</th>
<th>Parameter</th>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>AERMOVE_ABSOLUTE</td>
<td>AerMoveAbsolute</td>
<td>Target</td>
<td>Speed</td>
</tr>
<tr>
<td>AERMOVE_HOME</td>
<td>AerMoveHome</td>
<td>Direction</td>
<td>Speed</td>
</tr>
<tr>
<td>AERMOVE_INCREMENTAL</td>
<td>AerMoveIncremental</td>
<td>Distance</td>
<td>Speed</td>
</tr>
<tr>
<td>AERMOVE_FREERUN</td>
<td>AerMoveFreerun</td>
<td>Direction</td>
<td>Speed</td>
</tr>
<tr>
<td>AERMOVE_INFEEDSLAVE</td>
<td>AerMoveInfeedSlave</td>
<td>Distance</td>
<td>Speed</td>
</tr>
<tr>
<td>AERMOVE_QINCREMENTAL</td>
<td>AerMoveQueue-Incremental</td>
<td>Distance</td>
<td>Speed</td>
</tr>
<tr>
<td>AERMOVE_QABSOLUTE</td>
<td>AerMoveQueueAbsolute</td>
<td>Target</td>
<td>Speed</td>
</tr>
<tr>
<td>AERMOVE_HOMEQUICK</td>
<td>AerMoveHomeQuick</td>
<td>Direction</td>
<td>Speed</td>
</tr>
<tr>
<td>AERMOVE_HOMENOLIMIT</td>
<td>AerMoveHomeNoLimit</td>
<td>Direction</td>
<td>Speed</td>
</tr>
<tr>
<td>AERMOVE_OSCILLATE</td>
<td>AerMoveOscillate</td>
<td>Distance</td>
<td>Speed</td>
</tr>
<tr>
<td>AERMOVE_HALT</td>
<td>AerMoveHalt</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>AERMOVE_ABORT</td>
<td>AerMoveAbort</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>AERMOVE_FEEDHOLD</td>
<td>AerMoveFeedHold</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>AERMOVE_RELEASE</td>
<td>AerMoveRelease</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>AERMOVE_QFLUSH</td>
<td>AerMoveQueueFlush</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>AERMOVE_QHOLD</td>
<td>AerMoveQueueHold</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>AERMOVE_QRELEASE</td>
<td>AerMoveQueueRelease</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>AERMOVE_LINEAR</td>
<td>see AerMoveLinear</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
C Language and LabView Constants

AXISINDEX_1
to
AXISINDEX_16

AERMOVE_XXXX

VB Constants

aerAxisIndex1
to
aerAxisIndex16

aerMovexxxx

See Also

AerConfig
AerMoveMulti
AerMoveWaitDone

Example

Samples\Lib\AexMove.C
12.7. AerMoveFeedhold, AerMoveMFeedhold

The AerMoveMFeedhold function is identical to AerMoveFeedhold, except that it operates over a set of axes, rather than just one axis.

AERERR_CODE AerMoveMFeedhold (HAERCTRL hAerCtrl, AXISMASK mAxes);

For Visual Basic programming, see AerMoveAxis and AerMoveMulti.

Parameters

- hAerCtrl Handle to the axis processor card.
- iAxis Axis index (AXISINDEX_xxxx constant).
- mAxes Axis mask (combination of AXISMASK_xxxx constants).

This function will feedhold the motion in progress on the specified axis by decelerating the axis to a stop in the current mode (linear/sinusoidal, rate/time). The current mode is determined by the DECELMODE axis parameter. The DECRATE and DECEL axis parameters determine the current deceleration rate or time, respectively. To restart the move that was in progress, the AerMoveRelease function should be called in order to accelerate the axis back up to the programmed velocity based upon the axis acceleration parameters. This function will cause any motion commands issued to the specified axis before the function is called to be placed into a queue that is one level deep. After the feedhold is released, the last commanded move will be executed and any others ignored.

See Also
- AerMoveAxis
- AerMoveRelease

Example

Samples\Lib\AexMove.C
12.8. AerMoveFreerun, AerMoveMFreerun

AERERR_CODE AerMoveFreerun (HAERCTRL hAerCtrl, AXISINDEX iAxis, LONG lDir, DWORD dwSpeed);

AerMoveMFreerun is identical to AerMoveFreerun, except that it operates over a set of axes, rather than just one axis.

AERERR_CODE AerMoveMFreerun (HAERCTRL hAerCtrl, AXISMASK mAxes, PLONG plMoveArray, PDWORD pdwSpeedArray);

For Visual Basic programming, see AerMoveAxis and AerMoveMulti.

Parameters

- *hAerCtrl*: Handle to the axis processor card.
- *iAxis*: Axis index (AXISINDEX_xxxx constant).
- *mAxes*: Axis mask. (combination of AXISMASK_xxxx constants).
- *lDir*: Direction (+/- = 1/0) to move.
- *dwSpeed*: Speed in machine steps per second.
- *plMoveArray*: Array of direction values.
- *pdwSpeedArray*: Array of speeds in machine steps.

This function will start the specified axis moving in a specified direction at the specified velocity. The function is typically used to start continuous motion such as a spindle, where only velocity and direction are of significance. The velocity is specified in machine units per second. The axis will accelerate and decelerate using the current accel/decel axis parameters (ACCEL, DECEL, etc.). The specified drive must be enabled and the axis must not be in the sync mode or a programming error will occur. If distance and velocity are zero, and the axis is already executing an asynchronous move, the axis will stop.

With the AerMoveMFreerun function, unpredictable results will be obtained if the number of elements in either array is less than the number of bits set in mAxes. The specified drives must be enabled and the axes must not be in the sync mode or a programming error will occur.

See Also

- AerMoveAxis
- AerConfig
- AerMoveHalt
- AerMoveInfeedSlave
- AerMoveMultiple

Example

Samples\Lib\AexMove.C
12.9. AerMoveHalt, AerMoveMHalt

AERERR_CODE AerMoveHalt (HAERCTRL hAerCtrl, AXSINDEX iAxis);

The AerMoveMHalt function is identical to AerMoveHalt, except that it operates over a set of axes, rather than just one axis.

AERERR_CODE AerMoveMHalt (HAERCTRL hAerCtrl, AXISMASK mAxes);

For Visual Basic programming, see AerMoveAxis and AerMoveMulti.

Parameters

- **hAerCtrl** Handle to the axis processor card.
- **iAxis** Axis index (AXISINDEX_xxxx constant).
- **mAxes** Axis mask (combination of AXISMASK_xxxx constants).

This function will decelerate the specified axis to zero velocity using the current deceleration mode (linear/sinusoidal). The deceleration mode is dependent upon the setting of the DECELMODE axis parameter. The deceleration rate is defined by the DECELRATE axis parameter and the deceleration time is set by the DECEL axis parameter.

See Also

- AerMoveAxis
- AerMoveAbort

Example

See Samples\Lib\AexMove.C
12.10. AerMoveHome, AerMoveMHome

AERERR_CODE AerMoveHome (HAERCTRL hAerCtrl, AXISINDEX iAxis, LONG lDir, DWORD dwSpeed);

The AerMoveMHome function is identical to AerMoveHome, except that it operates over a set of axes, rather than just one axis.

AERERR_CODE AerMoveMHome (HAERCTRL hAerCtrl, AXISMASK mAxes, PLONG plMoveArray, PDWORD pdwSpeedArray);

For Visual Basic programming, see AerMoveAxis and AerMoveMulti.

Parameters

- hAerCtrl: Handle to the axis processor card.
- iAxis: Axis index (AXISINDEX_xxxx constant).
- mAxes: Axis mask (combination of AXISMASK_xxxx constants).
- lDir: Direction (+/- = 1/0) to home.
- dwSpeed: Home speed in machine steps per second.
- plMoveArray: Array of direction values.
- pdwSpeedArray: Array of speeds in machine steps.

This function performs the “HomeType=0” homing procedure (see the HomeType machine parameter). Refer to the U600 Series User’s Guide, P/N EDU157 for more information, or the U600MMI.hlp file.

The specified drive must be enabled and the axis must not be in the sync mode or a programming error will occur.

For AerMoveMHome, unpredictable results will be obtained if the number of elements in the either array is less than the number of bits set in mAxes. The specified drives must be enabled and the axes must not be in the sync mode or a programming error will occur.

See Also

- AerMoveAxis
- AerConfig
- AerMoveHomeNoLimit
- AerMoveHomeQuick
- AerMoveHomeRev
- AerMoveMHomeNoLimit
- AerMoveMHomeQuick
- AerMoveMHomeRev

Example

Samples\Lib\AexMove.C
12.11. AerMoveHomeNoLimit, AerMoveMHomeNoLimit

AERERR_CODE AerMoveHomeNoLimit (HAERCTRL hAerCtrl, AXISINDEX iAxis,
    LONG lDir, DWORD dwSpeed);

The AerMoveMHomeNoLimit function is identical to AerMoveHomeNoLimit, except that it operates over a set of axes, rather than just one axis.

AERERR_CODE AerMoveMHomeNoLimit (HAERCTRL hAerCtrl, AXISMASK mAxes,
    PLONG plMoveArray, PDWORD pdwSpeedArray);

For Visual Basic programming, see AerMoveAxis and AerMoveMulti.

Parameters

  hAerCtrl     Handle to the axis processor card.
  iAxis        Axis index (AXISINDEX_xxxx constant).
  mAxes        Axis mask. (combination of AXISMASK_xxxx constants).
  lDir         Direction (+/- = 1/0) to move.
  dwSpeed      Speed in machine steps per second to move at.
  plMoveArray  Array of direction values.
  pdwSpeedArray Array of speeds in machine steps.

This function performs the “HomeType=2” homing procedure (see the HomeType machine parameter). Refer to the U600 Series User’s Guide, P/N EDU157 for more information, or the U600MMI.hlp file.

The specified drive must be enabled and the axis must not be in the sync mode or a programming error will occur.

With the AerMoveMHomeNoLimit function, unpredictable results will be obtained if the number of elements in either array is less than the number of bits set in mAxes. The specified drives must be enabled and the axes must not be in the sync mode or a programming error will occur.

See Also

  AerMoveAxis
  AerConfig
  AerMoveHome
  AerMoveHomeQuick
  AerMoveHomeRev
  AerMoveMHome
  AerMoveMHomeQuick
  AerMoveMHomeRev

Example

  Samples\Lib\AexMove.C
12.12. AerMoveHomeQuick, AerMoveMHomeQuick

AERERR_CODE AerMoveHomeQuick (HAERCTRL hAerCtrl, AXISINDEX iAxis,
LONG lDir, DWORD dwSpeed);

AerMoveMHomeQuick is identical to AerMoveHomeQuick, except that it operates over a
set of axes, rather than just one axis.

AERERR_CODE AerMoveMHomeQuick (HAERCTRL hAerCtrl, AXISMASK mAxes,
PLONG plMoveArray, PDWORD pdwSpeedArray);

For Visual Basic programming, see AerMoveAxis and AerMoveMulti.

Parameters

- **hAerCtrl** Handle to the axis processor card.
- **iAxis** Axis index (AXISINDEX_xxxx constant).
- **mAxes** Axis mask. (combination of AXISMASK_xxxx constants).
- **lDir** Direction (+/- = 1/0) to move.
- **dwSpeed** Speed in machine steps per second to move at.
- **plMoveArray** Array of direction values.
- **pdwSpeedArray** Array of speeds in machine steps.

This function performs the “HomeType=3” homing procedure (see the HomeType
machine parameter). Refer to the U600 Series User’s Guide, P/N EDU157 for more
information, or the U600MMI.hlp file.

The velocity is specified in machine units per second. The axis will accelerate and
decelerate at the currently selected modes (linear/sinusoidal) and rates/times. The
specified drive must be enabled and the axis must not be in the sync mode or a
programming error will occur.

For the AerMoveMHomeQuick function, unpredictable results will be obtained if the
number of elements in either array is less than the number of bits set in mAxes. The
specified drives must be enabled and the axes must not be in the sync mode or a
programming error will occur.

See Also

- AerMoveAxis
- AerConfig
- AerMoveHome
- AerMoveHomeNoLimit
- AerMoveHomeRev
- AerMoveMHome
- AerMoveMHomeNoLimit
- AerMoveMHomeRev

Example

Samples\Lib\AexMove.C

AERERR_CODE AerMoveHomeRev (HAERCTRL hAerCtrl, AXISINDEX iAxis, LONG lDir, DWORD dwSpeed);

`AerMoveHomeRev` is identical to `AerMoveHomeRev`, except that it operates over a set of axes, rather than just one axis.

AERERR_CODE AerMoveMHomeRev (HAERCTRL hAerCtrl, AXISMASK mAxes, PLONG plMoveArray, PDWORD pdwSpeedArray);

`AerMoveHomeAlt` and `HomeMoveMHomeAlt` have different names but duplicate the functionality of `AerMoveHomeRev` and `AerMoveMHomeRev`.

AERERR_CODE AerMoveHomeAlt (HAERCTRL hAerCtrl, AXISINDEX iAxis, PLONG lDir, PDWORD dwSpeed);

AERERR_CODE AerMoveMHomeAlt (HAERCTRL hAerCtrl, AXISMASK mAxes, PLONG plMoveArray, PDWORD pdwSpeedArray);

For Visual Basic programming, see `AerMoveAxis` and `AerMoveMulti`.

**Parameters**

- **hAerCtrl**  Handle to the axis processor card.
- **iAxis**  Axis index (see constants below).
- **mAxes**  Axis mask (see constants below).
- **lDir**  Direction (+/- = 1/0) to move.
- **dwSpeed**  Speed in machine steps per second to move at.
- **plMoveArray**  Array of direction values (+/- = 1/0) to move.
- **pdwSpeedArray**  Array of speeds in machine steps.

These functions perform the “HomeType=1” homing procedure (see the `HomeType` machine parameter). Refer to the *U600 Series User’s Guide, P/N EDU157* for more information, or the U600MMI.hlp file.

The velocity is specified in machine units per second. The axis will accelerate and decelerate at the currently selected modes (linear/sinusoidal) and rates/times. The specified drive must be enabled and the axis must not be in the sync mode or a programming error will occur.

For the `AerMoveMHomeRev` and `AerMoveMHomeAlt` functions, unpredictable results will be obtained if the number of elements in either the `plMoveArray` or the `pdwSpeedArray` parameter is less than the number of bits set in `mAxes`. The specified drives must be enabled and the axes must not be in the sync mode or a programming error will occur.
C Language and LabView Constants
AXISMASK_1
to
AXISMASK_16
AXISINDEX_1
to
AXISINDEX_16

VB Constants
aerAxisMask1
to
aerAxisMask16
aerAxisIndex1
to
aerAxisIndex16

See Also
AerMoveAxis
AerConfig
AerMoveHome, AerMoveMHome
AerMoveHomeNoLimit, AerMoveMHomeNoLimit
AerMoveHomeQuick, AerMoveMHomeQuick

Example
Samples\Lib\AexMove.C

AERERR_CODE AerMoveIncremental (HAERCTRL hAerCtrl, AXISINDEX iAxis, 
LONG lLen, DWORD dwSpeed);

AerMoveMIncremental is identical to AerMoveIncremental, except that it operates over a 
set of axes, rather than just one axis.

AERERR_CODE AerMoveMIncremental (HAERCTRL hAerCtrl, AXISMASK mAxes, 
PLONG plMoveArray, PDWORD pdwSpeedArray);

For Visual Basic programming, see AerMoveAxis and AerMoveMulti.

Parameters

- **hAerCtrl**: Handle to the axis processor card.
- **iAxis**: Axis index (AXISINDEX_xxxx constant).
- **mAxes**: Axis mask. (combination of AXISMASK_xxxx constants).
- **lLen**: Length (+/-) of move in machine steps.
- **dwSpeed**: Speed in machine steps per second to move at.
- **plMoveArray**: Array of move values specified in machine steps.
- **pdwSpeedArray**: Array of speeds in machine steps.

This function will start a specified axis moving in a specified direction at a specified 
velocity. The specified move increment will override any currently executing move, 
beginning a new incremental move from the current position at which this function was 
called (not recommended). If motion is begun on a different axis, the current motion will 
not be affected. A move currently executing will immediately take on the new velocity. 
The velocity is specified in machine units per second. The axis will accelerate and 
decelerate based on the current accel/decel axis parameters (ACCEL, DECEL, etc.). The 
specified drive must be enabled and the axis must not be in the sync mode or a 
programming error will occur.

For the AerMoveIncremental function, unpredictable results will be obtained if the 
number of elements in either array is less than the number of bits set in mAxes. The 
specified drives must be enabled and the axes must not be in the sync mode or a 
programming error will occur.

See Also

- AerMoveAxis
- AerConfig
- AerMoveMAbsolute

Example

Samples\Lib\AexMove.C
12.15. AerMoveInfeedSlave, AerMoveMInfeedSlave

AERERR_CODE AerMoveInfeedSlave (HAERCTRL hAerCtrl, AXISINDEX iAxis, 
LONG lDist, DWORD dwSpeed);

The AerMoveMInfeedSlave function is identical to AerMoveInfeedSlave, except that it 
operates over a set of axes, rather than just one axis.

AERERR_CODE AerMoveMInfeedSlave (HAERCTRL hAerCtrl, AXISMASK mAxes, 
PLONG plMoveAry, PDWORD pdwSpeedAry);

For Visual Basic programming, see AerMoveAxis and AerMoveMulti.

Parameters

- **hAerCtrl**: Handle to the axis processor card.
- **iAxis**: Axis index (AXISINDEX_xxxx constant).
- **mAxes**: Axis mask (combination of AXISMASK_xxxx constants).
- **lDist**: Distance( +/- ) to move to in machine steps.
- **dwSpeed**: Speed in machine steps per second to move at.
- **plMoveAry**: Array of move values specified in machine steps.
- **pdwSpeedAry**: Array of speed values specified in machine steps.

This function will start a specified axis moving a specified distance at a specified 
velocity. A move currently executing will have its move increment summed with the 
distance specified in this function. The velocity is specified in machine units per second. 
The axis will accelerate and decelerate based on the current accel/decel axis parameters 
(ACCEL, DECEL, etc.). The specified drive must be enabled and the axis must be in the 
sync mode or a programming error will occur.

This function should be used only on axes that are in sync or cam table mode. The 
motion generated by this command will be added to the motion output by the cam table.

See Also

- AerConfig
- AerCam...

Example

Samples\Lib\AexMove.C
12.16. AerMoveLinear

AERERR_CODE AerMoveLinear (HAERCTRL hAerCtrl, AXISMASK mAxes, PLONG plTargetArray, DWORD dwSpeed);

Declare Function AerMoveLinear Lib "AERSYS.DLL" (ByVal hAerCtrl As Long, ByVal mAxes As Double, ByRef plTargetArray As Long, ByVal dwSPeed As Long) As Long

Parameters

- **hAerCtrl** Handle to the axis processor card.
- **mAxes** Axis mask. (see constants below).
- **plTargetArray** Pointer to an array of axes move distances.
- **dwSpeed** Vectorial machine steps per second velocity.

This function is used to start the linear coordinated motion of the specified axes. Coordinated motion is identical to a G1 CNC motion (see the U600MMI.hlp help file for details). AerMoveLinear(), like all other AerMove() functions, however, will return immediately, not waiting for the motion to complete (G1 waits for motion to complete). See the AerMoveWaitDone function to wait for completion. The *mAxes* parameter specifies the axes to move by setting their respective bit true with the first axis represented by bit 0. Each axis specified within the *mAxes* bitmask will start and stop at the same time using the current acceleration and deceleration modes and rates. The vector direction and speed are specified in machine units. The distances specified in the *plTargetArray* are ‘packed’ (i.e. an AxisMask of 0x09 - AXISMASK_1 | AXISMASK_4 - would have the target distance for Axis 1 in *plTargetArray*[0] and Axis 4 would be specified in *plTargetArray*[1]). The specified drives must be enabled or a programming error will occur.

C Language and LabView Constants

- **AXISMASK_1**
- **AXISMASK_16**

VB Constants

- **aerAxisMask1**
- **aerAxisMask16**

See Also

- AerConfig
- AerMoveWaitDone

Example

- Samples\Lib\AexMove.C
12.17. AerMoveMulti

AERERR_CODE AerMoveMulti (HAERCTRL hAerCtrl, AXISMASK mAxis, DWORD dwMoveCmd, PLONG plMoveArray, PDWORD pdwSpeedArray);

Declare Function AerMoveMulti Lib “AERSYS.DLL” (ByVal hAerCtrl As Long ByVal mAxis As Double, ByVal dwMoveCmd As Long, ByRef plMoveArray As Long, ByRef pdwSpeedArray As Long) As Long

Parameters

hAerCtrl Handle to the axis processor card.
mAxis Mask of axis to move (see constants below).
dwMoveCmd Type of move command (see constants below).
plMoveArray Array of Move-commands. Depends on the dwMoveCmd. Distances/targets are specified in machine steps. Direction is either (+/-) 1 for positive/negative direction.
pdwSpeedArray Array of speeds in machine steps.

The AerMoveMulti function implements the basic asynchronous motion commands for multiple axes. It is a direct counterpart to AerMoveAxis. The input arrays must have one element per axis. See Table 12-2. This function will not wait until motion is complete (see AerMoveWaitDoneMulti() for that).

Table 12-2. AerMoveMulti Functions

<table>
<thead>
<tr>
<th>MoveCmd (VB, see below)</th>
<th>Macro</th>
<th>plMoveArray Parameter</th>
<th>pdwSpeedArray Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>AERMOVE_ABORT</td>
<td>AerMoveMAbort</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>AERMOVE_ABSOLUTE</td>
<td>AerMoveMAbsolute</td>
<td>Target</td>
<td>Speed</td>
</tr>
<tr>
<td>AERMOVE_FEEDHOLD</td>
<td>AerMoveMFeedHold</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>AERMOVE_FREE_RUN</td>
<td>AerMoveMFreeRun</td>
<td>Direction</td>
<td>Speed</td>
</tr>
<tr>
<td>AERMOVE_HALT</td>
<td>AerMoveMHalt</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>AERMOVE_HOME</td>
<td>AerMoveMHome</td>
<td>Direction</td>
<td>Speed</td>
</tr>
<tr>
<td>AERMOVE_HOME_ALT_REV</td>
<td>AerMoveMAlt</td>
<td>Direction</td>
<td>Speed</td>
</tr>
<tr>
<td>AERMOVE_HOME_NO_LIMIT</td>
<td>AerMoveMNoLimit</td>
<td>Direction</td>
<td>Speed</td>
</tr>
<tr>
<td>AERMOVE_HOME_QUICK</td>
<td>AerMoveMQuick</td>
<td>Direction</td>
<td>Speed</td>
</tr>
<tr>
<td>AERMOVE_INCREMENTAL</td>
<td>AerMoveMIncremental</td>
<td>Distance</td>
<td>Speed</td>
</tr>
<tr>
<td>AERMOVE_INFEED_SLAVE</td>
<td>AerMoveMInfeedSlave</td>
<td>Distance</td>
<td>Speed</td>
</tr>
<tr>
<td>AERMOVE_LINEAR</td>
<td>see AerMoveLinear</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AERMOVE_OSCILLATE</td>
<td>AerMoveMOscillate</td>
<td>Distance</td>
<td>Speed</td>
</tr>
<tr>
<td>AERMOVE_Q_ABSOLUTE</td>
<td>AerMoveMQueueAbsolute</td>
<td>Target</td>
<td>Speed</td>
</tr>
<tr>
<td>AERMOVE_Q_INCREMENTAL</td>
<td>AerMoveMQueueIncremental</td>
<td>Distance</td>
<td>Speed</td>
</tr>
<tr>
<td>AERMOVE_Q_FLUSH</td>
<td>AerMoveMQueueFlush</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>AERMOVE_Q_HOLD</td>
<td>AerMoveMQueueHold</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>AERMOVE_Q_RELEASE</td>
<td>AerMoveMQueueRelease</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>AERMOVE_RELEASE</td>
<td>AerMoveMRelease</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>
C Language and LabView Constants
  AXISMASK_1
to
  AXISMASK_16

  AERMOVE_XXXX

VB Constants
  aerAxisMask1
to
  aerAxisMask16

  aerMovexxxx

See Also
  AerConfig
  AerMoveMulti
  AerMoveWaitDoneMulti

Example
  Samples\Lib\AexMove.C
12.18. AerMoveOscillate, AerMoveMOscillate

AERERR_CODE AerMoveOscillate (HAERCTRL hAerCtrl, AXISINDEX iAxis, LONG lDistance, DWORD dwSpeed);

The AerMoveMOscillate function is identical to AerMoveOscillate, except that it operates over a set of axes, rather than just one axis.

AERERR_CODE AerMoveMOscillate (HAERCTRL hAerCtrl, AXISMASK mAxes, PLONG plMoveAry, PDWORD pdwSpeedAry);

For Visual Basic programming, see AerMoveAxis and AerMoveMulti.

Parameters

- **hAerCtrl**: Handle to the axis processor card.
- **iAxis**: Axis index (AXISINDEX_xxxx constant).
- **mAxes**: Axis mask (combination of AXISMASK_xxxx constants).
- **lDistance**: The distance of the moves (sign is initial direction).
- **dwSpeed**: The speed at which the axis will move (specified in user units/min). Refer to “v” in Figure 12-1.
- **plMoveAry**: Array of move values specified in machine steps.
- **pdwSpeedAry**: Array of speed values specified in machine steps.

![Graph of Position and Speed](image)

This asynchronous motion command causes the specified axis to oscillate (cycle) the specified relative distance at the specified speed. The sign (+ or -) of the distance determines the initial direction of the move and is in user units. The speed is in user units per minute. The arithmetic sign of the speed parameter is ignored. The controller chooses whatever sign of the speed is appropriate to achieve a specified distance. To halt the axis, specify a zero feedrate or distance in subsequent uses of this command, or execute an ENDM CNC command on that axis.

Each move of the oscillation is a separate INDEX move. An AerMoveOscillate is equivalent to:

```c
While FOREVER
  AerMoveIndex (positive direction)
  AerMoveIndex (negative direction)
EndWhile
```

Each “leg” (INDEX in the example above) of the oscillate command will have its own acceleration and deceleration, as determined by the acceleration/deceleration axis parameters (ACCEL, DECEL, etc.). Thus the shape of the oscillation profile is effected by the ACCEL and DECEL parameters. In Figure 12-1 is a velocity profile given a linear ACCEL, but a sinusoidal DECEL.
12.19. AerMoveQueueAbsolute, AerMoveMQueueAbsolute

AERERR_CODE AerMoveQueueAbsolute (HAERCTRL hAerCtrl, AXISINDEX iAxis, LONG lTarg, DWORD dwSpeed);

The AerMoveMQueueAbsolute function is identical to AerMoveQueueAbsolute, except that it operates over a set of axes, rather than just one axis.

AERERR_CODE AerMoveMQueueAbsolute (HAERCTRL hAerCtrl, AXISMASK mAxes, PLONG plMoveAry, PDWORD pdwSpeedAry);

For Visual Basic programming, see AerMoveAxis and AerMoveMulti.

Parameters

- `hAerCtrl` Handle to the axis processor card.
- `iAxis` Axis index (AXISINDEX_xxxx constant).
- `mAxes` Axis mask (combination of AXISMASK_xxxx constants).
- `lTarg` Absolute position (+/-) to move to in machine steps.
- `dwSpeed` Speed in machine steps per second to move at.
- `plMoveAry` Array of move values specified in machine steps.
- `pdwSpeedAry` Array of speed values specified in machine steps.

This function is a queued version of the AerMoveAbsolute function. There is a 16 level queue for each axis. If a move is in progress, the queued move will not begin to execute until the current move is completed. If the queue is empty the move will begin immediately. The specified `iAxis` will move to the `lTarg` absolute position at the specified `dwSpeed`. The velocity is specified in machine units per second. The axis will accelerate and decelerate at the currently selected modes (linear/sinusoidal) and rates/times. The specified drive must be enabled and the axis must not be in the sync mode or a programming error will occur.

See Also

- AerConfig
- AerMoveAbsolute
- AerMoveIncremental
- AerMoveQueueIncremental

Example

Samples\Lib\AexMove.C
12.20. AerMoveQueueFlush, AerMoveMQueueFlush

AERERR_CODE AerMoveQueueFlush (HAERCTRL hAerCtrl, AXISINDEX iAxis);

The AerMoveMQueueFlush function is identical to AerMoveQueueFlush, except that it operates over a set of axes, rather than just one axis.

AERERR_CODE AerMoveMQueueFlush (HAERCTRL hAerCtrl, AXISMASK mAxes);

For Visual Basic programming, see AerMoveAxis and AerMoveMulti.

Parameters
- hAerCtrl Handle to the axis processor card.
- iAxis Axis index (AXISINDEX_xxxx constant).
- mAxes Axis mask (combination of AXISMASK_xxxx constants).

This function will clear the 16 level deep axis queue for the specified axis. All commands that were in the queue will not be executed.

See Also
- AerMoveQueueHold
- AerMoveQueueRelease

Example

Samples\Lib\AexMove.C
12.21. AerMoveQueueHold

AERERR_CODE AerMoveQueueHold (HAERCTRL hAerCtrl, AXISINDEX iAxis);

The AerMoveQueueHold function is identical to AerMoveQueueHold, except that it operates over a set of axes, rather than just one axis.

AERERR_CODE AerMoveMQueueHold (HAERCTRL hAerCtrl, AXISMASK mAxes);

For Visual Basic programming, see AerMoveAxis and AerMoveMulti.

Parameters

  hAerCtrl   Handle to the axis processor card.
  iAxis      Axis index (AXISINDEX_xxxx constant).
  mAxes      Axis mask (combination of AXISMASK_xxxx constants).

This function places the queue for the specified axis into the hold state upon completion of the current motion command. The current command will be completed and the queue can be restarted by using the AerMoveQueueRelease function.

See Also

  AerMoveQueueRelease

Example

  Samples\Lib\AexMove.C
12.22. AerMoveQueueIncremental, AerMoveMQueueIncremental

AERERR_CODE AerMoveQueueIncremental (HAERCTRL hAerCtrl, AXISINDEX iAxis, LONG lLen, DWORD dwSpeed);

The AerMoveMQueueIncremental function is identical to AerMoveQueueIncremental, except that it operates over a set of axes, rather than just one axis.

AERERR_CODE AerMoveMQueueIncremental (HAERCTRL hAerCtrl, AXISMASK mAxes, PLONG plMoveAry, PDWORD pdwSpeedAry);

For Visual Basic programming, see AerMoveAxis and AerMoveMulti.

Parameters

- **hAerCtrl** Handle to the axis processor card.
- **iAxis** Axis number for the specified motion (AXISINDEX_XXXX constants).
- **mAxes** Axis mask (combination of AXISMASK_XXXX constants).
- **lLen** Length ( +/- ) of move in machine steps.
- **dwSpeed** Speed in machine steps per second to move at.
- **plMoveAry** Array of move values specified in machine steps.
- **pdwSpeedAry** Array of speed values, specified in machine steps.

This function is a queued version of the AerMoveIncremental function. There is a 16 level queue for each axis. If a move is in progress, the queued move will not begin to execute until the current move is complete. If the queue is empty the move will begin immediately. The specified **iAxis** will move the specified **lLen** increment at the specified **dwSpeed**. The velocity is specified in machine units per second. The axis will accelerate and decelerate at the currently selected modes (linear/sinusoidal) and rates/times. The specified drive must be enabled and the axis must not be in the sync mode or a programming error will occur.

See Also

- AerConfig
- AerMoveAbsolute
- AerMoveIncremental
- AerMoveQueueAbsolute

Example

Samples\Lib\AexMove.C
12.23. AerMoveQueueRelease, AerMoveMQueueRelease

AERERR_CODE AerMoveQueueRelease (HAERCTRL hAerCtrl, AXISINDEX iAxis);

The AerMoveMQueueRelease function is identical to AerMoveQueueRelease, except that it operates over a set of axes, rather than just one axis.

AERERR_CODE AerMoveMQueueRelease (HAERCTRL hAerCtrl, AXISMASK mAxes);

For Visual Basic programming, see AerMoveAxis and AerMoveMulti.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hAerCtrl</td>
<td>Handle to the axis processor card.</td>
</tr>
<tr>
<td>iAxis</td>
<td>Axis index (AXISINDEX xxxx constant).</td>
</tr>
<tr>
<td>mAxes</td>
<td>Axis mask (combination of AXISMASK xxxx constants)</td>
</tr>
</tbody>
</table>

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

See Also

AerMoveQueueHold

Example

Samples\Lib\AexMove.C

AERERR_CODE AerMoveRelease (HAERCTRL hAerCtrl, AXISINDEX iAxis);

The AerMoveMRelease function is identical to AerMoveRelease, except that it operates over a set of axes, rather than just one axis.

AERERR_CODE AerMoveMRelease (HAERCTRL hAerCtrl, AXISMASK mAxes);

For Visual Basic programming, see AerMoveAxis and AerMoveMulti.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hAerCtrl</td>
<td>Handle to the axis processor card.</td>
</tr>
<tr>
<td>iAxis</td>
<td>Axis index (AXISINDEX_xxxx constant).</td>
</tr>
<tr>
<td>mAxes</td>
<td>Axis mask (combination of AXISMASK_xxxx constants).</td>
</tr>
</tbody>
</table>

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

See Also

AerMoveFeedhold

Example

Samples\Lib\AexMove.C
12.25. AerMoveWaitDone

AERERR_CODE AerMoveWaitDone (HAERCTRL hAerCtrl, AXISINDEX iAxis, DWORD dwTimeOutMsec, LONG flags);

Declare Function AerMoveWaitDone Lib "AERSYS.DLL" (ByVal hAerCtrl As Long, ByVal iAxis As Double, ByVal dwTimeOutMsec As Long, ByVal flags As Long) As Long

Parameters

- hAerCtrl Handle to the axis processor card.
- iAxis Index of axis to move (see constants below).
- dwTimeOutMsec Timeout value (10 millisecond units).
- flags 1 waits for “in-position bit” (STATUS axis parameter) to turn on.
          0 waits for “done bit” (STATUS axis parameter) to turn on.

This function waits until motion is complete on the given axis or an axis fault occurs on that axis. If the passed timeout value is exceeded before motion is done, it returns the error: AER960_RET_TIMEOUT (532,486 decimal or 0x82006). Pass dwTimeOutMsec as 0 to wait forever. This function sleeps for 10 msec, after each check. Therefore, dwTimeOutMsec is in units of 10 milliseconds (i.e., dwTimeOutMsec = 1, indicates a timeout value of 10 milliseconds).

C Language and LabView Constants

AXISINDEX_1 to AXISINDEX_16

VB Constants

aerAxisIndex1 to aerAxisIndex16

See Also

AerMoveAxis

Example

Samples\Lib\AexMove.C
12.26. AerMoveWaitDoneMulti

AERERR_CODE AerMoveWaitDoneMulti (HAERCTRL hAerCtrl, AXISMASK mAxis, DWORD dwTimeOutMsec, LONG flags);

Declare Function AerMoveWaitDoneMulti Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal mAxis As Double, ByVal dwTimeOutMsec As Long, ByVal flags As Long) As Long

Parameters

- **hAerCtrl**: Handle to the axis processor card.
- **mAxis**: Mask of axes to move (see constants below).
- **dwTimeOutMsec**: Timeout value (10 millisecond units).
- **flags**: 1 waits for “in-position bit” (STATUS axis parameter) to turn on. 0 waits for “done bit” (STATUS axis parameter) to turn on.

This function waits until motion is complete on all of the given axes or an axis fault occurs. If the passed timeout value is exceeded before motion is done, it returns the error: AER960_RET_TIMEOUT (532,486 decimal 0x82006). Pass dwTimeOutMsec as 0 to wait forever. This function sleeps for 10 msec, after each check. Therefore, dwTimeOutMsec is in units of 10 milliseconds (i.e., dwTimeOutMsec = 1, indicates a timeout value of 10 milliseconds).

### C Language and LabView Constants

- **AXISMASK_1**
- **to**
- **AXISMASK_16**

### VB Constants

- **aerAxisMask1**
- **to**
- **aerAxisMask16**

### See Also

- AerMoveAxis

### Example

- Samples\Lib\AexMove.C

∇  ∇  ∇
CHAPTER 13: PARAMETER FUNCTIONS

13.1. Introduction

The AerParm functions perform the reading and writing of parameter values, determining the number of parameters in each parameter group, and provide textual as well as min/max settings for each parameter. The parameters are divided into four groups, refer to Table 13-1.

Table 13-1. AerParm Groups (see constants below)

<table>
<thead>
<tr>
<th>Group</th>
<th>Scope</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Axis</td>
<td>Relate only to the given axis</td>
<td>Integer</td>
</tr>
<tr>
<td>Machine</td>
<td>Relate only to the given axis</td>
<td>Floating point</td>
</tr>
<tr>
<td>Task</td>
<td>Relate only to the given task</td>
<td>Floating point</td>
</tr>
<tr>
<td>Global</td>
<td>Relate to the entire controller</td>
<td>Floating point</td>
</tr>
</tbody>
</table>

These functions refer to the parameters by number. The base define for the parameter groups is given below, where XXXX is the parameter name. For example, the KP axis parameter is referred to by the constant AXISPARM_KP (C Language) or aerAxisParmKP (VB).

Some task and global parameters listed above as floating point values have a valid range for integers only, but are returned in a double data type (floating point). However, if some parameters in a group are floating point, then for generality they must all be floating point. For example, the task parameter TASKPARM_NumTaskDoubles is clearly an integer, although it is returned and set by a double data type variable.

The “get” and “set” functions download and upload from the controller respectively. The “download” functions automatically read ASCII text .INI files on disk that contains parameter assignments, and downloads these to the controller. The “read” and “write” functions read and write, respectively, ASCII text on parameters to disk .INI files. In functions that interact with disk files (“read,” “write,” “download”), if the file name is passed as NULL, it retrieves the filename from the “project file” INI\U600.INI.

C Language and LabView Constants

<table>
<thead>
<tr>
<th>Constants</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>AXISPARM_XXXX</td>
<td>aerAxisParmXXXX</td>
</tr>
<tr>
<td>MACHPARM_XXXX</td>
<td>aerMachParmXXXX</td>
</tr>
<tr>
<td>TASKPARM_XXXX</td>
<td>aerTaskParmXXXX</td>
</tr>
<tr>
<td>GLOBPARM_XXXX</td>
<td>aerAxisparmXXXX</td>
</tr>
</tbody>
</table>

VB Constants

<table>
<thead>
<tr>
<th>Constants</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>aerAxisParmXXXX</td>
<td>aerAxisParmXXXX</td>
</tr>
<tr>
<td>aerMachParmXXXX</td>
<td>aerMachParmXXXX</td>
</tr>
<tr>
<td>aerTaskParmXXXX</td>
<td>aerTaskParmXXXX</td>
</tr>
<tr>
<td>aerAxisparmXXXX</td>
<td>aerAxisparmXXXX</td>
</tr>
</tbody>
</table>
13.2. AerParmAxisDownloadFile

AERERR_CODE AerParmAxisDownloadFile (HAERCTRL hAerCtrl, LPCTSTR pszFile, AXISMASK mAxis);

Declare Function AerParmAxisDownloadFile Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal pszFile, ByVal mAxis) As Long

Parameters
- hAerCtrl Handle to axis processor card.
- pszFile File to download (can be Null or an Empty String).
- mAxis Mask of the Axis to download parameters for (see constants below).

The AerParmAxisDownloadFile function downloads the axis parameters to the controller, for the specified axes. If the file name is passed as NULL, it retrieves the filename from the “project file” INIU600.INI. The default filename currently in use by the user can be retrieved with the AerRegGetFileName function.

C Language and LabView Constants
- AXISMASK_1 to AXISMASK_16

VB Constants
- aerAxisMask1 to aerAxisMask16

See Also
- AerRegGetFileName
13.3. AerParmAxisGetCount

AERERR_CODE AerParmAxisGetCount (HAERCTRL hAerCtrl, PDWORD pdwCount);

Declare Function AerParmAxisGetCount Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByRef pdwCount As Long) As Long

Parameters

- **hAerCtrl** Handle to the axis processor card.
- **pdwCount** Pointer to double word to receive the number of axis parameters.

This function returns the number of axis parameters.

See Also

- AerParmAxisGetValue

Example

Samples\Lib\AexSys.C
13.4. AerParmAxisGetInfo

AERERR_CODE AerParmAxisGetInfo (HAERCTRL hAerCtrl, DWORD dwParm, PAER_PARM_INFO pInfo);

AERERR_CODE AerParmAxisGetInfoEx (HAERCTRL hAerCtrl, DWORD dwParm, LPTSTR pszName, PDDOUBLE pdMin, PDDOUBLE pdMax, PWORD pwAttr);

AERERR_CODE AerParmAxisGetInfoEx2 (HAERCTRL hAerCtrl, DWORD dwParm, LPTSTR pszName, PDDOUBLE pdMin, PDDOUBLE pdMax, PWORD pwAttr, PDDOUBLE pdDefault, PDWORD pdwDisplaySubGroup);

Declare Function AerParmAxisGetInfoEx Lib "AERSYS.DLL" (ByVal hAerCtrl As Long, ByVal dwParm As Long, ByVal pszName As String, ByRef pdMin As Double, ByRef pdMax As Double, ByRef pwAttr As Integer) As Long

Declare Function AerParmAxisGetInfoEx2 Lib "AERSYS.DLL" (ByVal hAerCtrl As Long, ByVal dwParm As Long, ByVal pszName As String, ByRef pdMin As Double, ByRef pdMax As Double, ByRef pwAttr As Integer, ByRef pdDefault as Double, ByRef pdwDisplaySubGroup As Long) As Long

Parameters

- **hAerCtrl**: Handle to the axis processor card.
- **dwParm**: Parameter number, (see constants below).
- **pInfo**: Pointer to structure to receive the parameter information (C Language Only, see AER_PARM_INFO in Appendix C: Structures).
- **pszName**: Pointer to buffer to hold name of parameter.
- **pdMin**: Pointer to hold Minimum value of parameter.
- **pdMax**: Pointer to hold Maximum value of parameter.
- **pwAttr**: Pointer to hold parameter attribute (see constants below).
- **pdDefault**: Pointer to hold the Default value of parameter
- **pdwDisplaySubGroup**: Pointer to hold the subgroup to display the parameter in the parameter editor.

All string variables in MS Visual Basic, passed by reference (ByRef), must be declared as fixed length strings within your program, long enough to hold the string value returned by the function. Also, those string variables which are passed, with another parameter indicating the length of the string variable, must also be fixed length strings, otherwise, you would not be able to pass the length of the string. For example, to declare a fixed length string:

```
DIM sGlobStr as STRING * 50 ; 50 characters long
```
This function returns data concerning an axis parameter. If the parameter number you provide is not a valid number, the function will still return with no error code. In this case, the “wAttr” mask of the pInfo structure will not have the VALID bit set. See example for details.

C Language and LabView Constants

AXISPARM_XXXX

AXIS_ATTR_XXXX

VB Constants

aerAxisParmXXXX

aerAxisAttrXXXX

Example

Samples\Lib\AexSys.C
13.5. AerParmAxisGetValue

AERERR_CODE AerParmAxisGetValue (HAERCTRL hAerCtrl, AXISINDEX iAxis, DWORD dwParm, PDWORD pdwValue);

Declare Function AerParmAxisGetValue Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal iAxis As Long, ByVal dwParm As Long, ByRef pdwValue As Long ) As Long

Parameters
- **hAerCtrl** Handle to the axis processor card.
- **iAxis** Physical axis index, (see constants below).
- **dwParm** Parameter number, (see constants below).
- **pdwValue** Pointer to double word to receive the current parameter value.

This function returns the value of an axis parameter, from the controller.

C Language and LabView Constants

**AXISINDEX_1** to **AXISINDEX_16**

**AXISPARM_XXXX**

VB Constants
- **aerAxisIndex1** to **aerAxisIndex16**
- **aerAxisParmXXXX**

See Also
- **AerParmAxisSetValue**

Example
- Samples\Lib\AexSys.C
- Samples\Lib\VisualBasic\RunPgm.vbp
13.6. AerParmAxisReadValue

AERERR_CODE AerParmAxisReadValue (LPCTSTR pszFile, AXISINDEX iAxis,
    LPCTSTR pszName, PDWORD pdwValue, PDWORD pdwDefValue);

Declare Function AerParmAxisReadValue Lib “AERSYS.DLL” (ByVal pszFile As
    String, ByVal iAxis As Long, ByVal pszName As String, ByRef pdwValue As Long, ByRef pdwDefValue As Long) As Long

Parameters

- **pszFile**: Pointer to name of axis parameter file (can be NULL or empty string).
- **iAxis**: Which axis to retrieve from file (see constants below).
- **pszName**: Name of axis parameter to read.
- **pdwValue**: Pointer to hold value read from file.
- **pdwDefValue**: Pointer to value holding the default value (may be NULL).

All string variables in MS Visual Basic, passed by reference (ByRef), must be declared as fixed length strings within your program, long enough to hold the string value returned by the function. Also, those string variables which are passed, with another parameter indicating the length of the string variable, must also be fixed length strings, otherwise, you would not be able to pass the length of the string. For example, to declare a fixed length string:

```vb
DIM sGlobStr as STRING * 50 ; 50 characters long
```

This function retrieves the value of the specified axis parameter from its .INI file. If the file name is passed as NULL, it retrieves the filename from the “project file” INIU600.INI. If a default value is specified, then this value is returned as the value if the parameter could not be found in the file.

**C Language and LabView Constants**

- `AXISINDEX_1` to `AXISINDEX_16`

**VB Constants**

- `aerAxisIndex1` to `aerAxisIndex16`

**See Also**

- `AerRegGetFileName`
13.7. AerParmAxisSetValue

AERERR_CODE AerParmAxisSetValue (HAERCTRL hAerCtrl, AXISINDEX iAxis, DWORD dwParm, DWORD dwValue);

Declare Function AerParmAxisSetValue Lib "AERSYS.DLL" (ByVal hAerCtrl As Long, ByVal iAxis As Long, ByVal dwParm As Long, ByVal dwValue As Long) As Long

Parameters
- **hAerCtrl** Handle to the axis processor card.
- **iAxis** Physical axis index (see constants below).
- **dwParm** Parameter number, (see constants below).
- **dwValue** New value for the parameter.

AerParmAxisSetValue sets an axis parameter value on the controller. A programming error will occur if there is a problem setting the parameter. In addition, some parameters are read-only. Use AerParmAxisGetInfo to find out the minimum, maximum, and other characteristics of the parameter.

Parameters do have minimum and maximum allowed values.

C Language and LabView Constants

AXISINDEX_1 to AXISINDEX_16

AXISPARM_XXXX

VB Constants

aerAxisIndex1 to aerAxisIndex16

aerAxisParmXXXX

See Also

AerParmAxisGetValue
AerParmAxisGetInfo

Example

Samples\Lib\AexSys.C
13.8. AerParmAxisWriteValue

AERERR_CODE AerParmAxisWriteValue (LPCTSTR pszFile, AXISINDEX iAxis, LPCTSTR pszName, DWORD dwValue);

Declare Function AerParmAxisWriteValue Lib “AERSYS.DLL” (ByVal pszFile As String, ByVal iAxis As Long, ByVal pszName As String, ByVal dwValue As Long); As Long

Parameters

- **pszFile**: Name of Axis Parameter file (can be Null or Empty String).
- **iAxis**: Which axis to retrieve from file (see constants below).
- **pszName**: Name of axis parameter to write.
- **dwValue**: Value to write to file.

This function writes the value of the specified axis parameter to its .INI file. If the file name is passed as NULL, it retrieves the filename from the “project file” INI\U600.INI.

C Language and LabView Constants

- `AXISINDEX_1` to `AXISINDEX_16`

VB Constants

- `aerAxisIndex1` to `aerAxisIndex16`

See Also

- `AerRegGetFileName`
13.9. AerParmGlobalDownloadFile

AERERR_CODE AerParmGlobalDownloadFile (HAERCTRL hAerCtrl, LPCTSTR pszFile);

Declare Function AerParmGlobalDownloadFile Lib "AERSYS.DLL" (ByVal hAerCtrl As Long, ByVal pszFile As String) As Long

Parameters

- hAerCtrl Handle to axis processor card.
- pszFile File to download (can be Null or Empty String).

This function downloads the global parameter file to the motion control card. If the file name is passed as NULL, it retrieves the filename from the "project file" INI/U600.INI. The default filename currently in use by the user can be retrieved with the AerRegGetFileName function.

See Also

- AerRegGetFileName
13.10. AerParmGlobalGetCount

AERERR_CODE AerParmGlobalGetCount (HAERCTRL hAerCtrl, PDWORD pdwCount);

Declare Function AerParmGlobalGetCount Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByRef pdwCount As Long) As Long

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hAerCtrl</td>
<td>Handle to the axis processor card.</td>
</tr>
<tr>
<td>pdwCount</td>
<td>Pointer to double word to receive the number of global parameters.</td>
</tr>
</tbody>
</table>

Returns the number of global parameters.

See Also

AerParmGlobalGetValue

Example

Samples\Lib\AexSys.C
13.11. AerParmGlobalGetInfo

AERERR_CODE AerParmGlobalGetInfo (HAERCTRL hAerCtrl, DWORD dwParm, PAER_PARM_INFO pInfo);

AERERR_CODE AerParmGlobalGetInfoEx (HAERCTRL hAerCtrl, DWORD dwParm, LPTSTR pszName, PDOUBLE pdMin, PDOUBLE pdMax, PWORD pwAttr);

AERERR_CODE AerParmGlobalGetInfoEx2 (HAERCTRL hAerCtrl, DWORD dwParm, LPTSTR pszName, PDOUBLE pdMin, PDOUBLE pdMax, PWORD pwAttr, PDOUBLE pdDefault, PDWORD pdwDisplaySubGroup);

Declare Function AerParmGlobalGetInfoEx Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal dwParm As Long, ByVal pszName As String, ByRef pdMin As Double, ByRef pdMax As Double, ByRef pwAttr As Integer ) As Long;

Declare Function AerParmGlobalGetInfoEx2 Lib “AERSYS.DLL” (ByVal hAerCrctl As Long, ByVal dwParm As Long, ByVal pszName As String, ByRef pdMin As Double, ByRef pdMax As Double, ByRef pwAttr As Integer, ByRef pdDefault as Double, ByRef pdwDisplaySubGroup As Long)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hAerCtrl</td>
<td>Handle to the axis processor card.</td>
</tr>
<tr>
<td>dwParm</td>
<td>Parameter number, (see constants below).</td>
</tr>
<tr>
<td>pInfo</td>
<td>Pointer to structure to receive the parameter information (C Language Only, refer to AER_PARM_INFO in Appendix C: Structures).</td>
</tr>
<tr>
<td>pszName</td>
<td>Pointer to buffer to hold name of parameter.</td>
</tr>
<tr>
<td>pdMin</td>
<td>Pointer to hold minimum value of parameter.</td>
</tr>
<tr>
<td>pdMax</td>
<td>Pointer to hold maximum value of parameter.</td>
</tr>
<tr>
<td>pwAttr</td>
<td>Pointer to hold parameter attribute (see constants below).</td>
</tr>
<tr>
<td>pdDefault</td>
<td>Pointer to hold the Default value of parameter</td>
</tr>
<tr>
<td>pdwDisplaySubGroup</td>
<td>Pointer to hold the subgroup to display the parameter in the parameter editor.</td>
</tr>
</tbody>
</table>

All string variables in MS Visual Basic, passed by reference (ByRef), must be declared as fixed length strings within your program, long enough to hold the string value returned by the function. Also, those string variables which are passed, with another parameter indicating the length of the string variable, must also be fixed length strings, otherwise, you would not be able to pass the length of the string. For example, to declare a fixed length string:

```
DIM sGlobStr as STRING * 50 ; 50 characters long
```
This function returns data concerning a global parameter. If the parameter number provided by the user is not a valid number, the function will still return with no error code. In this case, the “wAttr” mask of the pInfo structure will not have the VALID bit set.

C Language and LabView Constants

    GLOBPARM_XXXX

    PARM_ATTR_XXXX

VB Constants

    aerGlobParmXXXX

    aerParmAttrXXXX

Example

    Samples\Lib\AexSys.C
13.12. AerParmGlobalGetValue

AERERR_CODE AerParmGlobalGetValue (HAERCTRL hAerCtrl, DWORD dwParm, PDOUBLE pfdValue);

Declare Function AerParmGlobalGetValue Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal iAxis As Long, ByVal dwParm As Long, ByVal pfdValue As Double) As Long

Parameters
- hAerCtrl  Handle to the axis processor card.
- dwParm   Parameter number, (see constants below).
- pfdValue Pointer to double floating point to receive the current parameter value.

Returns the value of a global parameter from the controller.

C Language and LabView Constants

GLOBPARM_XXXX

VB Constants

aerGlobParmXXXX

See Also

AerParmGlobalSetValue

Example

Samples\Lib\AexSys.C
13.13. **AerParmGlobalReadValue**

AERERR_CODE AerParmGlobalReadValue (LPCTSTR pszFile, LPCTSTR pszName, PDOUBLE pdValue, PDOUBLE pdDefValue);

Declare Function AerParmGlobalReadValue Lib "AERSYS.DLL" (ByVal pszFile As String, ByVal pszName As String, ByRef pdValue As Double, ByRef pdDefValue As Double) As Long

**Parameters**

- **pszFile**
  Name of global parameter file (can be Null or Empty String).
- **pszName**
  Name of global parameter to read.
- **pdValue**
  Pointer to hold value read from file.
- **pdDefValue**
  Pointer to value holding the default value (may be NULL).

This function retrieves the value of the specified global parameter from its .INI file. If the file name is passed as NULL, it retrieves the filename from the “project file” INIU600.INI. If a default value is specified, then this value is returned as the value if the parameter could not be found in the file.

**See Also**

*AerRegGetFileName*
13.14. AerParmGlobalSetValue

AERERR_CODE AerParmGlobalSetValue( HAERCTRL hAerCtrl, DWORD dwParm, DOUBLE fdValue );

Declare Function AerParmGlobalSetValue Lib “AERSYS.DLL” ( ByVal hAerCtrl As Long, ByVal iAxis As Long, ByVal dwParm As Long, ByVal fdValue As Double) As Long

Parameters

\( hAerCtrl \) Handle to the axis processor card.
\( dwParm \) Parameter number, (see the constants below).
\( fdValue \) New value for the parameter.

AerParmGlobalSetValue sets a global parameter value on the controller. An error code will be returned if there is a problem modifying the parameter. Also, some parameters are read-only. Use AerParmGlobalGetInfo to find out the minimum, maximum, and other characteristics of the parameter.

Parameters do have minimum and maximum allowed values.

C Language and LabView Constants

GLOBPARM_XXXX

VB Constants

aerGlobParmXXXX

See Also

AerParmGlobalGetValue
AerParmGlobalGetInfo

Example

Samples\Lib\AexSys.C
13.15. AerParmGlobalWriteValue

AERERR_CODE AerParmGlobalWriteValue( LPCTSTR pszFile, LPCTSTR pszName, DOUBLE dValue );

Declare Function AerParmGlobalWriteValue Lib “AERSYS.DLL” ( ByVal pszFile As String, ByVal pszName As String, ByVal dValue As Double) As Long

Parameters

pszFile Name of global parameter file (can be Null or Empty String).
pszName Name of global parameter to write.
dValue Value to write to file.

This function writes the value of the specified task parameter to its .INI file. If the file name is passed as NULL, it retrieves the filename from the “project file” INIU600.INI.

See Also

AerRegGetFileName
13.16. AerParmMachineDownloadFile

AERERR_CODE AerParmMachineDownloadFile ( HAERCTRL hAerCtrl, LPCTSTR pszFile, AXISMASK mAxis );

AERERR_CODE AerParmMachineDownloadFile Lib “AERSYS.DLL” ( ByVal hAerCtrl As Long, ByVal pszFile As String, ByVal mAxis As Long) As Long

Parameters

\begin{itemize}
\item \texttt{hAerCtrl} Handle to axis processor card.
\item \texttt{pszFile} File to download (can be Null or Empty String).
\item \texttt{mAxis} Mask of the axis to download parameters for (see constants below).
\end{itemize}

This function downloads the machine parameters to the controller for the specified axes. If the file name is passed as NULL, it retrieves the filename from the “project file” INIU600.INI. The default filename currently in use by the user can be retrieved with the \textit{AerRegGetFileName} function.

C Language and LabView Constants

\begin{itemize}
\item AXISMASK_1 to AXISMASK_16
\end{itemize}

VB Constants

\begin{itemize}
\item aerAxisMask1 to aerAxisMask16
\end{itemize}

See Also

\begin{itemize}
\item AerRegGetFileName
\end{itemize}
13.17. AerParmMachineGetCount

AERERR_CODE AerParmMachineGetCount( HAERCTRL hAerCtrl, PDWORD pdwCount );

Declare Function AerParmMachineGetCount Lib "AERSYS.DLL" ( ByVal hAerCtrl As Long, ByRef pdwCount As Long ) As Long

Parameters
   hAerCtrl          Handle to the axis processor card.
   pdwCount          Pointer to double word to receive the number of machine parameters.

Returns the number of machine parameters.

See Also
   AerParmMachineGetValue

Example
   Samples\Lib\AexSys.C
### 13.18. AerParmMachineGetInfo

AERERR_CODE AerParmMachineGetInfo (HAERCTRL hAerCtrl, DWORD dwParm, PAER_PARM_INFO pInfo);

AERERR_CODE AerParmMachineGetInfoEx (HAERCTRL hAerCtrl, DWORD dwParm, LPTSTR pszName, PDOUBLE pdMin, PDOUBLE pdMax, PWORD pwAttr);

AERERR_CODE AerParmMachineGetInfoEx2 (HAERCTRL hAerCtrl, DWORD dwParm, LPTSTR pszName, PDOUBLE pdMin, PDOUBLE pdMax, PWORD pwAttr, PDOUBLE pdDefault, PWORD pdwDisplaySubGroup);

Declare Function AerParmMachineGetInfoEx Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal dwParm As Long, ByVal pszName As String, ByRef pdMin As Double, ByRef pdMax As Double, ByRef pwAttr As Integer) As Long;

Declare Function AerParmMachineGetInfoEx2 Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal dwParm As Long, ByVal pszName As String, ByRef pdMin As Double, ByRef pdMax As Double, ByRef pwAttr As Integer, ByRef pdDefault As Double, ByRef pdwDisplaySubGroup As Long) As Long

**Parameters**

- **hAerCtrl**
  Handle to the axis processor card.

- **dwParm**
  Parameter number, (see constants below).

- **pInfo**
  Pointer to structure to receive the parameter information (C Language Only, refer to AER_PARM_INFO in Appendix C: Structures).

- **pszName**
  Pointer to buffer to hold name of parameter.

- **pdMin**
  Pointer to hold minimum value of parameter.

- **pdMax**
  Pointer to hold maximum value of parameter.

- **pwAttr**
  Pointer to hold parameter attribute (see constants below).

- **pdDefault**
  Pointer to hold the Default value of parameter.

- **pdwDisplaySubGroup**
  Pointer to hold the subgroup to display the parameter in the parameter editor.

All string variables in MS Visual Basic, passed by reference (ByRef), must be declared as fixed length strings within your program, long enough to hold the string value returned by the function. Also, those string variables which are passed, with another parameter indicating the length of the string variable, must also be fixed length strings, otherwise, you would not be able to pass the length of the string. For example, to declare a fixed length string:

```vbnet
DIM sGlobStr as STRING * 50   ' 50 characters long
```
This function returns data concerning a machine parameter. If the parameter number provided by the user is not a valid number, the function will still return with no error code. But in this case, the “wAttr” mask of the pInfo structure will not have the VALID bit set.

C Language and LabView Constants

MACHPARM_XXXX

PARM_ATTR_XXXX

VB Constants

AerMachParmXXXX

aerParmAttrXXXX

Example

Samples\Lib\AexSys.C
13.19. **AerParmMachineGetValue**

AERERR_CODE AerParmMachineGetValue (HAERCTRL hAerCtrl, AXISINDEX iAxis, DWORD dwParm, PDOUBLE pfdValue);

Declare Function AerParmMachineGetValue Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal iAxis As Long, ByVal dwParm As Long, ByVal dwParm As Long, ByRef pfdValue As Double) As Long

**Parameters**
- **hAerCtrl** Handle to the axis processor card.
- **iAxis** Physical axis index (see constants below).
- **dwParm** Parameter number, (see constants below).
- **pfdValue** Pointer to double word to receive the current parameter value.

Returns a machine parameter value from the controller.

**C Language and LabView Constants**
- **AXISINDEX_1** to **AXISINDEX_16**
- **MACHPARM_XXXX**

**VB Constants**
- **aerAxisIndex1** to **aerAxisIndex16**
- **aerMachParmXXXX**

**See Also**
- AerParmMachineSetValue

**Example**
- Samples\Lib\AexSys.C
- Samples\Lib\VisualBasic\RunPgm.vbp
13.20. **AerParmMachineReadValue**

AERERR_CODE AerParmMachineReadValue (LPCTSTR pszFile, AXISINDEX iAxis, LPCTSTR pszName, PDOUBLE pdValue, PDOUBLE pdDefValue);

Declare Function AerParmMachineReadValue Lib "AERSYS.DLL" (ByVal pszFile As String, ByVal iAxis As Long, ByVal pszName As String, ByRef pdValue As Double, ByRef pdDefValue As Double) As Long

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>pszFile</code></td>
<td>Name of machine parameter file (can be Null or Empty String).</td>
</tr>
<tr>
<td><code>iAxis</code></td>
<td>Which axis to retrieve from file (see constants below).</td>
</tr>
<tr>
<td><code>pszName</code></td>
<td>Name of machine parameter to read.</td>
</tr>
<tr>
<td><code>pdValue</code></td>
<td>Pointer to hold value read from file.</td>
</tr>
<tr>
<td><code>pdDefValue</code></td>
<td>Pointer to value holding the default value (may be NULL).</td>
</tr>
</tbody>
</table>

This function retrieves the value of the specified machine parameter from its .INI file. If the file name is passed as NULL, it retrieves the filename from the “project file” INI\U600.INI. If a default value is specified, then this value is returned as the value if the parameter could not be found in the file.

**C Language and LabView Constants**

AXISINDEX_1 to AXISINDEX_16

**VB Constants**

aerAxisIndex1 to aerAxisIndex16

**See Also**

AerRegGetFileName
13.21. AerParmMachineSetValue

AERERR_CODE AerParmMachineSetValue (HAERCTRL hAerCtrl, AXISINDEX iAxis, DWORD dwParm, DOUBLE fdValue);

Declare Function AerParmMachineSetValue Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal iAxis As Long, ByVal dwParm As Long, ByVal fdValue As Double) As Long

Parameters

- **hAerCtrl** Handle to the axis processor card.
- **iAxis** Physical axis index (see constants below).
- **dwParm** Parameter number, (see constants below).
- **fdValue** New value for the parameter.

`AerParmMachineSetValue` sets a machine parameter value on the controller. An error code will be returned if there is a problem setting the parameter. In addition, some parameters are read-only. Use `AerParmMachineGetInfo` to find out the minimum, maximum, and other characteristics of the parameter.

Parameters do have minimum and maximum allowed values.

**C Language and LabView Constants**

- **AXISINDEX_1**
- **AXISINDEX_16**
- **MACHPARM_XXXX**

**VB Constants**

- **aerAxisIndex1**
- **aerAxisIndex16**
- **aerMachParmXXXX**

**See Also**

- `AerParmMachineGetValue`
- `AerParmMachineGetInfo`

**Example**

- Samples\Lib\AexSys.C
13.22. AerParmMachineWriteValue

AERERR_CODE AerParmMachineWriteValue (LPCTSTR pszFile, AXISINDEX iAxis, LPCTSTR pszName, DOUBLE dValue);

Declare Function AerParmMachineWriteValue Lib "AERSYS.DLL" (ByVal pszFile As String, ByVal iAxis As Long, ByVal pszName As String, ByVal dValue As Double) As Long

Parameters

- **pszFile**: Name of machine parameter file (can be Null or Empty String).
- **iAxis**: Which axis to retrieve from file (see constants below).
- **pszName**: Name of machine parameter to write.
- **dValue**: Value to write to file.

This function writes the value of the specified machine parameter to its .INI file. If the file name is passed as NULL, it retrieves the filename from the “project file” INI\U600.INI.

C Language and LabView Constants

- **AXISINDEX_1** to **AXISINDEX_16**

VB Constants

- **aerAxisIndex1** to **aerAxisIndex16**

See Also

- *AerRegGetFileName*
### 13.23. AerParmMAxisGetValue

AERERR_CODE AerParmMAxisGetValue( HAERCTRL hAerCtrl, AXISMASK mAxis, DWORD dwParm, PDWORD pdwValueArray );

Declare Function AerParmMAxisGetValue Lib "AERSYS.DLL" Lib "AERSYS.DLL" (ByVal hAerCtrl As Long, ByVal mAxis As Long, ByVal dwParm As Long, ByRef pdwValueArray As Long) As Long

**Parameters**

- **hAerCtrl** Handle to the axis processor card.
- **mAxis** Axis mask of axes to set value (see constants below).
- **dwParm** Which axis parameter to set (see constants below).
- **pdwValueArray** Pointer to array of DWORD to hold array values.

This function gets the axis parameter specified by **dwParm** for all axes in **mAxis** from the controller. The values are returned in **pdwValueArray**. The array is packed; it only needs to be allocated for each of the axes specified in **mAxis**.

**C Language and LabView Constants**

- **AXISMASK_1** to **AXISMASK_16**
- **AXISPARM_XXXX**

**VB Constants**

- **aerAxisMask1** to **aerAxisMask16**
- **aerAxisParmXXXX**
13.24. AerParmMAxisSetValue

AERERR_CODE AerParmMAxisSetValue (HAERCTRL hAerCtrl, AXISMASK mAxis, DWORD dwParm, DWORD dwValue);

Declare Function AerParmMAxisSetValue Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal mAxis As Long, ByVal dwParm As Long, ByVal dwValue As Long) As Long

Parameters

  hAerCtrl   Handle to the axis processor card.
  mAxis      Axis mask of axes to set value (see constants below).
  dwParm     Which axis parameter to set (see constants below).
  dwValue    Value to set parameters.

This function sets the axis parameter specified by dwParm to dwValue for all the axes specified in maxis, on the controller.

C Language and LabView Constants

  AXISMASK_1
  to
  AXISMASK_16

  AXISPARM_XXXX

VB Constants

  aerAxisMask1
  to
  aerAxisMask16

  aerAxisParmXXXX
13.25. AerParmTaskDownloadFile

AERERR_CODE AerParmTaskDownloadFile (HAERCTRL hAerCtrl, LPCTSTR pszFile, TASKMASK mTask);

Declare Function AerParmTaskDownloadFile Lib "AERSYS.DLL" (ByVal hAerCtrl As Long, ByVal pszFile, TASKMASK mTask) As Long

Parameters

- **hAerCtrl** Handle to axis processor card.
- **pszFile** File to download (can be Null or Empty String).
- **mTask** Mask of the tasks to download parameters for.

The `AerParmTaskDownloadFile` function downloads the task parameters to the controller for the specified tasks. If the file name is passed as NULL, it retrieves the filename from the “project file” INI\U600.INI. The default filename currently in use by the user can be retrieved with the `AerRegGetFileName` function.

**C Language and LabView Constants**

- `TASKMASK_1` to `TASKMASK_4`

**VB Constants**

- `aerTaskMask1` to `aerTaskMask4`

**See Also**

- `AerRegGetFileName`
13.26. AerParmTaskGetCount

AERERR_CODE AerParmTaskGetCount (HAERCTRL hAerCtrl, PDWORD pdwCount);

Declare Function AerParmTaskGetCount Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByRef pdwCount As Long) As Long

Parameters

- **hAerCtrl**: Handle to the axis processor card.
- **pdwCount**: Pointer to double word to receive the number of task parameters.

Returns the number of task parameters.

See Also

*AerParmTaskGetValue*
13.27. AerParmTaskGetInfo

AERERR_CODE AerParmTaskGetInfo (HAERCTRL hAerCtrl, DWORD dwParm, PAER_PARM_INFO pInfo);

AERERR_CODE AerParmTaskGetInfoEx (HAERCTRL hAerCtrl, DWORD dwParm, LPTSTR pszName, PDOUBLE pdMin, PDOUBLE pdMax, PWORD pwAttr);

AERERR_CODE AerParmTaskGetInfoEx2 (HAERCTRL hAerCtrl, DWORD dwParm, LPTSTR pszName, PDOUBLE pdMin, PDOUBLE pdMax, PWORD pwAttr, PDOUBLE pdDefault, PDWORD pdwDisplaySubGroup);

Declare Function AerParmTaskGetInfoEx Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal dwParm As Long, ByVal pszName As String, ByRef pdMin As Double, ByRef pdMax As Double, ByRef pwAttr As Integer) As Long

Declare Function AerParmTaskGetInfoEx2 Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal dwParm As Long, ByVal pszName As String, ByRef pdMin As Double, ByRef pdMax As Double, ByRef pwAttr As Integer, ByRef pdDefault As Double, ByRef pdwDisplaySubGroup As Long) As Long

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hAerCtrl</td>
<td>Handle to the axis processor card.</td>
</tr>
<tr>
<td>dwParm</td>
<td>Parameter number, (see constants below).</td>
</tr>
<tr>
<td>pInfo</td>
<td>Pointer to structure to receive the parameter information</td>
</tr>
<tr>
<td>(C Language Only, refer to AER_PARM_INFO in Appendix C: Structures).</td>
<td></td>
</tr>
<tr>
<td>pszName</td>
<td>Pointer to buffer to hold name of parameter.</td>
</tr>
<tr>
<td>pdMin</td>
<td>Pointer to hold minimum value of parameter.</td>
</tr>
<tr>
<td>pdMax</td>
<td>Pointer to hold maximum value of parameter.</td>
</tr>
<tr>
<td>pwAttr</td>
<td>Pointer to hold parameter attribute (see constants below).</td>
</tr>
<tr>
<td>pdDefault</td>
<td>Pointer to hold the Default value of parameter</td>
</tr>
<tr>
<td>pdwDisplaySubGroup</td>
<td>Pointer to hold the subgroup to display the parameter in</td>
</tr>
<tr>
<td></td>
<td>the parameter editor.</td>
</tr>
</tbody>
</table>

All string variables in MS Visual Basic, passed by reference (ByRef), must be declared as fixed length strings within your program, long enough to hold the string value returned by the function. Also, those string variables which are passed, with another parameter indicating the length of the string variable, must also be fixed length strings, otherwise, you would not be able to pass the length of the string. For example, to declare a fixed length string:

```vbnet
DIM sGlobStr as STRING * 50 ; 50 characters long
```
This function returns data concerning a task parameter. If the parameter number provided by the user is not a valid number, the function will still return with no error code. In this case, the “wAttr” mask of the pInfo structure will not have the VALID bit set.

C Language and LabView Constants

\[ \text{TASKPARM}_\text{XXXX} \]

\[ \text{PARM\_ATTR}_\text{XXXX} \]

VB Constants

\[ \text{aerTaskParm}_\text{XXXX} \]

\[ \text{aerParmAttr}_\text{XXXX} \]

Example

Samples\Lib\AexSys.C
13.28. AerParmTaskGetValue

AERERR_CODE AerParmTaskGetValue( HAERCTRL hAerCtrl, TASKINDEX iTask, DWORD dwParm, PDOUBLE pfdValue );

Declare Function AerParmTaskGetValue Lib “AERSYS.DLL” ( ByVal hAerCtrl As Long, ByVal iTAxis As Long, ByVal dwParm As Long, ByRef pfdValue As Double) As Long

Parameters

hAerCtrl  Handle to the axis processor card.
iTask     Task index (see constants below).
dwParm    Parameter number (see constants below).
pfdValue  Pointer to double word to receive the current parameter value.

This function returns a task parameter value from the controller.

C Language and LabView Constants

    TASKINDEX_1
to
    TASKINDEX_4

    TASKPARM_XXXX

VB Constants

    aerTaskIndex1
to
    aerTaskIndex4

    aerTaskParmXXXX

See Also

AerParmTaskSetValue

Example

    Samples\Lib\AexSys.C
13.29. AerParmTaskReadValue

AERERR_CODE AerParmTaskReadValue (LPCTSTR pszFile, TASKINDEX iTask, LPCTSTR pszName, PDOUBLE pdValue, PDOUBLE pdDefValue);

Declare Function AerParmTaskReadValue Lib “AERSYS.DLL” (ByVal pszFile As String, ByVal iTask As Long, ByVal pszName As String, ByRef pdValue As Double, ByRef pdDefValue As Double) As Long

Parameters

- **pszFile**: Name of task parameter file (can be Null or Empty String).
- **iTask**: Which task to retrieve from file (see constants below).
- **pszName**: Name of task parameter to read.
- **pdValue**: Pointer to hold value read from file.
- **pdDefValue**: Pointer to value holding the default value (may be NULL).

This function retrieves the value of the specified task parameter from the .INI file. If the file name is passed as NULL, it retrieves the filename from the “project file” INI\U600.INI. If a default value is specified, then this value is returned as the value if the parameter could not be found in the file.

C Language and LabView Constants

- `TASKINDEX_1` to `TASKINDEX_4`

VB Constants

- `aerTaskIndex1` to `aerTaskIndex4`

See Also

- `AerRegGetFileName`
13.30. AerParmTaskSetValue

AERERR_CODE AerParmTaskSetValue (HAERCTRL hAerCtrl, TASKINDEX iTask, DWORD dwParm, DOUBLE fdValue);

Declare Function AerParmTaskSetValue Lib "AERSYS.DLL" (ByVal hAerCtrl As Long, ByVal iTask As Long, ByVal dwParm As Long, ByVal fdValue As Double) As Long

Parameters
   hAerCtrl Handle to the axis processor card.
   iTask Task index (see constants below).
   dwParm Parameter number, (see constants below).
   fdValue New value for the parameter.

AerParmTaskSetValue sets a task parameter value on the controller. An error code will be returned if there is a problem setting the parameter. In addition, some parameters are read-only. Use AerParmTaskGetInfo to find the minimum, maximum, and other characteristics of the parameter.

Parameters do have minimum and maximum allowed values.

C Language and LabView Constants

   TASKINDEX_1
to
   TASKINDEX_4

   TASKPARM_XXXX

VB Constants

   aerTaskIndex1
to
   aerTaskIndex4

   aerTaskParmXXXX

See Also

   AerParmTaskGetValue
   AerParmTaskGetInfo

Example

   Samples\Lib\AexSys.C
13.31. AerParmTaskWriteValue

AERERR_CODE AerParmTaskWriteValue (LPCTSTR pszFile, TASKINDEX iTask,
  LPCTSTR pszName, DOUBLE dvalue);

Declare Function AerParmTaskWriteValue Lib “AERSYS.DLL” (ByVal pszFile As
  String, ByVal iTask As Long, ByVal pszName As String, ByVal dValue
  As Double) As Long

Parameters
  pszFile Name of task parameter file (can be Null or Empty String).
  iTask Which task to write to file (see constants below).
  pszName Name of task parameter to write.
  dValue Value to write to file.

This function writes the value of the specified task parameter to its .INI file. If the file
name is passed as NULL, it retrieves the filename from the “project file” INIU600.INI.

C Language and LabView Constants

			TASKINDEX_1
  to
  TASKINDEX_4

VB Constants

aerTaskIndex1
to
aerTaskIndex4

See Also

AerRegGetFileName
CHAPTER 14: PROBE FUNCTIONS

14.1. Introduction

The Probe functions give the programmer access to a binary input for latching position information. This is usually done to “touch off” a part via a sensitive touch sensor, such as a precision touch probe. This chapter provides functionality similar to the PROBE and G_CNC commands (see the U600MMI.hlp file for more information).
14.2. AerProbeDisable

AERERR_CODE AerProbeDisable (HAERCTRL hAerCtrl, AXISINDEX iAxis);

Declare Function AerProbeDisable Lib “AERSYS.DLL” (ByVal hAerCtrl As Long,
ByVal iAxis As Long) As Long

Parameters

  hAerCtrl Handle to axis processor card.
  iAxis Axis to enable probe (see constants below).

The AerProbeDisable function disables the probe.

C Language and LabView Constants

  AXISINDEX_1 to AXISINDEX_16

VB Constants

  aerAxisIndex1 to aerAxisIndex16

Example

  See Section 14.7.
14.3. **AerProbeEnable**

AERERR_CODE AerProbeEnable (HAERCTRL hAerCtrl, AXISINDEX iAxis);

Declare Function AerProbeEnable Lib “AERSYS.DLL” (ByVal hAerCtrl As Long,
ByVal iAxis As Long) As Long

**Parameters**

- *hAerCtrl* Handle to axis processor card.
- *iAxis* Axis to enable probe (see constants below).

This function will enable the probe. The probe status then indicate
AER_PROBESTATUS_ARMED (C Language), aerProbeStatusArmed (VB).

**C Language and LabView Constants**

- `AXISINDEX_1` to `AXISINDEX_16`

**VB Constants**

- `aerAxisIndex1` to `aerAxisIndex16`

**Example**

See Section 14.7.
14.4. AerProbeGetPosition

AERERR_CODE AerProbeGetPosition (HAERCTRL hAerCtrl, AXISINDEX iAxis, PAER_PROBE_POS_PACKET pPos);

AERERR_CODE AerProbeGetPositionEx (HAERCTRL hAerCtrl, AXISINDEX iAxis, PWORD pwStatus, PDWORD pdwPos);

Declare Function AerProbeGetPositionEx Lib "AERSYS.DLL" (ByVal hAerCtrl As Long, ByVal iAxis As Long, ByRef pwStatus As Integer, ByRef pdwPos As Long) As Long

Parameters
- hAerCtrl Handle to axis processor card.
- iAxis Axis to get position.
- pPos Pointer to AER_PROBE_POS_PACKET to receive information.
- pwStatus Pointer to hold probe status word (AER_PROBESTATUS_xxxx constant).
- pdwPos Pointer to hold current position.

This function gets the position when the probe was hit. If valid, pwStatus will have AER_PROBESTATUS_VALIDPOS.

C Language and LabView Constants
- AXISINDEX_1 to AXISINDEX_16
- AER_PROBESTATUS_XXXX

VB Constants
- aerAxisIndex1 to aerAxisIndex16
- aerProbeStatusXXXX

Example
See Section 14.7.
14.5. AerProbeGetStatus

AERERR_CODE AerProbeGetStatus (HAERCTRL hAerCtrl, AXISINDEX iAxis, PAER_PROBE_STATUS_PACKET pStatus);

AERERR_CODE AerProbeGetStatusEx (HAERCTRL hAerCtrl, AXISINDEX iAxis, PWORD pwStatus, PWORD pwInput, PWORD pwLevel);

Declare Function AerProbeGetStatusEx Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal iAxis As Long, ByRef pwStatus As Integer, ByRef pwInput As Integer, ByRef pwLevel As Integer) As Long

Parameters
- hAerCtrl Handle to axis processor card.
- iAxis Axis to get status (see constants below).
- pStatus Pointer to AER_PROBE_STATUS_PACKET to receive information (C Language Only).
- pwStatus Pointer to hold probe status word (see constants below).
- pwInput Pointer to current input setting.
- pwLevel Pointer to current logic level.

This function gets the current probe status for an axis.

C Language and LabView Constants
- AXISINDEX_1
  to
- AXISINDEX_16

- AER_PROBESTATUS_XXXX

VB Constants
- aerAxisIndex1
  to
- aerAxisIndex16

- aerProbeStatusXXXX

Example
- See Section 14.7.
14.6. AerProbeSetInput

AERERR_CODE AerProbeSetInput (HAERCTRL hAerCtrl, AXISINDEX iAxis, WORD wInput, WORD wLevel);

Declare Function AerProbeSetInput Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal iAxis As Long, ByVal wInput As Integer, ByVal wLevel As Integer) As Long

Parameters

- **hAerCtrl** Handle to axis processor card.
- **iAxis** Axis to check for probe hit (see constants below).
- **wInput** Which binary input to use for probe (0 to 15 for standard input, -1 for high speed position latch).
- **wLevel** Logic level to trigger probe hit. If 0, triggered when input is low. If 1, triggered when input is high.

The AerProbeSetInput function sets an input to an axis for the probe condition.

C Language and LabView Constants

- **AXISINDEX_1** to **AXISINDEX_16**

VB Constants

- **aerAxisIndex1** to **aerAxisIndex16**

Example

See Section 14.7
14.7. Example Code Fragment to Use Probe Functions

The following code fragment shows how to use the probe functions.

```c
HAERCTRL hAerCtrl;
WORD  wInput = 8;       // using Binary Input 8
WORD  wLevel = 0;       // Input is active - low
AXISINDEX iProbeAxis = AXISINDEX_4;  // Looking for
// probe on axis 4
AERERR_CODE eRc;

// Open communication to card
eRc = AerSysOpen( AER_UNIDEX_DEFAULT, AER_CARD_DEFAULT,
    &hAerCtrl );
RETURN_ON_ERR( eRc );

// Setup the probe input
eRc = AerProbeSetInput( hAerCtrl, iProbeAxis, wInput,
    wLevel );
....

///// The following are the steps necessary to move a set
///// of axes and to enable the probe.

LONG lTargetArray[3];  // array of targets
DWORD dwPos;
WORD wStatus;

// Move the XYZ Axis in coordinated move to (1000, 3000,
// 1000) counts at 1000 counts/sec
lTargetArray[0] = 1000;
lTargetArray[1] = 3000;
lTargetArray[2] = 1000;
eRc = AerMoveLinear( hAerCtrl, AXISMASK_1 | AXISMASK_2 |
    AXISMASK_3, &lTargetArray[0], 1000 );

// Arm the Probe
eRc = AerProbeEnable( hAerCtrl, iProbeAxis );

// Start moving the axis
eRc = AerMoveFreerun( hAerCtrl, iProbeAxis, 1, 100 );

AerProbeGetPositionEx( hAerCtrl, iProbeAxis, &wStatus,
    &dwPos );

// wait until we get a hit
while( !(wStatus & AER_PROBESTATUS_VALIDPOS) )
{
    AerProbeGetPositionEx( hAerCtrl, iProbeAxis, &wStatus,
        &dwPos );
    Sleep(100);
}
```
//
// To account for decel, make a move back to position
// where probe hit
eRc=AerMoveAbsolute( hAerCtrl, iProbeAxis, dwPos, 100 );
....
AerSysClose( hAerCtrl );

∇ ∇ ∇
CHAPTER 15: PROFILE FUNCTIONS

15.1. Introduction

The Profiling Functions give the programmer low level access to derive their own profile motion similar to the PVT CNC command. See the U600MMI.hlp file for more information. Each axis has its own “profile queue.” As profile data is added to the queue for a specified axis, the information is processed and the motion is generated. The profiling data consists of a target position, the time it should take to reach the target position, starting velocity, and the ending velocity that should be reached at the target position.

The profile queue for each axis is limited in size. This size is specified by the axis parameter PROFQSIZE. As the profiling data is executed, it is removed from the queue so that more profiling data can be added. The current depth of the queue can be determined by the axis parameter PROFQDEPTH. The depth cannot exceed the size.
15.2. AerProfileFlushQueue

AERERR_CODE AerProfileFlushQueue (HAERCTRL hAerCtrl, AXISINDEX iAxis);

Declare Function AerProfileFlushQueue Lib “AERSYS.DLL” (ByVal hAerCtrl As Long,
ByVal iAxis As Long) As Long

Parameters
  
  hAerCtrl  Handle to the axis processor card.
  iAxis     Which axis to flush the profile queue (see constants below).

The AerProfileFlushQueue function flushes the profiling queue for the specified axis.

C Language and LabView Constants
  
  AXISINDEX_1
  to
  AXISINDEX_16

VB Constants
  
  aerAxisIndex1
  to
  aerAxisIndex16

Example
  
  Samples\Lib\AexProf\AexProf.C
15.3. **AerProfileFlushQueueMulti**

AERERR_CODE AerProfileFlushQueueMulti (HAERCTRL hAerCtrl, AXISINDEX mAxis);

Declare Function AerProfileFlushQueueMulti Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal mAxis As Long) As Long

**Parameters**

- **hAerCtrl** Handle to the axis processor card.
- **mAxis** Axis mask specifying which axes to flush the profile queue (see constants below).

This function flushes the profiling queue for the specified axes.

**C Language and LabView Constants**

- **AXISMASK_1**
- **AXISMASK_16**

**VB Constants**

- **aerAxisMask1**
- **aerAxisMask16**

**Example**

Samples\Lib\AexProf\AexProf.C
15.4. AerProfileLoadQueue

AERERR_CODE AerProfileLoadQueue (HAERCTRL hAerCtrl, AXISINDEX iAxis,
PAER_PROFILE pProf);

AERERR_CODE AerProfileLoadQueueEx (HAERCTRL hAerCtrl, AXISINDEX iAxis,
LONG lPoint, DWORD dwTime, LONG lVelStart, LONG lVelEnd);

Declare Function AerProfileLoadQueueEx Lib “AERSYS.DLL” (ByVal
hAerCtrl As Long, ByVal iAxis As Long, ByVal lPoint As Long, ByVal dwTime As
Long, ByVal lVelStart As Long, ByVal lVelEnd As Long) As Long

Parameters
  hAerCtrl Handle to the axis processor card.
  iAxis Which axis to profile (see constants below).
  pProf Pointer to the profiling information for the axis specified in iAxis.
  lPoint Target position in counts.
  dwTime Motion time in msec.
  lVelStart Starting velocity in msec/sec.
  lVelEnd Ending velocity in msec/sec.

This function loads the specified profile information into the profile queue. The profile
information is added to the queue for the axis specified in iAxis.

C Language and LabView Constants
  AXISINDEX_1
to
  AXISINDEX_16

VB Constants
  aerAxisIndex1
to
  aerAxisIndex16

Example
  Samples\Lib\AexProf\AexProf.C
15.5. **AerProfileLoadQueueMulti**

AERERR_CODE AerProfileLoadQueueMulti (HAERCTRL hAerCtrl, AXISMASK mAxis, PAER_PROFILE pProfArray);

AERERR_CODE AerProfileLoadQueueMultiEx (HAERCTRL hAerCtrl, AXISMASK mAxis, PLONG plPosArray, PDWORD pdwTimeArray, PLONG plVelStartArray, PLONG plVelEndArray);

Declare Function AerProfileLoadQueueMultiEx Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal mAxis As Long, ByRef plPosArray As Long, ByRef pdwTimeArray As Long, ByRef plVelStartArray As Long, ByRef plVelEndArray As Long) As Long

**Parameters**

- **hAerCtrl**: Handle to the axis processor card.
- **mAxis**: Axis mask specifying which axes to profile (see constants below).
- **pProfArray**: Pointer to an array containing the profiling information for each axis specified in `plPosArray` – Array of target positions in counts (C Language Only).
- **pdwTimeArray**: Array of motion times in msec.
- **plVelStartArray**: Array of starting velocities in msec/sec.
- **plVelEndArray**: Array of ending velocities in msec/sec.

This function loads the specified profile information into the profile queue. The profile information is added to the queue for each axis specified in `mAxis`. The motion between the specified axes will be coordinated.

All arrays are packet arrays. The `pProfArray` is a pointer to an array of AER_PROFILE points. If three axes are specified in the axis mask, then the first three elements in the arrays are applied to each axis in order.

**C Language and LabView Constants**

- `AXISMASK_1` to `AXISMASK_16`

**VB Constants**

- `aerAxisMask1` to `aerAxisMask16`

**Example**

Samples\Lib\AexProf\AexProf.C
15.6. **AerProfileStartQueue**

AERERR_CODE AerProfileStartQueue (HAERCRTL hAerCtrl, AXISINDEX iAxis);

Declare Function AerProfileStartQueue Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal iAxis As Long) As Long

**Parameters**
- *hAerCtrl* Handle to the axis processor card.
- *iAxis* Which axis to start the profile queue (see constants only).

This function starts the profiling queue for the specified axis.

**C Language and LabView Constants**
- `AXISINDEX_1` to `AXISINDEX_16`

**VB Constants**
- `aerAxisIndex1` to `aerAxisIndex16`
15.7. AerProfileStartQueueMulti

AERERR_CODE AerProfileStartQueueMulti (HAERCTRL hAerCtrl, AXISMASK mAxis);

Declare Function AerProfileStartQueueMulti Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal mAxis As Long) As Long

Parameters

hAerCtrl Handle to the axis processor card.

mAxis Axis mask specifying which axes to start the profile queue (see constants below).

This function starts the profiling queue for the specified axes.

C Language and LabView Constants

AXISMASK_1
to
AXISMASK_16

VB Constants

aerAxisMask1
to
aerAxisMask16
15.8. AerProfileStopQueue

AERERR_CODE AerProfileStopQueue (HAERCTRL hAerCtrl, AXISINDEX iAxis);

Declare Function AerProfileStopQueue Lib “AERSYS.DLL” (ByVal hAerCtrl As Long,
ByVal iAxis As Long) As Long

Parameters
   hAerCtrl    Handle to the axis processor card.
   iAxis       Which axis to stop the profile queue for (see constants below).

The AerProfileStopQueue function stops the profiling queue for the specified axis.

C Language and LabView Constants
   AXISINDEX_1 to AXISINDEX_16

VB Constants
   aerAxisIndex1 to aerAxisIndex16
15.9. AerProfileStopQueueMulti

AERERR_CODE AerProfileStopQueueMulti (HAERCCTRL hAerCtrl, AXISMASK mAxis);

Declare Function AerProfileStopQueueMulti Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal mAxis) As Long

Parameters

hAerCtrl Handle to the axis processor card.
mAxis Axis mask specifying which axes to stop the profile queue (see constants below).

This function stops the profiling queue for the specified axes.

C Language and LabView Constants

AXISMASK_1 to AXISMASK_16

VB Constants

aerAxisMask1 to aerAxisMask16
CHAPTER 16: PROGRAMMING ERROR FUNCTIONS

In This Section:
- Introduction ............................................................................................16-1
- AerProgErrFlush ....................................................................................16-2
- AerProgErrGet .......................................................................................16-3
- AerProgErrGetAltStatusMessage ...........................................................16-4
- AerProgErrGetFaultMessage ..................................................................16-5
- AerProgErrGetMessage .........................................................................16-6
- AerProgErrGetMotionMessage ..............................................................16-7
- AerProgErrGetServoMessage ..................................................................16-8
- AerProgErrGetStatusMessage ...............................................................16-9
- AerProgErrSetUserFault ......................................................................16-10
- AerProgErrWaitModeGet ....................................................................16-11
- AerProgErrWaitModeSet ....................................................................16-12

16.1. Introduction
These functions return data about programming errors (defined by axis parameters), or
return strings. Refer to AerProgErrErrWaitModeSet or the introduction in Chapter 9:
Error Functions, for details on programming errors.
16.2. AerProgErrFlush

AERERR_CODE AerProgErrFlush (HAERCTRL hAerCtrl, AXISINDEX iAxis);

Declare Function AerProgErrFlush Lib “AERSYS.DLL” (ByVal hAerCtrl As Long,
ByVal iAxis As Long) As Long

Parameters

hAerCtrl Handle to the axis processor card.
iAxis Axis index specifying the axis whose fault to clear (see constants).

This function is used to clear the programming errors stored on the axis processor. Each axis stores its own programming errors. Error strings for each programming error are stored on the axis processor. Clearing the program errors will free memory on the axis processor.

C Language and LabView Constants

AXISINDEX_1 to AXISINDEX_16

VB Constants

aerAxisIndex1 to aerAxisIndex16

See Also

AerProgErrGet
AerProgErrGetMessage

Example

Samples\Lib\AexProg.C
16.3. AerProgErrGet

AERERR_CODE AerProgErrGet (HAERCTRL hAerCtrl, AXISINDEX iAxis,
  PAER_PROG_FAULT pFault);

Parameters
  
  hAerCtrl  Handle to the axis processor card.
  iAxis     Axis index specifying the axis to retrieve its fault axis (use the
            AXISINDEX_xxxx constants).
  pFault    Pointer to a AER_PROG_FAULT structure for an Aerotech program
            error.

This function retrieves the information about the command that caused the last
programming error condition for the specified axis.

See Also
  AerProgErrFlush
  AerProgErrGetMessage
  AerProgErrSetUserFault

Example
  Samples\Lib\AexProg.C
16.4. AerProgErrGetAltStatusMessage

AERERR_CODE AerProgErrGetAltStatusMessage (HAERCTRL hAerCtrl, WORD wMsg, LPTSTR pszMsg);

Declare Function AerProgErrGetAltStatusMessage Lib "AERSYS.DLL" (ByVal hAerCtrl As Long, ByVal wMsg As Integer, ByVal pszMsg As String) As Long

Parameters
- hAerCtrl Handle to the axis processor card.
- wMsg Message number to retrieve.
- pszMsg Pointer to the message string.

This function retrieves an ALTSTATUS axis parameter message ASCII text string that defines the 32 bits of the axis status. The messages are permanently stored within the memory of the UNIDEX 600 series controller card.

All string variables in MS Visual Basic, passed by reference (ByRef), must be declared as fixed length strings within your program, long enough to hold the string value returned by the function. Also, those string variables which are passed, with another parameter indicating the length of the string variable, must also be fixed length strings, otherwise, you would not be able to pass the length of the string. For example, to declare a fixed length string:

```
DIM sGlobStr as STRING * 50 ; 50 characters long
```

See Also
- AerProgErrGetMessage
- AerProgErrGetStatusMessage
- AerProgErrGetServoMessage
- AerProgErrGetMotionMessage
- AerProgErrGetAltStatusMessage

Example

```
Samples\Lib\AexProg.C
```
16.5. AerProgErrGetFaultMessage

AERERR_CODE AerProgErrGetFaultMessage (HAERCTRL hAerCtrl, WORD wMsg, LPTSTR pszMsg);

Declare Function AerProgErrGetFaultMessage Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal wMsg As Integer, ByVal pszMsg As String) As Long

Parameters
  hAerCtrl  Handle to the axis processor card.
  wMsg     Message number to retrieve.
  pszMsg   Pointer to the message string.

This function retrieves a FAULT axis parameter message (as an ASCII text string) that defines the function of the 32 bits of the system fault status. The messages are stored within the memory of the axis processor card. The message number is in the range of 0 through 31 as represented by the fault status bits.

All string variables in MS Visual Basic, passed by reference (ByRef), must be declared as fixed length strings within your program, long enough to hold the string value returned by the function. Also, those string variables which are passed, with another parameter indicating the length of the string variable, must also be fixed length strings, otherwise, you would not be able to pass the length of the string. For example, to declare a fixed length string:

```
DIM sGlobStr as STRING * 50 ; 50 characters long
```

See Also
  AerProgErrGetFaultMessage
  AerProgErrGetStatusMessage
  AerProgErrGetServoMessage
  AerProgErrGetMotionMessage
  AerProgErrGetAltStatusMessage

Example
  Samples\Lib\AexProg.C
16.6. AerProgErrGetMessage

AERERR_CODE AerProgErrGetMessage (HAERCTRL hAerCtrl, WORD wSubCode, WORD wMsg, LPCTSTR pszMsg);

Declare Function AerProgErrGetMessage Lib "AERSYS.DLL" (ByVal hAerCtrl As Long, ByVal wSubCode As Integer, ByVal wMsg As Integer, ByVal pszMsg As String) As Long

Parameters
- **hAerCtrl**: Handle to the axis processor card.
- **wSubCode**: Sub-Code 0 - 4 determining the type of message to return.
- **wMsg**: Message number to retrieve.
- **pszMsg**: Pointer to the message string.

This function retrieves a message text string stored on the axis processor. Several convenient “wrapper functions” are available that automatically use the appropriate **wSubCode** to retrieve the correct type of message.

All string variables in MS Visual Basic, passed by reference (ByRef), must be declared as fixed length strings within your program, long enough to hold the string value returned by the function. Also, those string variables which are passed, with another parameter indicating the length of the string variable, must also be fixed length strings, otherwise, you would not be able to pass the length of the string. For example, to declare a fixed length string:

```
DIM sGlobStr as STRING * 50 ; 50 characters long
```

<table>
<thead>
<tr>
<th>Message Type (axis parameter)</th>
<th><strong>wSubCode</strong> value</th>
<th>Wrapper function Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>FAULT</td>
<td>0</td>
<td>AerProgErrGetFaultMessage</td>
</tr>
<tr>
<td>STATUS</td>
<td>1</td>
<td>AerProgErrGetStatusMessage</td>
</tr>
<tr>
<td>SERVOSTATUS</td>
<td>2</td>
<td>AerProgErrGetServoMessage</td>
</tr>
<tr>
<td>MOTIONSTATUS</td>
<td>3</td>
<td>AerProgErrGetMotionMessage</td>
</tr>
<tr>
<td>ALTSTATUS</td>
<td>4</td>
<td>AerProgErrGetAltStatusMessage</td>
</tr>
</tbody>
</table>

For these macro functions, simply omit the **wSubcode** argument. Please see the documentation for the wrapper functions listed above, for details on the messages returned.

See Also
- AerProgErrGetFaultMessage
- AerProgErrGetStatusMessage
- AerProgErrGetServoMessage
- AerProgErrGetMotionMessage
- AerProgErrGetAltStatusMessage

Example
- Samples\Lib\AexProg.C
16.7. AerProgErrGetMotionMessage

AERERR_CODE AerProgErrGetMotionMessage (HAERCTRL hAerCtrl, WORD wMsg, LPTSTR pszMsg);

Declare Function AerProgErrGetMotionMessage Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal wMsg As Integer, ByVal pszMsg As String) As Long

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hAerCtrl</td>
<td>Handle to the axis processor card.</td>
</tr>
<tr>
<td>wMsg</td>
<td>Message number to retrieve.</td>
</tr>
<tr>
<td>pszMsg</td>
<td>Pointer to the message string.</td>
</tr>
</tbody>
</table>

This function retrieves a motion status message as an ASCII text string that defines the 32 bits of the system MOTIONSTATUS axis parameter. The messages are stored within the memory of the axis processor card. The message number is in the range of 0 through 31 as represented by the motion status bits.

All string variables in MS Visual Basic, passed by reference (ByRef), must be declared as fixed length strings within your program, long enough to hold the string value returned by the function. Also, those string variables which are passed, with another parameter indicating the length of the string variable, must also be fixed length strings, otherwise, you would not be able to pass the length of the string. For example, to declare a fixed length string:

```
DIM sGlobStr as STRING * 50 ; 50 characters long
```

See Also

- AerProgErrGetFaultMessage
- AerProgErrGetStatusMessage
- AerProgErrGetServoMessage
- AerProgErrGetMotionMessage
- AerProgErrGetAltStatusMessage

Example

```
Samples\Lib\AexProg.C
```
16.8. AerProgErrGetServoMessage

AERERR_CODE AerProgErrGetServoMessage (HAERCTRL hAerCtrl, WORD wMsg, LPTSTR pszMsg);

Declare Function AerProgErrGetServoMessage Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal wMsg As Integer, ByVal pszMsg As String) As Long

Parameters

- **hAerCtrl**  Handle to the axis processor card.
- **wMsg**  Message number to retrieve.
- **pszMsg**  Pointer to the message string.

This function retrieves a servo status message as an ASCII text string that defines the 32 bits of the system SERVOSTATUS axis parameter. The messages are stored within the memory of the axis processor card. The message number is in the range of 0 through 31 as represented by the axis servo status parameter bits.

All string variables in MS Visual Basic, passed by reference (ByRef), must be declared as fixed length strings within your program, long enough to hold the string value returned by the function. Also, those string variables which are passed, with another parameter indicating the length of the string variable, must also be fixed length strings, otherwise, you would not be able to pass the length of the string. For example, to declare a fixed length string:

```vbnet
DIM sGlobStr as STRING * 50 ; 50 characters long
```

See Also

- AerProgErrGetFaultMessage
- AerProgErrGetStatusMessage
- AerProgErrGetServoMessage
- AerProgErrGetMotionMessage
- AerProgErrGetAltStatusMessage

Example

Samples\Lib\AexProg.C
16.9. AerProgErrGetStatusMessage

AERERR_CODE AerProgErrGetStatusMessage (HAERCTRL hAerCtrl, WORD wMsg, LPTSTR pszMsg);

Declare Function AerProgErrGetStatusMessage Lib "AERSYS.DLL" (ByVal hAerCtrl As Long, ByVal wMsg As Integer, ByVal pszMsg As String) As Long

Parameters
   hAerCtrl Handle to the axis processor card.
   wMsg Message number to retrieve.
   pszMsg Pointer to the message string.

This function retrieves a STATUS axis parameter message as an ASCII text string that defines the 32 bits of the system status. The messages are stored within the memory of the axis processor card. The message number is in the range of 0 through 31 as represented by the axis status bits.

All string variables in MS Visual Basic, passed by reference (ByRef), must be declared as fixed length strings within your program, long enough to hold the string value returned by the function. Also, those string variables which are passed, with another parameter indicating the length of the string variable, must also be fixed length strings, otherwise, you would not be able to pass the length of the string. For example, to declare a fixed length string:

```
DIM sGlobStr as STRING * 50 ; 50 characters long
```

See Also

AerProgErrGetFaultMessage
AerProgErrGetStatusMessage
AerProgErrGetServoMessage
AerProgErrGetMotionMessage
AerProgErrGetAltStatusMessage

Example

Samples\Lib\AexProg.C
16.10. AerProgErrSetUserFault

AERERR_CODE AerProgErrSetUserFault (HAERCTRL hAerCtrl, AXISINDEX iAxis);

Declare Function AerProgErrSetUserFault Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal iAxis As Long) As Long

Parameters

hAerCtrl Handle to the axis processor card.
iAxis The axis number to generate a user fault for (see constants below).

This function generates a user-programming fault for the specified iAxis.

C Language and LabView Constants

AXISINDEX_1 to AXISINDEX_16

VB Constants

aerAxisIndex1 to aerAxisIndex16

See Also

AerProgErrGetFault

Example

Samples\Lib\AexProg.C
16.11. AerProgErrWaitModeGet

AERERR_CODE AerProgErrWaitModeGet (HAERCTRL hAerCtrl, PBOOL pbWait);

Declare Function AerProgErrWaitModeGet Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByRef pbSet As Long) As Long

Parameters

   hAerCtrl       Handle to the axis processor card.
   pbWait         Returns the state of the library error code return state.

This function returns the current wait mode. See AerProgErrWaitModeSet for more details.

See Also

   AerProgErrWaitModeSet

Example

   Samples\Lib\AexProg.C
16.12. AerProgErrWaitModeSet

AERERR_CODE AerProgErrWaitModeSet (HAERCTRL hAerCtrl, BOOL bWait);

Declare Function AerProgErrWaitModeSet Lib "AERSYS.DLL" (ByVal hAerCtrl As Long, ByVal bSet As Long) As Long

Parameters
  hAerCtrl Handle to the axis processor card.
  bWait Set True to hold function return until function completion.

The wait mode should be set to 1 for maximum error checking, or to 0 for maximum library function execution speed. In the wait mode, every library function that communicates with the axis processor will wait until the axis processor has fully completed the function before that library function returns. If the axis processor finds an error while processing, the programming error code for that error returns in the library function call (all these errors are in the form AER_960PROG_xxxx).

In “non-wait” mode, the library function will not wait until the axis processor completes the function before returning (unless the library function is requesting data back from the axis processor, in which case it always waits). This mode allows the front-end application to continue processing while the axis processor completes the function. In non-wait mode the user must read the programming errors manually to check for an error.

Also, in the non-wait mode, the library will wait for and return an error in some cases. If the axis processor never acknowledges the command, the function will return an AERERR_DRV_xxxx error code.

If the library function is requesting data from the axis processor and the axis processor does not return the correct amount of data, then the library will return an AERERR_DRV_RECV_LENGTH error.

The default is with the wait mode on (=1). Users can examine the current wait mode by typing “INFO” at an AERDEBUG prompt.

See Also
  AerProgErrWaitModeGet

Example
  Samples\Lib\AexProg.C

∇  ∇  ∇
17.1. Introduction

These functions allow the user to obtain information about, and set associated parameters for a program residing on the axis processor card. Please see the Win NT/95 U600 Series Users Guide, P/N EDU157, under CNC Program Execution for descriptions of programs and program execution.

These functions do not allow the user to upload or download CNC programs to the controller. This functionality is packaged in the CNC compiler and is not available at the C library level.

These functions do not allow the user to execute, or manipulate the execution of, a CNC program. See the AerTaskProgramXXXX functions for these capabilities.

These functions are used by the CNC compiler and MMI and are not normally useful to the application programmer.

Many of these functions require a parameter of type AER_PROG_HANDLE. The programmer must define this structure before calling such functions. The main element of this structure is the program name, by which the axis processor identifies the program.
17.2. AerProgramFree

AERERR_CODE AerProgramFree (HAERCTRL hAerCtrl, PAER_PROG_HANDLE pHandle);

Declare Function AerProgramFree Lib “AERSYS.DLL” (ByVal hAerCtrl As Long,
ByVal pName As String) As Long

Parameters

hAerCtrl Handle to an Aerotech control.

pHandle Pointer to a program handle (see AER_PROG_HANDLE structure) (C Language Only).

pName A string variable passed ByVal, indicating the name of the CNC program (VB Only).

This function will free the memory on the axis processor card that is currently being used
to store the program specified by the pHandle parameter. This will allow the memory to
be reallocated for other uses. The programmer must provide a program name through the
pName/pHandle parameter.

See Also

AerProgramAllocate
AerProgramGetHandle
17.3. **AerProgramGetBreakPoint**

AERERR_CODE AerProgramGetBreakPoint (HAERCTRL hAerCtrl, PAER_PROG_HANDLE pHandle, DWORD dwLineUser, PBOOL pbOn);

Declare Function AerProgramGetBreakPoint Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal pName As String, ByVal dwLineUser As Long, ByRef pbOn As Long) As Long

**Parameters**

- **hAerCtrl**
  Handle to an Aerotech control.

- **pHandle**
  String variable containing the program handle (see AER_PROG_HANDLE) (C Language Only).

- **pName**
  A string variable passed ByVal, indicating the name of the CNC program (VB Only).

- **dwLineUser**
  The line number to get the breakpoint status of.

- **pbOn**
  A pointer to receive the breakpoint status of the program line.

This function will return the state of the breakpoint status for the specified user line number in the specified program handle. The `pbOn` parameter will be set to AER_BP_ON (true) if a break point is set on the line or AER_BP_OFF (false) if no breakpoint is set on the line. The programmer must provide the program name through the `pHandle` parameter.

**C Language and LabView Constants**

AER_BP_XXXX

**VB Constants**

aerBrkPntXXXX

**See Also**

AerProgramSetBreakPoint  
AerProgramGetLine
17.4. AerProgramGetHandle

AERERR_CODE AerProgramGetHandle (HAERCTRL hAerCtrl, DWORD dwNum, PAER_PROG_HANDLE pHandle);

Declare Function AerProgramGetHandle Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal dwNum As Long, ByVal pName As String) As Long

Parameters
- **hAerCtrl** - Handle to an Aerotech control.
- **dwNum** - Program index.
- **pHandle** - String variable that will receive the program handle (see AER_PROG_HANDLE) (C Language Only).
- **pName** - A string variable passed ByVal, indicating the name of the CNC program (VB Only).

This function will return the program handle from the program index specified by dwNum. The program indices are zero-based. Passing a dwNum program index greater than the number of programs present (-1) will return a null (empty) string, with the function error return code equal to zero.

<table>
<thead>
<tr>
<th>All string variables in MS Visual Basic, passed by reference (ByRef), must be declared as fixed length strings within your program, long enough to hold the string value returned by the function. Also, those string variables which are passed, with another parameter indicating the length of the string variable, must also be fixed length strings, otherwise, you would not be able to pass the length of the string. For example, to declare a fixed length string:</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIM sGlobStr as STRING * 50 ; 50 characters long</td>
</tr>
</tbody>
</table>

See Also
- AerProgramGetNumber
17.5. **AerProgramGetHeader**

AERERR_CODE AerProgramGetHeader (HAERCTRL hAerCtrl, 
   PAER_PROG_HANDLE pHandle, PAER_PROG_HEADER pHeader)

**Parameters**

- **hAerCtrl**: Handle to an Aerotech control.
- **pHandle**: Address of the variable that will receive AER_PROG_HANDLE.
- **pHeader**: Address of the variable that will receive AER_PROG_HEADER.

This function retrieves header information for the specified program handle.

**See Also**

*AerProgramGetInfo*
17.6. AerProgramGetInfo

AERERR_CODE AerProgramGetInfo (HAERCTRL hAerCtrl, PAER_PROG_HANDLE pHandle, PAER_PROG_INFO pInfo);

Parameters

hAerCtrl Handle to an Aerotech control.
pHandle Pointer to completed AER_PROG_HANDLE structure.
pInfo Address of the variable that will receive AER_PROG_INFO.

This function returns information about a program residing on the axis processor card. The information returned in the pInfo structures includes data such as number of lines, number of labels, number of variables, and line (if any) currently being executed. The programmer must provide the program name through the pHandle parameter.

See Also

AerProgramAllocate
AerProgramGetHandle
17.7. AerProgramGetNumber

AERERR_CODE AerProgramGetNumber (HAERCTRL hAerCtrl, PAER_PROG_HANDLE pHandle, PDWORD pdw960ProgNum);

Declare Function AerProgramGetNumber Lib "AERSYS.DLL" (ByVal hAerCtrl As Long, ByVal pHandle As String, ByRef pdw960ProgNum As Long) As Long

Parameters

hAerCtrl Handle to an Aerotech control.
pHandle Pointer to string containing a program handle string variable to receive the program handle (see AER_PROG_HANDLE).
pdw960ProgNum Pointer to receive the program number on the axis processor card.

This function will return the program number for the specified program handle (AER_PROG_HANDLE). The programmer must provide the program name through the pHandle parameter. It will return an error if the passed handle does not specify an existing program on the axis processor.

See Also

AerProgramGetHandle
17.8. **AerProgramSetBreakPoint**

AERERR_CODE AerProgramSetBreakPoint (HAERCTRL hAerCtrl, PAER_PROG_HANDLE pHandle, DWORD dwLineUser, DWORD dwOn_Off);

Declare Function AerProgramSetBreakPoint Lib "AERSYS.DLL" (ByVal hAerCtrl As Long, ByVal pName As String, ByVal dwLineUser As Long, ByVal dwOn_Off As Long) As Long

**Parameters**

- **hAerCtrl**  Handle to an Aerotech control.
- **pHandle**  Pointer to completed AER_PROG_HANDLE structure (C Language).
- **pName**  String containing program name (VB Only)
- **dwLineUser**  Program user line index.
- **dwOn_Off**  Break point on/off.

This function allows a breakpoint to be set at the specified user line number in the specified program handle. The `dwOn_Off` parameter may be set equal to AER_BP_ON, AER_BP_OFF, or AER_BP_TOGGLE. The toggle value changes the value from its current state; for example sets it to on if it is currently off. The programmer must provide the program name through the `pName/pHandle` parameter.

**C Language and LabView Constants**

AER_BP_XXXX

**VB Constants**

aerBrkPntXXXX

**See Also**

AerProgramGetLine  
AerProgramSetBreakPoint
CHAPTER 18: PSO (LASER) FUNCTIONS

In This Section:
- Introduction ................................................................. 18-1
- AerPSOAllocateFiringTable ............................................. 18-2
- AerPSOFirePulse ............................................................ 18-3
- AerPSOSetDigitalOutput .................................................. 18-4
- AerPSOSetFiringDist ...................................................... 18-5
- AerPSOSetMultAnalogOutput .......................................... 18-6
- AerPSOSetPulse ............................................................. 18-7
- AerPSOSetTracking ....................................................... 18-9
- AerPSOStopFiring .......................................................... 18-10
- AerPSOWriteMultFiringPoints ....................................... 18-11

18.1. Introduction

The AerPSO library functions allow Aerotech’s Position Synchronized Output (PSO) or Laser Firing Card to be configured for tracking up to three axes of motion and generating an output signal to fire a laser. This signal may be in several forms. The PSO card will generate a firing pulse output with programmable frequency and duration as well as more sophisticated pulse duration control. In addition, an analog output (+/-10 volts) may be used to generate a voltage that is proportional to velocity, position, or varied as desired by the user.

The four basic steps (some optional) to using the PSO card are:

1. Define firing pulse (or analog output) Refer to AerPSOSetPulse
2. Define firing distance Refer to AerPSOSetFiringDistance
3. Define axes to track Refer to AerPSOSetTracking
   Begin motion Refer to AerMovexxxx
4. Disable firing Refer to AerPSOStopFiring
18.2. AerPSOAllocateFiringTable

AERERR_CODE AerPSOAllocateFiringTable (HAERCTRL hAerCtrl, WORD wPSOCard, WORD wType, DWORD dwNumPoints);

 Declare Function AerPSOAllocateFiringTable Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal wPSOCard As Integer, ByVal wType As Integer, ByVal dwNumPoints As Long) As Long

Parameters

- **hAerCtrl**: Handle to the axis processor card.
- **wPSOCard**: The number of the PSO Card to be addressed.
- **wType**: Indicates Abs/Incr. table positions (see constants below).
- **dwNumPoints**: Number of points in table (<= 400 decimal).

This function implements the CNC PSOD,6 command that allocates memory for a table of firing points and defines them as absolute or incremental distances. This is a precursor to writing the points to the table with the CNC PSOD,1,2 command (AerPSOWriteMultFiringPoints). The table may have a maximum of 400 points. Currently only one board is supported, wPSOCard == 0.

C Language and LabView Constants

- **PSO_TABLE_XXXX**

VB Constants

- **aerPSOTableXXXX**

See Also

- AerPSOFirePulse
- AerPSOSetDigitalOutput
- AerPSOSetFiringDistance
- AerPSOSetMultAnalogOutput
- AerPSOSetPulse
- AerPSOSetTracking
- AerPSOSStopFiring
- AerPSOWriteMultFiringPoints

Example

Samples\Lib\AexPSO\AexPSO.C
18.3. **AerPSOFirePulse**

AERERR_CODE AerPSOFirePulse (HAERCTRL hAerCtrl, WORD wPsoCard, DWORD dwCount);

Declare Function AerPSOFirePulse Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal wPSOCard As Integer, ByVal dwCount As Long) As Long

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hAerCtrl</td>
<td>Handle to the axis processor card.</td>
</tr>
<tr>
<td>wPSOCard</td>
<td>The number of the PSO Card to be addressed.</td>
</tr>
<tr>
<td>dwCount</td>
<td>The number of pulse train sequences to output.</td>
</tr>
<tr>
<td></td>
<td>(0 = indefinitely).</td>
</tr>
</tbody>
</table>

This function implements the CNC PSOF.1 and PSOF.2 commands that activate the output firing pulse train defined by the CNC PSOP command (*AerPSOSetPulse*) for the specified *dwCount* times. If *dwCount* is zero the pulse train will repeat indefinitely. No position tracking occurs in this mode. Currently only one board is supported, *wPSOCard == 0*.

**See Also**

- `AerPSOAllocateFiringTable`
- `AerPSOSetDigitalOutput`
- `AerPSOSetFiringDistance`
- `AerPSOSetMultAnalogOutput`
- `AerPSOSetPulse`
- `AerPSOSetTracking`
- `AerPSOSTopFiring`
- `AerPSOWriteMultFiringPoints`

**Example**

Samples\Lib\AexPSO\AexPSO.C
18.4. AerPSOSetDigitalOutput

AERERR_CODE AerPSOSetDigitalOutput (HAERCTRL hAerCtrl, WORD wPsoCard, DWORD dwMask, DWORD dwData);

Declare Function AerPSOSetDigitalOutput Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal wPsoCard As Integer, ByVal dwMask As Integer, ByVal dwData As Long) As Long

Parameters

- **hAerCtrl**: Handle to the axis processor card.
- **wPsoCard**: The number of the PSO Card to be addressed.
- **dwMask**: Bit mask of the bits to change the state of.
- **dwData**: State of bits to write to PSO digital outputs.

This function will set the specified bits on the PSO card outputs by setting the bits set true (logic 1) in the `dwMask` to the value of the bit specified in the `dwData` word. Bits set to zero in the `dwMask` will not be changed. This implements the CNC PSOT.0 command.

See Also

- AerPSOAllocateFiringTable
- AerPSOFirePulse
- AerPSOSetFiringDistance
- AerPSOSetMultAnalogOutput
- AerPSOSetPulse
- AerPSOSetTracking
- AerPSOSTopFiring
- AerPSOWriteMultFiringPoints

Example

```
Samples\Lib\AexPSO\AexPSO.C
```
18.5. **AerPSOSetFiringDist**

AERERR_CODE AerPSOSetFiringDist (HAERCTRL hAerCtrl, WORD wPsoCard, DWORD dwDist);

Declare Function AerPSOSetFiringDist Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal wPsoCard As Integer, ByVal dwDist As Long) As Long

**Parameters**

- **hAerCtrl** Handle to an Aerotech Control.
- **wPsoCard** The number of the PSO Card to be addressed.
- **dwDist** The incremental firing distance in machine steps.

This function implements the CNC PSOD,0 command that activate the output firing pulse every incremental `dwDist` machine steps. Currently only one board is supported, `wPsoCard == 0`. This command is used in conjunction with the CNC PSOF,3 (AerPSOSetTracking) command.

**See Also**

- AerPSOAllocateFiringTable
- AerPSOFirePulse
- AerPSOSetDigitalOutput
- AerPSOSetMultiAnalogOutput
- AerPSOSetPulse
- AerPSOSetTracking
- AerPSOStopFiring
- AerPSOWriteMultiFiringPoints

**Example**

```
Samples\Lib\AexPSO\AexPSO.C
```
18.6. AerPSOSetMultAnalogOutput

AERERR_CODE AerPSOSetMultAnalogOutput (HAERCTRL hAerCtrl, WORD wPsoCard, WORD wType, WORD wMode, WORD wNumChans, PAER_PSO_D2A pD2A);

**Parameters**

- **hAerCtrl**  
  Handle to the axis processor card.
- **wPsoCard**  
  The number of the PSO Card to be addressed.
- **wType**  
  Always Zero (wType == 0).
- **wMode**  
  Analog output mode.
- **wNumChans**  
  Analog output channel specifier.
- **pD2A**  
  Pointer to structure of type AER_PSO_D2A.

This function allows the 2 analog outputs on a PSO card (currently only one board is supported, wPsoCard == 0) to be set to a value or to begin tracking an axis, as defined by the CNC PSOT, 2, 4, or 6 commands. The DAC is specified by the wNumChans parameter (0-1). The wType parameter is always zero to specify a bipolar DAC.

The wMode parameter specifies the analog output mode (see the constants, shown below).

Setting the DAC to a fixed value (wType = 0) allows the DAC to be set to a fixed value between +/- 10 volts in .3 millivolt increments. wType 1 allows the specified DAC to proportionally track the axes velocity. A minimum DAC voltage may be specified as well as the DAC voltage at a specified target velocity. The velocity is specified in machine steps per millisecond. wType 2 allows the specified DAC to proportionally track the axes position as wType 1 tracks axes velocity, a DAC minimum voltage and DAC voltage at the specified target position. The wNumChans parameter specifies the analog output channel, 0 or 1. The pointer to the AER_PSO_DATA structure provides all other parameters for the function.

**C Language and LabView Constants**

```
PSO_MODE_XXXX
```

**VB Constants**

```
aerPSOModeXXXX
```

See Also

- AerPSOAllocateFiringTable
- AerPSOFirePulse
- AerPSOSetDigitalOutput
- AerPSOSetFiringDistance
- AerPSOSetPulse
- AerPSOSetTracking
- AerPSOSTopFiring
- AerPSOWriteMultFiringPoints

Example

```
Samples\Lib\AexPSO\AexPSO.C
```
18.7. AerPSOSetPulse

AERERR_CODE AerPSOSetPulse (HAERCTRL hAerCtrl, WORD wPsOCard, WORD wType, PVOID pvPulse);

AERERR_CODE AerPSOSetPulseExWidth (HAERCTRL hAerCtrl, WORD wPsOCard, WORD wType, DWORD dwWidth);

AERERR_CODE AerPSOSetPulseExDefine (HAERCTRL hAerCtrl, WORD wPsOCard, DWORD dwLead, DWORD dwWidth, DWORD dwTail);

AERERR_CODE AerPSOSetPulseExRamp (HAERCTRL hAerCtrl, WORD wPsOCard, DWORD dwLead, DWORD dwWidth, DWORD dwTail, WORD wRampTime, WORD wRampInterval);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hAerCtrl</td>
<td>Handle to the axis processor card.</td>
</tr>
<tr>
<td>wPsOCard</td>
<td>The number of the PSO Card to be addressed.</td>
</tr>
<tr>
<td>wType</td>
<td>Indicates type of data in pvPulse (see constants below).</td>
</tr>
<tr>
<td>pvPulse</td>
<td>Pointer to structure used to characterize the firing pulse (AER_PULSE_xxxx)</td>
</tr>
<tr>
<td>dwLead</td>
<td>De-assertion time preceding ramp-up</td>
</tr>
<tr>
<td>dwWidth</td>
<td>Assertion time at ramp completion</td>
</tr>
<tr>
<td>dwTail</td>
<td>De-assertion time following ramp down</td>
</tr>
<tr>
<td>wRampTime</td>
<td>Pulse ramp increment</td>
</tr>
<tr>
<td>wRampInterval</td>
<td>Interval between ramp increments</td>
</tr>
</tbody>
</table>

This function allows the firing pulse of a PSO laser firing card (currently only one board is supported, wPsOCard == 0) to be configured the same as the CNC PSOP.0-2 and 4 commands.

The wType must be one of the PULSE_YYYY (C Language) or aerPulseWidthXXX (VB) constants

The wType 0 and 1 commands allow the pulse output to be defined in milliseconds and microseconds respectively. The wType 2 command allows the pulse lead, width, and trail to be specified in tenths of milliseconds. The wType 3 command is similar to the wType 2 with the addition of specifying the ramp up/down and pulse gap characteristics in tenths of milliseconds. The pulse width begins at the ramp up/down pulse width and increments this pulse width until it reaches the specified pulse width. Once it does, the pulse width begins decreasing at the rate it increased down to the starting pulse width. The pulse gaps may be a fixed width (gap >0) or may ramp up/down to match the size of the pulse width (gap = 0).
For the C language function, the pointer *pvPulse* structure is dependent upon the *wType* parameter. *wType* 0 and 1 use the structure type AER_PULSE_WIDTH, *wType* 2 uses the structure AER_PULSE_DEFINE, and *wType* 3 uses the structure AER_PULSE_DEFINE_RAMP to configure the laser firing pulse output.

**C Language and LabView Constants**

<table>
<thead>
<tr>
<th>Constant Name</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>#define PULSE_WIDTH_MSEC</td>
<td>Define Pulse Width in mSec</td>
<td>0x00</td>
</tr>
<tr>
<td>#define PULSE_WIDTH_USEC</td>
<td>Define Pulse Width in uSec</td>
<td>0x01</td>
</tr>
<tr>
<td>#define PULSE_DEFINE</td>
<td>Define 3 Phase Pulse</td>
<td>0x02</td>
</tr>
<tr>
<td>#define PULSE_DEFINE_RAMP</td>
<td>Define 3 Phase Pulse &amp; Ramp</td>
<td>0x03</td>
</tr>
</tbody>
</table>

**VB Constants**

- `aerPulseWidthXXXX`

**See Also**

- `AerPSOAllocateFiringTable`
- `AerPSOFirePulse`
- `AerPSOSetDigitalOutput`
- `AerPSOSetFiringDistance`
- `AerPSOSetMultAnalogOutput`
- `AerPSOSetTracking`
- `AerPSOStopFiring`
- `AerPSOWriteMultFiringPoints`

**Example**

- Samples\Lib\AexPSO\AexPSO.C
18.8. AerPSOSetTracking

AERERR_CODE AerPSOSetTracking (HAERCTRL hAerCtrl, WORD wPsoCard,
AXISMASK mChannel);

Declare Function AerPSOSetTracking Lib “AERSYS.DLL” (ByVal hAerCtrl As Long,
ByVal wPSOCard As Integer, ByVal mChannel As Long) As Long

Parameters
  hAerCtrl     Handle to the axis processor card.
  wPSOCard     The number of the PSO card to be addressed.
  mChannel     Channel bitmask to define the PSO channel numbers to track for
                firing.

This function implements the CNC PSOF,3 command that allows up to three axes to be
defined for tracking. Currently only one board is supported, wPSOCard == 0. mChannel
defines up to three channels to track with bit 0 representing the first channel.

See Also
  AerPSOAllocateFiringTable
  AerPSOFirePulse
  AerPSOSetDigitalOutput
  AerPSOSetFiringDistance
  AerPSOSetMultAnalogOutput
  AerPSOSetPulse
  AerPSOSStopFiring
  AerPSOWriteMultFiringPoints

Example
  Samples\Lib\AexPSO\AexPSO.C
18.9. **AerPSOSStopFiring**

AERERR_CODE AerPSOSStopFiring (HAERCTRL hAerCtrl, WORD wPsoCard);

Declare Function AerPSOSStopFiring Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal wPSOCard As Integer) As Long

**Parameters**

- **hAerCtrl**: Handle to the axis processor card.
- **wPSOCard**: The number of the PSO Card to be addressed.

This function disables firing and tracking for the selected PSO card similar to the CNC PSOF,0 command. Currently only one board is supported, \( wPSOCard = 0 \).

**See Also**

- AerPSOAllocateFiringTable
- AerPSOFirePulse
- AerPSOSetDigitalOutput
- AerPSOSetFiringDistance
- AerPSOSetMultiAnalogOutput
- AerPSOSetPulse
- AerPSOSetTracking
- AerPSOWriteMultiFiringPoints

**Example**

Samples\Lib\AexPSO\AexPSO.C
18.10. AerPSOWriteMultFiringPoints

AERERR_CODE AerPSOWriteMultFiringPoints (HAERCTRL hAerCtrl, WORD wPSOCard, DWORD dwStart, WORD wNumPoints, PDWORD pdwPoints);

Declare Function AerPSOWriteMultFiringPoints Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal wPSOCard As Integer, ByVal dwStart As Long, ByVal wNumPoints As Integer, ByRef pdwPoints As Long) As Long

Parameters

- **hAerCtrl** Handle to the axis processor card.
- **wPSOCard** The number of the PSO Card to be addressed.
- **dwStart** Starting point in table to write points to.
- **dwNumPoints** Number of points in table (<= 400 decimal).
- **pdwPoints** Pointer to values to write to the table.

This function implements the CNC PSOD,1,2 commands that allow a table of firing points to be created for tracking/firing based upon the axes absolute or incremental distances. The table must have memory allocated by the AerPSOAllocateFiringTable function (PSOD,6 command) and may have a maximum of 400 points. Currently only one board is supported, wPSOCard == 0.

See Also
- AerPSOAllocateFiringTable
- AerPSOFirePulse
- AerPSOSetDigitalOutput
- AerPSOSetFiringDistance
- AerPSOSetMultAnalogOutput
- AerPSOSetPulse
- AerPSOSetTracking
- AerPSOSetStopFiring

Example

Samples\Lib\AexPSO\AexPSO.C

∇ ∇ ∇
CHAPTER 19: REGISTRY FUNCTIONS

In This Section:
- Introduction ................................................................. 19-1
- AerRegGetDefDevice .................................................... 19-3
- AerRegGetDefDeviceInfo .............................................. 19-4
- AerRegGetDeviceInfo .................................................. 19-5
- AerRegGetFileName ...................................................... 19-7
- AerRegQueryCardCount .................................................. 19-8
- AerRegQueryCardList .................................................... 19-9
- AerRegSetDefDevice ..................................................... 19-10
- AerRegSetDeviceInfo .................................................... 19-11
- AerRegSetFileName ....................................................... 19-12

19.1. Introduction

The Windows registry is used by the UNIDEX 600 series controllers to store system information such as interrupt numbers and I/O address. The stored values may be slightly different depending on the operating system. The registry is automatically configured during setup of the U600 software. Aerotech also provides AerReg.exe to add or update current registry settings. The AerReg functions provide a way to set and query registry information about the controller.

Configuration allows for more than one device and one card to be setup in a system. The device is identified by its Device ID (AER_UNIDEX_xxxx constant) and a card number (AER_CARD_xxxx constant). The communications to a device is established based on this Device ID-Card combination (Refer to AerSysOpen). All information is stored in the registry based on this combination. This information is stored differently for Windows 95 and Windows NT. The registry should not be manipulated directly by the user either programmatically (Win32 registry functions) or through the use of Windows Regedit.exe/Regedt32.exe. The AerReg functions and utility have been provided so that a consistent interface is provided and compatibility is guaranteed. Use of Win32 registry functions may not be compatible from device to device, OS to OS, or version to version.

The following table provides a chart for the default values that are used by Aerotech Inc. to configure the various devices. If it is indicated that an item is “set by Aerotech,” then the indicated value is the value that must always be used. The value of the defaults as well as other valid settings can be found in the controller’s hardware manual.
## Table 19-1. Device Default Values

<table>
<thead>
<tr>
<th></th>
<th>C Language</th>
<th>VB</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATWindow</td>
<td>AER_DRV600_DEFAULT_ATWIN</td>
<td>AerDrv600DefaultATWin</td>
</tr>
<tr>
<td>IOBase</td>
<td>AER_DRV600_DEFAULT_IO</td>
<td>AerDrv600DefaultIO</td>
</tr>
<tr>
<td>IRQ</td>
<td>AER_DRV600_DEFAULT_IRQ</td>
<td>AerDrv600DefaultIRQ</td>
</tr>
<tr>
<td>VMEAddress</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>DSC</td>
<td>AER_DRV600_DEFAULT_DSC</td>
<td>AerDrv600DefaultDSC</td>
</tr>
<tr>
<td>BootImage</td>
<td>PC960BT.IMG</td>
<td></td>
</tr>
<tr>
<td>Image</td>
<td>PC960.IMG</td>
<td></td>
</tr>
<tr>
<td>SymbolicName¹</td>
<td>U600.VXD (Win95)</td>
<td>U600 (WinNT)</td>
</tr>
</tbody>
</table>

1. The SymbolicName is used to establish communications through the device driver. Under Windows 95, this is the name of the device driver (with path). Under Windows NT, this is the name exported by the device driver. This value is “U600” (or AER_DEFAULT_SYMBOLIC_NAME) for both the U600/U620 controllers.
19.2. **AerRegGetDefDevice**

AERERR_CODE AerRegGetDefDevice (PDWORD pdwDeviceID, PDWORD pdwCard);

Declare Function AerRegGetDefDevice Lib “AERSYS.DLL” (ByRef pdwDeviceID As Long, ByRef pdwCard As Long) As Long

**Parameters**
- `pdwDeviceID` Address of variable to hold Device identifier (see constants).
- `pdwCard` Address of variable to hold Card identifier (see constants).

This function queries the system registry and retrieves the default device and card identifiers. Refer to *Aerotech Registry* for further details on how the registry is setup and configured.

These values can then be passed to AerSysOpen or any other function that requires a device or card identifier.

**C Language and LabView Constants**
- `AER_DRV600_DEFAULT_XXXX`
- `AER_UNIDEX_XXXX`
- `AER_CARD_XXXX`

**VB Constants**
- `aerDrv600DefaultXXXX`
- `aerCardXXXX`
- `aerDeviceIDXXXX`

**See Also**
- `AerRegSetDefDevice`
19.3.  **AerRegGetDefDeviceInfo**

AERERR_CODE AerRegGetDefDeviceInfo (DWORD dwDeviceID, 
PAER_REG_DEVICE_INFO pInfo);

**Parameters**

| dwDeviceID | Device identifier. (AER_UNIDEX_XXXX constant). |
| pInfo      | Pointer to the variable that will receive AER_REG_DEVICE_INFO. |

This function returns the default information associated with a given device. This provides the user a way to fill in an AER_REG_DEVICE_INFO packet with default values.

**See Also**

AerRegGetDefDevice
AerRegGetDeviceInfo
19.4. **AerRegGetDeviceInfo**

AERERR_CODE AerRegGetDeviceInfo (DWORD dwDeviceID, DWORD dwCard, PAER_REG_DEVICE_INFO pInfo);

AERERR_CODE AerRegGetDeviceInfoEx (DWORD dwDeviceID, DWORD dwCard, LPTSTR pszBootImage, LPTSTR pszImage, LPTSTR pszSymbolicName, PDWORD pdwATWindow, PDWORD pdwIOBase, PDWORD pdwIRQ);

Declare Function AerRegGetDeviceInfoEx Lib “AERSYS.DLL” (ByVal dwDeviceID As Long, ByVal dwCard As Long, ByRef pszBootImage As String, ByRef pszImage As String, ByRef pszSymbolic As String, ByRef dwATWindow As Long, ByRef pdwIOBase As Long, ByRef pdwIRQ As Long) As Long

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dwDeviceID</td>
<td>Device identifier (see constants).</td>
</tr>
<tr>
<td>dwCard</td>
<td>Card identifier. (see constants).</td>
</tr>
<tr>
<td>pszBootImage</td>
<td>String returning the Boot Image filename (*.IMG).</td>
</tr>
<tr>
<td>pszImage</td>
<td>String returning the Main Image filename (*.IMG).</td>
</tr>
<tr>
<td>pszSymbolic</td>
<td>String returning the symbolic name in the registry.</td>
</tr>
<tr>
<td>pdwATWindow</td>
<td>Returns the PC-AT memory window address.</td>
</tr>
<tr>
<td>pdwIOBase</td>
<td>Returns the I/O base address of the UNIDEX 600.</td>
</tr>
<tr>
<td>pdwIRQ</td>
<td>Returns the IRQ number of the UNIDEX 600.</td>
</tr>
<tr>
<td>pInfo</td>
<td>Pointer to the variable that will receive AER_REG_DEVICE_INFO (see AER_REG_DEVICE_INFO in Appendix C: Structures).</td>
</tr>
</tbody>
</table>

All string variables in MS Visual Basic, passed by reference (ByRef), must be declared as fixed length strings within your program, long enough to hold the string value returned by the function. Also, those string variables which are passed, with another parameter indicating the length of the string variable, must also be fixed length strings, otherwise, you would not be able to pass the length of the string. For example, to declare a fixed length string:

```
DIM sGlobStr as STRING * 50 ; 50 characters long
```

This function queries the operating system registry based on the supplied device identifier and retrieves information about how the device is configured. Refer to Aerotech Registry for further details on how the registry is setup and configured.

This information is used to determine the location of the device driver and image files so that communication can be established. This information is primarily used by AerSysOpen and AerSysDownload.
The following “wrapper” functions are also available. They can be used to set a smaller subset of data.

// wrapper only gets base device information
// must call device specific function to get device specific info when using wrapper
AERERR_CODE AerRegGetU600DeviceInfoEx (DWORD dwCard, PDWORD pdwDSC);

Declare Function AerRegGetU600DeviceInfoEx Lib "AERSYS.DLL" (ByVal dwCard As Long, ByRef pdwDSC As Long) As Long

### C Language and LabView Constants

- AER_UNIDEX_XXXX
- AER_CARD_XXXX

### VB Constants

- aerCardXXXX
- aerDeviceIDXXXX

### See Also

- AerRegSetDeviceInfo

### Example

Samples\Lib\VisualBasic\RunPgm.vbp
19.5. AerRegGetFileName

AERERR_CODE AerRegGetFileName (DWORD dwDeviceID, DWORD dwCard,
      DWORD dwRegId, LPTSTR pszFile);

Declare Function AerRegGetFileName Lib “AERSYS.DLL” (ByVal dwDeviceID As
      Long, ByVal dwCard As Long, ByVal dwRegId As Long, ByRef pszFile As String) As Long

Parameters
      dwDeviceID   Device identifier (see constants below).
      dwCard       Card identifier (see constants below).
      dwRegId      File to get from registry (see constants below).
      pszFile      Pointer to string to hold file name.

All string variables in MS Visual Basic, passed by reference (ByRef), must be
declared as fixed length strings within your program, long enough to hold the string
value returned by the function. Also, those string variables which are passed, with
another parameter indicating the length of the string variable, must also be fixed
length strings, otherwise, you would not be able to pass the length of the string. For
example, to declare a fixed length string;

      DIM sGlobStr as STRING * 50 ; 50 characters long

This function gets the name of the desired file from the registry. The filenames are fully
qualified file paths (i.e. c:\u600\ini\AxisParm.ini). The directory names are path names
that can be used to determine where the software has been installed. The installation
program for the U600 libraries and utilities initializes the registry with the install paths.

The UNIDEX 600 MMI and other utilities use this information to determine which files it
needs to use for initializing the system and saving/restoring relative data.

C Language and LabView Constants
      AER_UNIDEX_XXX
      AER_CARD_XXX
      AERREGID_XXX

VB Constants
      aerCardXXX
      aerDeviceIDXXX
      aerRegIDXXX

See Also
      AerRegSetFileName

Example
      Samples\Lib\VisualBasic\RunPgm.vbp
19.6. AerRegQueryCardCount

AERERR_CODE AerRegQueryCardCount (DWORD dwDeviceID, PDWORD pdwCount);

Declare Function AerRegQueryCardCount Lib "AERSYS.DLL" (ByVal dwDeviceID As Long, ByRef pdwCount As Long) As Long

Parameters
  dwDeviceID  Device identifier (see constants below).
  pdwCount    Address of variable to hold number of possible card configurations.

This function queries the system registry and retrieves the number of cards that have been setup for a device. Refer to Aerotech Registry for further details on how the registry is setup and configured.

C Language and LabView Constants
  AER_UNIDEX_XXX

VB Constants
  aerDeviceIDXXX

See Also
  AerRegQueryCardList
19.7. **AerRegQueryCardList**

AERERR_CODE AerRegQueryCardList (DWORD _dwDeviceID_, DWORD _dwCount_, PDWORD _pdwList_);

Declare Function AerRegQueryCardList Lib “AERSYS.DLL” (ByVal _dwDeviceID_ As Long, ByVal _dwCount_ As Long, ByRef _pdwList_ As Long) As Long

**Parameters**

- **_dwDeviceID_** Device identifier (see constants below).
- **_dwCount_** Maximum number of values that _pdwList_ can hold.
- **_pdwList_** Pointer to an array that can hold _dwCount_ card numbers.

This function queries the system registry and retrieves the number of each card that has been setup for a device. Refer to *Aerotech Registry* for further details on how the registry is setup and configured.

It is possible to have a system configured for one or more cards. These cards do not have to be in any particular order. A system can be set up using a single U600 card and be identified as Card3. The values returned in _pdwList_ can then be passed to AerRegGetDeviceInfo to determine how a particular card is configured.

**C Language and LabView Constants**

- **AER_UNIDEX_XXXX**

**VB Constants**

- **aerDeviceIDXXXX**

**See Also**

- **AerRegQueryCardCount**
- **AerRegGetDeviceInfo**
19.8. **AerRegSetDefDevice**

AERERR_CODE AerRegSetDefDevice (DWORD dwDeviceID, DWORD dwCard);

Declare Function AerRegSetDefDevice Lib “AERSYS.DLL” (ByVal dwDeviceID As Long, ByVal dwCard As Long) As Long

**Parameters**

- **dwDeviceID**
  - Device identifier to use as the default value (see constants below).

- **dwCard**
  - Card identifier to use as the default value.

This function adds the default device and card information to the system registry. Refer to *Aerotech Registry* for further details on how the registry is setup and configured.

If the default entry does not exist, it is created. If the entry exists, the old information is overwritten.

**C Language and LabView Constants**

- **AER_UNIDEX_XXXX**

**VB Constants**

- **aerDeviceIDXXXX**

**See Also**

- **AerRegGetDefDevice**
19.9. AerRegSetDeviceInfo

AERERR_CODE AerRegSetDeviceInfo (DWORD dwCard,
   AER_REG_DEVICE_INFO pInfo);

AERERR_CODE AerRegSetDeviceInfoEx (DWORD dwDeviceID, DWORD dwCard,
   LPTSTR psBootImage, LPTSTR psImage, LPTSTR psSymbolic,
   DWORD dwATWindow, DWORD dwIOBase, DWORD dwIRQ);

Declare Function AerRegSetDeviceInfoEx Lib “AERSYS.DLL” (ByVal dwDeviceID As Long, ByVal dwCard As Long, ByVal psBootImage As String, ByVal psImage As String, ByVal psSymbolic As String, ByVal dwATWindow As Long, ByVal dwIOBase As Long, ByVal dwIRQ As Long) As Long

Parameters
   dwDeviceID    Device identifier (see constants below).
   dwCard        Card identifier. (see constants below).
   psBootImage   String defining the Boot Image filename (*.IMG).
   psImage       String defining the Main Image filename (*.IMG).
   psSymbolic    The symbolic name in the registry.
   dwATWindow    Defines the PC-AT memory window address.
   dwIOBase      Defines the I/O base address of the UNIDEX 600.
   dwIRQ         Defines the IRQ number of the UNIDEX 600.
   pInfo         Pointer to a AER_REG DEVICE_INFO structure that contains the device information (see AER_REG_DEVICE_INFO in Appendix C: Structures.

This function adds the device information to the operating system registry based on the card identifier and device identifier specified. Refer to Aerotech Registry for further details on how the registry is setup and configured.

If the entry for the device does not exist, it is created. If the entry exists, the previous information is overwritten.

The following wrapper functions are also available.

// wrapper only sets base device information

// must call device specific function to setup device specific info when using wrapper

AERERR_CODE AerRegSetU600DeviceInfoEx (DWORD dwCard, DWORD dwDSC);

Declare Function AerRegSetU600DeviceInfoEx Lib “AERSYS.DLL” (ByVal dwCard As Long, ByVal dwDSC As Long) As Long

C Language and LabView Constants
   AER_UNIDEX_XXXX
   AER_CARD_XXXX

VB Constants
   aerCardXXXX
   aerDeviceIDXXXX

See Also
   AerRegGetDeviceInfo
19.10. AerRegSetFileName

AERERR_CODE AerRegSetFileName (DWORD dwDeviceID, DWORD dwCard, DWORD dwRegId, LPCTSTR pszFile);

Declare Function AerRegSetFileName Lib “AERSYS.DLL” (ByVal dwDeviceID As Long, ByVal dwCard As Long, ByVal dwRegId As Long, ByVal pszFile As String) As Long

Parameters

- dwDeviceID: Device identifier (see constants below).
- dwCard: Card identifier (see constants below).
- dwRegId: File to get from registry (see constants below).
- pszFile: Pointer to file name.

This function sets the name of the desired file from the registry. The filenames are fully qualified file paths (i.e., c:\u600\ini\AxisParm.ini). The directory names are path names that can be used to determine where the software has been installed. The installation program for the U600 libraries and utilities initializes the registry with the install paths.

The UNIDEX 600 MMI and other utilities use this information to determine which files it needs to use for initializing the system and saving/restoring relative data.

C Language and LabView Constants

- AER_UNIDEX_XXXX
- AER_CARD_XXXX
- AERREGID_XXXX

VB Constants

- aerCardXXXX
- aerDeviceIDXXXX
- aerRegIDXXXX

See Also

AerRegGetFileName
CHAPTER 20:  STRIP FUNCTIONS

20.1. Introduction

The AerStrip functions provide an interface for single or multiple axis data collection similar to the AerTune and AerPlot utilities. The user can collect data on position/velocity (actual, commanded, or error) plus acceleration or torque. Data is collected at user specified, evenly spaced intervals of time. The collection will not effect the speed of the operation of the servo-loop. It could however, effect the rate of operation of background tasks such as the CNC tasks. The strip chart functions can require significant amounts of axis processor memory depending on the number of samples and axes requested. The data is stored in the axis processor during the collection process and can later be uploaded to the host for display upon completion.

Data collection can be started immediately, or can be triggered to start at a given position or velocity. There several other miscellaneous trigger conditions that can be defined (see AerStripSetTrigger or AerStripGlobalSetTrigger).

There are two kinds of strip charting: “axis” and “global.” These differ in three important ways. First, the trigger conditions are different between the two modes. Second, the axis strip charting is performed on a single axis basis, while the global charting can be done on multiple axes simultaneously. Finally, the regular strip charting collects only position, position command, accel, and torque data. The global strip charting, in addition, collects command and actual velocity, acceleration, and master position.

Global and axis strip charting can be performed independently because they use different parts of the axis processor memory for data storage. The AerStripGlobalxxxx functions perform global charting, while all other AerStripxxxx functions perform axis charting.

There are five steps to collecting data with the strip chart functions:

\[
\begin{align*}
&\text{AerStripAllocate} & \text{; Allocate controller memory to hold data} \\
&\text{AerStripSetTrigger} & \text{; Define trigger conditions, number of samples, etc.} \\
&\text{While (AerStripGetStatus != STRIP_STATUS_DONE)} & \text{; Wait for completion} \\
&\text{AerStripGetSample} & \text{; Read data from controller} \\
&\text{AerStripFree} & \text{; Free memory allocated on controller}
\end{align*}
\]
20.2. **AerStripAllocate**

AERERR_CODE AerStripAllocate (HAERCTRL hAerCtrl, AXISINDEX iAxis, WORD wSize);

Declare Function AerStripAllocate Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal iAxis As Long, ByVal wSize As Integer) As Long

**Parameters**
- **hAerCtrl** Handle to the controller.
- **iAxis** Index of a physical axis (see constants below).
- **wSize** Number of samples to be allocated.

This function allocates memory on the controller for storing a strip chart. Each sample uses 14 bytes of memory. If the axis processor fails to allocate the space, a programming error occurs.

**C Language and LabView Constants**

*AXISINDEX_XXXX*

**VB Constants**

*aerAxisIndex#*

**See Also**

*AerStripGlobalAllocate*
*AerStripFree*

**Example**

Samples\Lib\AexStrip.C
20.3. AerStripFree

AERERR_CODE AerStripFree (HAERCTRL hAerCtrl, AXISINDEX iAxis);

Declare Function AerStripFree Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal iAxis As Long) As Long

Parameters

    hAerCtrl Handle to the controller.
    iAxis Index of a physical axis (see constants below).

This function frees the memory allocated by a previous AerStripAllocate command. This function does nothing if no strip is allocated. If triggering is currently active or armed, then collection is halted first.

C Language and LabView Constants

    AXISINDEX_XXXX

VB Constants

    aerAxisIndex#

See Also

    AerStripAllocate

Example

    Samples\Lib\AexStrip.C
20.4. AerStripGetIOPosLatchStatus

AERERR_CODE AerStripGetIOPosLatchStatus (HAERCTRL hAerCtrl, AXISINDEX iAxis, PWORD pwMode, PWORD pwType, PWORD pwBit, PWORD pwLevel);

Declare Function AerStripGetIOPosLatchStatus Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal iAxis As Long, ByRef pwMode As Integer, ByRef pwType As Integer, ByRef pwBit As Integer, ByRef pwLevel As Integer) As Long

Parameters
- hAerCtrl Handle to the processor card.
- iAxis Index of a physical axis (see constants below).
- pwMode Pointer to receive Trigger mode (see constants below).
- pwType Pointer to receive Type of I/O to trigger on (see constants below).
- pwBit Pointer to receive Bit number (0 is the first bit) to trigger on.
- pwLevel Pointer to receive Level of input to trigger on (0 or 1).

This function gets the status of the high-speed position latch.

| At the present time the returned value, pwBit, may not necessarily be equal to the wBit set in the AerStripSetIOPosLatch command. |

C Language and LabView Constants
- AXISINDEX_XXXX
- STRIP_IOPOSLATCH_XXXX

VB Constants
- aerAxisIndex#
- aerStripIOPosLatchXXXX

See Also
- AerStripSetIOPosLatch
20.5. AerStripGetSample

AERERR_CODE AerStripGetSample (HAERCTRL hAerCtrl, AXISINDEX iAxis,
  WORD wStart, WORD wCount, PAER_STRIP_SAMPLE pData);

Parameters
  hAerCtrl Handle to the controller.
  iAxis Index of a physical axis (use the AXISINDEX_xxxx constants).
  wStart First point of samples to get (0 is first point).
  wCount Number of samples to get.
  pData Pointer to array of AER_STRIP_SAMPLE structures, to receive the data.

This function returns the collected data starting at the specified position in the strip chart. More than one sample may be read at a time. The values are put into the array pointed at by pData. It is the user’s responsibility to insure that enough space is allocated in pData to hold all the points. Refer to Appendix C: Structures under AER_STRIP_SAMPLE.

See Also
  AerStripAllocate

Example
  Samples\Lib\AexStrip.C
20.6. AerStripGetStatus

AERERR_CODE AerStripGetStatus( HAERCTRL hAerCtrl, AXISINDEX iAxis, PWORD pwStatus, PWORD pwAllocated, PWORD pwSize, PWORD pwCollected );

Declare Function AerStripGetStatus Lib "AERSYS.DLL" ( ByVal hAerCtrl As Long, ByVal iAxis As Long, ByRef pwStatus As Integer, ByRef pwAllocated As Integer, ByRef pwSize As Integer, ByRef pwCollected As Integer ) As Long

Parameters

hAerCtrl        Handle to the controller.
iAxis           Index of physical axis (see constants below).
pwStatus        Pointer to receive the status (see constants below).
pwAllocated     Pointer to receive number of points allocated.
pwSize          Pointer to receive number of points collected last.
pwCollected     Pointer to receive number of points collected so far.

This function returns data concerning the current data collection. Status is a bit-wise word defining the current state of the data collection.

If the status returned is 0, then no data collection array has been allocated (see AerStripAllocate). Otherwise, the status will be a mask of STRIP_STATUS_XXXX constants. The pwSize parameter reflects how many points were specified to be collected by the last call to AerStripSetTrigger. The pwCollected parameter refers to how many points were collected. pwCollected is reset to zero when AerStripSetTrigger is called and will begin increasing to pwSize when data collection is triggered.

C Language and LabView Constants

AXISINDEX_XXXX
STRIP_STATUS_XXXX

VB Constants

aerAxisIndex#
aerStripStatusXXXX

See Also
AerStripSetTrigger
AerStripAllocate

Example
Samples\Lib\AexStrip.C
20.7. AerStripGlobalAllocate

AERERR_CODE AerStripGlobalAllocate (HAERCTRL hAerCtrl, AXISMASK mAxis, WORD wSize);

Declare Function AerStripGlobalAllocate Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal mAxis As Double, ByVal wSize As Integer) As Long

Parameters

   hAerCtrl  Handle to the controller.
   mAxis     A mask of axes.
   wSize     Number of global samples to be allocated.

This function allocates memory on the controller for storing a global strip chart. Each global sample occupies \((n \times 24 + 24)\) bytes of memory, where \(n\) is the number of axes from which data is being collected. \(mAxis\) is the mask of the axis that the user will be collecting data on. If the axis processor cannot allocate sufficient space, a programming error is generated.

C Language and LabView Constants

   AXISMASK_#

VB Constants

   aerAxisMask#

See Also

   AerStripGlobalSetTrigger
   AerStripGlobalGetSample
   AerStripGlobalFree

Example

   Samples\Lib\AexStrip.C
20.8. AerStripGlobalAllocateTrigger

AERERR_CODE AerStripGlobalAllocateTrigger (HAERCTRL hAerCtrl, WORD wSize);

Declare Function AerStripGlobalAllocateTrigger Lib "AERSYS.DLL" (ByVal hAerCtrl
As Long, ByVal wSize As Integer) As Long

Parameters

  hAerCtrl   Handle to the controller.
  wSize      Number of trigger points to be allocated.

This function allocates memory on the controller for storing axis positions that will be
used as a trigger to begin data collection. This command is only used for position based
triggering modes. Each point occupies 8 bytes of storage. If the axis processor fails to
allocate memory, a programming error is generated.

See Also

  AerStripGlobalSetTrigger
  AerStripGlobalGetSample

Example

  Samples\Lib\AexStrip.C
20.9. AerStripGlobalFree

AERERR_CODE AerStripGlobalFree (HAERCTRL hAerCtrl);

Declare Function AerStripGlobalFree Lib “AERSYS.DLL” (ByVal hAerCtrl As Long)
As Long

Parameters

hAerCtrl Handle to the controller.

AerStripGlobalFree frees the memory allocated by a previous AerStripGlobalAllocate call. This function does nothing if no strip is allocated. If triggering is currently active or armed, then collection is halted first.

See Also

AerStripGlobalAllocate

Example

Samples\Lib\AexStrip.C
20.10. AerStripGlobalFreeTrigger

AERERR_CODE AerStripGlobalFreeTrigger (HAERCTRL hAerCtrl);

Declare Function AerStripGlobalFreeTrigger Lib "AERSYS.DLL" (ByVal hAerCtrl As Long) As Long

Parameters

  hAerCtrl   Handle to the controller.

This function is used to free memory used by the trigger table.

See Also

  AerStripGlobalAllocate

Example

  Samples\Lib\AexStrip.C
20.11. AerStripGlobalGetImmediate

AERERR_CODE AerStripGlobalGetImmediate (HAERCTRL hAerCtrl, AXISMASK mAxis, PAER_GSTRIP_SAMPLE pData, DWORD dwParm1, PDWORD pStatus1, DWORD dwParm2, PDWORD pStatus2);

Parameters

- **hAerCtrl**: Handle to the processor card.
- **mAxis**: Axis mask (use the AXISMASK_# constants).
- **pData**: Pointer to an AER_GSTRIP_SAMPLE structure to receive the data.
- **dwParm1**: Axis Parameter number (use AXISPARM_# constants).
- **pStatus1**: Pointer to an array containing the value of dwParm1 for all the desired axes specified by mAxis.
- **dwParm2**: Axis Parameter number (use AXISPARM_# constants).
- **pStatus2**: Pointer to an array containing the value of dwParm2 for all the desired axes specified by mAxis.

This function is similar in operation to AerStripGlobalGetQueue and AerStripGlobalGetQueueRecent except that it only returns the global strip data for the most recently completed sample. This function also returns the values of two axis parameters for all the selected axes.

See Also

- AerStripGlobalGetQueue
- AerStripGlobalGetQueueRecent
20.12. AerStripGlobalGetQueue

AERERR_CODE AerStripGlobalGetQueue (HAERCTRL hAerCtrl, AXISMASK mAxis, WORD wReq, PAER_GSTRIP_SAMPLE pData, PWORD pwRec);

Parameters

- **hAerCtrl**: Handle to the controller.
- **mAxis**: A mask of axes (use the AXISMASK_# constants).
- **wReq**: Number of points requested (no more than 40).
- **pData**: Pointer to an AER_GSTRIP_SAMPLE structure, to receive the data.
- **pwRec**: Pointer to WORD to receive number of points actually delivered.

This function is the analog of AerStripGlobalGetSample when the data collection has been triggered in queue mode. It returns the pwRec number of points, oldest first, that are in the circular queue maintained in the strip chart.

- **pwRec** can be less than **wReq** when the number of points in the queue is less than the number of points requested.

The mAxis mask passed here must be the same as the mask passed in the AerStripGlobalAllocate call.

The user cannot use this function when the trigger mode was non-queue. For this case, see AerStripGlobalGetSample.

See Also

- AerStripGlobalGetSample
- AerStripGlobalSetTrigger

Example

Samples\Lib\AexStrip.C
20.13. AerStripGlobalGetQueueDecimate

AERERR_CODE AerStripGlobalGetQueueDecimate (HAERCTRL hAerCtrl,
AXISMASK mAxis, WORD wUserNumReq,
PAER_GSTRIP_SAMPLE pData, PWORD pwNumRec, WORD
wStep, WORD wType);

Parameters

  hAerCtrl    Handle to the processor card.
  mAxis      Axis mask (use the AXISMASK_# constants).
  wUserNumReq Number of points requested (always a positive number).
  pData      Pointer to an AER_GSTRIP_SAMPLE structure to receive the data.
  pwNumRec   Pointer to WORD to receive the number of points actually delivered.
  wStep      Step size to step through the sampled data (get every nth point).
  wType      Get the oldest or most recent data (use the STRIPGLOBAL_ffff
             constants).

This function is similar in operation to AerStripGlobalGetQueue
and AerStripGlobalGetQueueRecent. However, it returns every (wStep)th point. This
function can either get the wUserNumReq points from the oldest data or the most recent
data from the queue.

Operation of this function can produce unexpected results if (wUserNumReq *
wStep) ≥ total number of points in the queue. In addition, when this function is
evoked in the “RECENT” mode, this function does not affect the pointers in the
queue. Therefore, two consecutive function calls may output overlapping data.

AERERR_CODE AerStripGlobalGetQueueRecent (HAERCTRL hAerCtrl, AXISMASK mAxis, WORD wUserNumReq, PAER_GSTRIP_SAMPLE pData, PWORD pwNumRec);

Parameters

  hAerCtrl  Handle to the processor card.
  mAxis     Axis mask (use the AXISMASK_# constants).
  wUserNumReq Number of points requested (always a positive number).
  pData     Pointer to an AER_GSTRIP_SAMPLE structure to receive the data.
  pwNumRec  Pointer to WORD to receive the number of points actually delivered.

This function is similar in operation to AerStripGlobalGetQueue. It returns the pwNumRec number of points starting from pwNumRec points before the most recent sample, up to the most recent sample from the data stored in the circular queue.

This function does not affect the pointers in the queue. Therefore, two consecutive function calls may output overlapping data.

See Also

   AerStripGlobalGetQueue
20.15. AerStripGlobalGetSample

AERERR_CODE AerStripGlobalGetSample (HAERCTRL hAerCtrl, WORD wFirst,
       WORD wCount, AXISMASK mAxis, PAER_GSTRIP_SAMPLE pData);

Parameters

- **hAerCtrl**: Handle to the controller.
- **wStart**: First point of samples to collect (0 is first point).
- **wCount**: Number of samples to get.
- **mAxis**: A mask of axes (use the AXISMASK_# constants).
- **pData**: Pointer to array of AER_STRIP_SAMPLE structures, to receive the data.

This function returns the values collected, starting at the specified position in the strip chart. More than one sample may be read at a time and the values are put into the structure pointed to by *pData*. This structure, AER_GSTRIP_SAMPLE, consists of pointers to other structures, one to receive the global data and the other is an array of pointers, each pointing to the data for one axis. Refer to Appendix C: Structures, under AER_GSTRIP_SYSTEM_DATA and AER_GSTRIP_AXIS_DATA for a complete description of the system and axis data collected. It is the user’s responsibility to ensure that enough memory is allocated in the structures pointed to by *pData*.

If the user passes a zero mAxis mask, no axis data will be retrieved. Otherwise, it represents a mask of the axes to collect data on.

The user cannot use this function to retrieve points when the trigger was queue mode or QUEUE_HOLD mode. Use AerStripGlobalGetQueue to get points that have been triggered in queue mode.

See Also

- AerStripGlobalSetSample
- AerStripGlobalGetQueue

Example

Samples\Lib\AexStrip.C
20.16. AerStripGlobalGetStatus

AERERR_CODE AerStripGlobalGetStatus (HAERCTRL hAerCtrl, PWORD pwStatus,
WORD pwAllocated, WORD pwSize, WORD pwCollected,
PAXISMASK pmAxis);

Declare Function AerStripGlobalGetStatus Lib “AERSYS.DLL” (ByVal hAerCtrl As
Long, ByRef pwStatus As Integer, ByRef pwAllocated As Integer,
ByRef pwSize As Integer, ByRef pwCollected As Integer, ByRef
pmAxis As Double) As Long

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hAerCtrl</td>
<td>Handle to the controller.</td>
</tr>
<tr>
<td>pwStatus</td>
<td>Pointer to receive the status of the global strip chart (see constants below).</td>
</tr>
<tr>
<td>pwAllocated</td>
<td>Pointer to word to receive the number of points allocated.</td>
</tr>
<tr>
<td>pwSize</td>
<td>Pointer to receive the number of samples to capture.</td>
</tr>
<tr>
<td>pwCollected</td>
<td>Pointer to receive the number of samples currently stored.</td>
</tr>
<tr>
<td>pmAxis</td>
<td>Pointer to receive the mask of the axes whose data is/will be captured (see constants below).</td>
</tr>
</tbody>
</table>

The AerStripGlobalGetStatus function obtains global strip chart status information. If the
status returned is zero, then no storage has been allocated yet (see AerGlobalAllocateStrip).
Otherwise, the status is a mask of the constants (below).

The pwAllocated parameter returns the current number of points allocated. pwSize indicates
the number of data points to be collected. pwCollected returns the number of points collected
at this time and pmAxis indicates which axes are being sampled.

C Language and LabView Constants

AXISMASK_#

STRIPGLOBAL_STATUS_XXXX

VB Constants

aerAxisMask#

aerGStripStatusXXXX

See Also

AerStripGlobalSetTrigger
AerStripGlobalGetSample
AerGlobalAllocateStrip

Example

Samples\Lib\AexStrip.C
20.17. AerStripGlobalGetTriggerPoint

AERERR_CODE AerStripGlobalGetTriggerPoint (HAERCTRL hAerCtrl, WORD wPoint, PLONG plData, PWORD pwStatus);

Declare Function AerStripGlobalGetTriggerPoint Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal wPoint As Integer, ByRef plData As Long, ByRef pwStatus As Integer) As Long

Parameters

hAerCtrl Handle to the controller.
wPoint An entry number in trigger list (0 is first entry).
plData Pointer to long word to receive axis trigger position.
pwStatus Pointer to receive the status of global trigger points.

This function obtains a trigger point that was previously set by an AerStripGlobalSetTrigger function call. The status returned is not the strip status, but the status of the trigger point. Currently, this status is not used and is always 1.

See Also

AerStripGlobalSetTrigger
AerStripGlobalAllocateTrigger

Example

Samples\Lib\AexStrip.C
20.18. AerStripGlobalGetTriggerStatus

AERERR_CODE AerStripGlobalGetTriggerStatus (HAERCTRL hAerCtrl, PWORD pwStatus, PWORD pwSize);

Declare Function AerStripGlobalGetTriggerStatus Lib "AERSYS.DLL" (ByVal hAerCtrl As Long, ByRef pwStatus As Integer, ByRef pwSize As Integer ) As Long

Parameters

hAerCtrl Handle to the controller.
pwStatus Pointer to receive the status of global trigger points (see constants below).
pwSize Pointer to receive the number of samples to capture.

This function obtains status information on the current global strip chart. The wStatus parameter returns the strip status (the same status value as returned by AerStripGlobalGetStatus). wSize returns the number of trigger points allocated.

C Language and LabView Constants

STRIPGLOBAL_STATUS_XXXX

VB Constants

aerGStripStatusXXXX

See Also

AerStripGlobalGetTriggerPoint
AerStripGlobalGetStatus

Example

Samples\Lib\AexStrip.C
20.19. AerStripGlobalHalt

AERERR_CODE AerStripGlobalHalt (HAERCTRL hAerCtrl);
Declare Function AerStripGlobalHalt Lib “AERSYS.DLL” (ByVal hAerCtrl As Long)
As Long

Parameters
  hAerCtrl  Handle to the controller.

Terminates global data capture. The user cannot restart data collection after halting it. To
restart collection the user must reissue an AerStripGlobalSetTrigger.

See Also
  AerStripGlobalRelease

Example
  Samples\Lib\AexStrip.C
20.20. AerStripGlobalHold

AERERR_CODE AerStripGlobalHold (HAERCTRL hAerCtrl);

Declare Function AerStripGlobalHold Lib "AERSYS.DLL" (ByVal hAerCtrl As Long)
As Long

Parameters
  hAerCtrl  Handle to the controller.

This function is used to suspend data capture mode until a subsequent
AerStripGlobalRelease is called.

See Also
  AerStripGlobalRelease

Example
  Samples\Lib\AexStrip.C
20.21. AerStripGlobalRelease

AERERR_CODE AerStripGlobalRelease (HAERCTRL hAerCtrl);

Declare Function AerStripGlobalRelease Lib “AERSYS.DLL” (ByVal hAerCtrl As Long) As Long

Parameters
   hAerCtrl   Handle to the controller.

This function is used to resume the data capture mode previously suspended using AerStripGlobalHold.

See Also
   AerStripGlobalHold

Example
   Samples\Lib\AexStrip.C
20.22. AerStripGlobalSetTrigger

AERERR_CODE AerStripGlobalSetTrigger (HAERCTRL hAerCtrl, WORD wMode, WORD wTime, WORD wSize, LONG lParm1, LONG lParm2);

Declare Function AerStripGlobalSetTrigger Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal wMode As Integer, ByVal wTime As Integer ByVal wSize As Integer, ByVal lParm1 As Long, ByVal lParm2 As Long) As Long

Parameters

- hAerCtrl: Handle to the controller.
- wMode: Trigger mode (use constants, see below).
- wTime: Time between samples (units are in milliseconds).
- wSize: Number of samples to collect (this trigger).
- lParm1: Additional trigger data (see below).
- lParm2: Additional trigger data (see below).

This function “arms,” or sets the strip chart operation to begin when the given condition is set. The mode determines the condition. Possible mode values are listed below.

In STRIPGLOBAL_MODE_TIME (C Language) / aerGStripModeTime (VB) mode, data collection starts immediately.

In STRIPGLOBAL_MODE_POSITION (C) / aerGStripModePosition (VB) mode, the sampling is not performed at regular time intervals, rather, a sample is collected every time the position of a given axis crosses over a given trigger value. The user must form a list of monotonically increasing trigger values prior to initiating this mode (refer to AerStripGlobalSetTriggerPoint). The sample is triggered when the position crosses into the area covered by the trigger list. Every time the position crosses over a value in the trigger list, regardless of the direction of movement, a data point is collected. Here lParm1 is an axis mask that must have only one axis represented in it, representing the axis whose position should be watched. For STRIPGLOBAL_MODE_POSITION (C) / aerGStripModePosition (VB) mode, a trigger table must have been previously allocated via an AerStripGlobalAllocateTrigger call.

Mode STRIPGLOBAL_MODE_QUEUE (C) / aerGStripModeQueue (VB) is used for continuous data collection. The axis processor will continue collecting data until an AerStripGlobalHold is performed. The axis processor will treat the allocated table space as a circular queue. When it fills the strip memory, it resets to the first point and continues collecting. With queue mode, when the user gets “n” samples, the user gets the last “n” samples.

Mode STRIPGLOBAL_MODE_QUEUE_HOLD (C) / aerGStripModeQueueHold (VB) is the same as STRIPGLOBAL_MODE_QUEUE (C) / aerGStripModeQueue (VB) mode, but it starts “held,” meaning data collection will only start when an explicit AerStripGlobalRelease is performed.

Mode STRIPGLOBAL_MODE_IO (C) / aerGStripModeIO (VB) is the same as mode STRIPGLOBAL_MODE_TIME (C) / aerGStripModeTime (VB), except that collection is started only when a particular binary input is set. Pass the virtual I/O bit number in lParm1.
This function does nothing if the strip chart is already triggered or armed.

C Language and LabView Constants

- STRIPGLOBAL_MODE_XXXX
- AXISMASK_#

VB Constants

- aerGStripModeXXX
- aerAxisMask#

See Also

- AerStripGlobalGetTrigger

Example

- Samples\Lib\AexStrip.C
20.23. AerStripGlobalSetTriggerPoint

AERERR_CODE AerStripGlobalSetTriggerPoint (HAERCTRL hAerCtrl, WORD wPoint, LONG lData);

Declare Function AerStripGlobalSetTriggerPoint Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal wPoint As Integer, ByVal lData As Long) As Long

Parameters

- hAerCtrl Handle to the controller.
- wPoint An entry number to table.
- lData Axis trigger position.

This function is used to enter position data in the trigger table allocated by AerStripGlobalAllocateTrigger. The wPoint parameter specifies the entry number in the table. The lData parameter specifies the axis trigger position. This function is only used if the trigger mode (see AerStripGlobalSetTrigger) is STRIPGLOBAL_MODE_POSITION (C) / aerGStripModePos (VB).

See Also

AerStripGlobalAllocateTrigger
AerStripGlobalSetTrigger

Example

Samples\Lib\AexStrip.C
20.24. AerStripHalt

AERERR_CODE AerStripHalt (HAERCTRL hAerCtrl, AXISINDEX iAxis);

Declare Function AerStripHalt Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal iAxis As Long) As Long

Parameters
    hAerCtrl  Handle to the controller.
    iAxis     Index of a physical axis (see constants below).

This function is used to terminate data capture. After halting, the status indicates that the “done” state (constants shown below) AerStripSetTrigger can be used to restart data capture.

C Language and LabView Constants
    STRIP_STATUS DONE
    AXISINDEX_XXXX

VB Constants
    aerStripDone
    aerAxisIndex#

See Also
    AerStripSetTrigger
    AerStripGetSample
    AerStripGetStatus

Example
    Samples\Lib\AexStrip.C
20.25. AerStripSetIOPosLatch

AERERR_CODE AerStripSetIOPosLatch (HAERCTRL hAerCtrl, AXISINDEX iAxis, WORD wMode, WORD wType, WORD wBit, WORD wLevel);

Declare Function AerStripSetIOPosLatch Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal iAxis As Long, ByVal wMode As Integer, ByVal wType As Integer, ByVal wBit As Integer, ByVal wLevel As Integer) As Long

Parameters

- hAerCtrl Handle to the processor card.
- iAxis Index of a physical axis (see constants below).
- wMode Trigger mode (see constants below).
- wType Type of I/O to trigger on (see constants below).
- wBit Bit number (0 is the first bit) to trigger on (see constants below).
- wLevel Level of input to trigger on (0 or 1).

This function arms the position latch. The latch is triggered from the status of either the axis I/O inputs (CW limit, CCW limit, Home, etc.) or one of the Virtual I/O digital inputs (Binary Input [BI], 0 through 511). When the trigger condition is met, this function will latch the current position of the axis into the HOMESWITCHPOS axis parameter. The latch may be set up in two configurations: one shot mode or continuous mode. In one shot mode, once the latch is triggered the mode is then set to STRIP_IOPOSLATCH_DISABLE to disable further triggers. In continuous mode, the function will continue placing the current position into the HOMESWITCHPOS axis parameter as long as the trigger condition is met.

C Language and LabView Constants

AXISINDEX_XXXX
See Also Table 20-1

VB Constants

aerAxisIndex#
See Also Table 20-1
Table 20-1. STRIP_IOPOSLATCHXXX (aerStripIOPosLatchXXX) Constants

The trigger mode (wMode) allows the following options:

<table>
<thead>
<tr>
<th>C Language</th>
<th>Description</th>
<th>VB</th>
</tr>
</thead>
<tbody>
<tr>
<td>STRIP_IOPOSLATCH_DISABLE</td>
<td>I/O position latch disable</td>
<td>aerStripIOPosLatchDisable</td>
</tr>
<tr>
<td>STRIP_IOPOSLATCH_ONESHOT</td>
<td>One shot operation</td>
<td>aerStripIOPosLatchOneshot</td>
</tr>
<tr>
<td>STRIP_IOPOSLATCH_CONTINUOUS</td>
<td>Continuous trigger</td>
<td>aerStripIOPosLatchContinuous</td>
</tr>
</tbody>
</table>

The type of I/O to trigger (wType) has the following choices:

<table>
<thead>
<tr>
<th>C Language</th>
<th>Description</th>
<th>VB</th>
</tr>
</thead>
<tbody>
<tr>
<td>STRIP_IOPOSLATCH_AXISIO</td>
<td>Use axis I/O</td>
<td>aerStripIOPosLatchAxisIO</td>
</tr>
<tr>
<td>STRIP_IOPOSLATCH_VIRTUALIO</td>
<td>Use virtual I/O bit</td>
<td>aerStripIOPosLatchVirtualIO</td>
</tr>
</tbody>
</table>

For axis I/O bit number, the following constants for (wBit) may be used:

<table>
<thead>
<tr>
<th>C Language</th>
<th>Description</th>
<th>VB</th>
</tr>
</thead>
<tbody>
<tr>
<td>STRIP_IOPOSLATCH_BITCCW</td>
<td>Use CCW input</td>
<td>aerStripIOPosLatchBitCCW</td>
</tr>
<tr>
<td>STRIP_IOPOSLATCH_BITCW</td>
<td>Use CW input</td>
<td>aerStripIOPosLatchBitCW</td>
</tr>
<tr>
<td>STRIP_IOPOSLATCH_BITHOME</td>
<td>Use home input</td>
<td>aerStripIOPosLatchBitHome</td>
</tr>
<tr>
<td>STRIP_IOPOSLATCH_BITENCFLT</td>
<td>Use encoder fault input</td>
<td>aerStripIOPosLatchBitEncFLT</td>
</tr>
<tr>
<td>STRIP_IOPOSLATCH_BITFLT</td>
<td>Use drive fault input</td>
<td>aerStripIOPosLatchBitFLT</td>
</tr>
<tr>
<td>STRIP_IOPOSLATCH_BITHALLA</td>
<td>Use Hall A input</td>
<td>aerStripIOPosLatchBitHallA</td>
</tr>
<tr>
<td>STRIP_IOPOSLATCH_BITHALLB</td>
<td>Use Hall B input</td>
<td>aerStripIOPosLatchBitHallB</td>
</tr>
<tr>
<td>STRIP_IOPOSLATCH_BITHALLC</td>
<td>Use Hall C input</td>
<td>aerStripIOPosLatchBitHallC</td>
</tr>
</tbody>
</table>

For Virtual I/O just enter the number of the Virtual I/O input for wBit (0 through 511)
20.26. AerStripSetTrigger

AERERR_CODE AerStripSetTrigger (HAERCTRL hAerCtrl, AXISINDEX iAxis, WORD wMode, WORD wTime, WORD wSize, LONG lParm1, LONG lParm2);

Declare Function AerStripSetTrigger Lib "AERSYS.DLL" (ByVal hAerCtrl As Long, ByVal iAxis As Long, ByVal wMode As Integer, ByVal wTime As Integer, ByVal wSize As Integer, ByVal lParm1 As Long, ByVal lParm2 As Long) As Long

Parameters

- **hAerCtrl** Handle to the controller.
- **iAxis** Index of a physical axis (see constants below).
- **wMode** Trigger mode (see constants below).
- **wTime** Time between samples (units are in milliseconds).
- **wSize** Number of samples to collect (this trigger).
- **lParm1** Additional trigger data (used in STRIP_TRIGGER_MASTER_POS mode and STRIP_TRIGGER_TORQUE mode only).
- **lParm2** Additional trigger data (used in STRIP_TRIGGER_MASTER_POS mode).

This function “arms,” or sets the strip chart operation to begin when the given condition is set. The mode determines the condition. Valid modes are listed below.

In STRIP_TRIGGER_IMMEDIATE mode, a data collection starts immediately.

If in STRIP_TRIGGER_MASTER_POS mode and lParm2 is zero, then the collection will start when the master position of iAxis is below the position specified in lParm1. If in STRIP_TRIGGER_MASTER_POS mode and lParm2 is not zero, collection will start when the master position of iAxis exceeds the value in lParm1.

In STRIP_TRIGGER_POINT0 mode, the triggering is synchronized with cam table or profile points. The collection is triggered when the cam or profile table pointer moves off of the first point.

STRIP_TRIGGER_TORQUE mode will start collection when the torque exceeds the given value (in lParm1). Note that lParm1 is interpreted as a signed value. Torque units are defined by the IMAX axis parameter.

STRIP_TRIGGER_EXTERNAL mode is for triggering from within the application program (see AerStripTrigger).

This function does nothing if the strip chart is already triggered or armed.

C Language and LabView Constants

<table>
<thead>
<tr>
<th>Axis Index</th>
<th>Trigger Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>AXISINDEX_XXXX</td>
<td>STRIP_TRIGGER_XXXX</td>
</tr>
</tbody>
</table>

VB Constants

<table>
<thead>
<tr>
<th>#constant</th>
<th>#constant</th>
</tr>
</thead>
<tbody>
<tr>
<td>aerAxisIndex#</td>
<td>aerStripTriggerXXXX</td>
</tr>
</tbody>
</table>

See Also

- AerStripAllocate
- AerStripGetSample

Example

Samples\Lib\AexStrip.C
20.27. AerStripTrigger

AERERR_CODE AerStripTrigger (HAERCCTRL hAerCtrl, AXISINDEX iAxis);

Declare Function AerStripTrigger Lib “AERSYS.DLL” (ByVal hAerCtrl As Long,
                                          ByVal iAxis As Long) As Long

Parameters

  hAerCtrl       Handle to the controller.
  iAxis          Index of a physical axis (see constants below).

This function begins the data collection process when the trigger mode is set for
STRIP_TRIGGER_EXTERNAL (external trigger).

C Language and LabView Constants

AXISINDEX_XXXX

VB Constants

  aerAxisIndex#

See Also

  AerStripSetTrigger

Example

  Samples\Lib\AexStrip.C
21.1. Introduction

The Aerotech System functions are for the initialization, configuration and control of UNIDEX 600 Series controllers. Various functions exist to open paths of communication to each card in a system as well as reset the card and download firmware.

There are two steps to initializing the axis processor card for a single-threaded application (see AerSysxxxxEx functions in this chapter for multi-threaded applications).

\[
\text{AerSysOpen} \quad \text{;}\text{Open communications to the device driver}
\]
\[
\text{AerSysInitSystem} \quad \text{;}\text{Download firmware, parameter files, and configure axes}
\]
\[
\quad \text{;}\text{Execute motion}
\]
\[
\text{AerSysClose} \quad \text{;}\text{Close communications}
\]
21.2. AerSysClose

AERERR_CODE AerSysClose (HAERCTRL hAerCtrl);

Declare Function AerSysClose Lib “AERSYS.DLL” (ByVal hAerCtrl As Long) As Long

Parameters

hAerCtrl   Handle to the axis processor card.

This function closes communications to the hardware specified by the Aerotech handle. If the reference count of the handle is 0 (refer to AerSysOpenEx), the Aerotech handle (hAerCtrl) is no longer valid. If the handle is referenced, the reference count will decrease by 1 and an error code is returned notifying the user that the hAerCtrl is still open (and valid), indicating that it is in use by another thread.

See Also

AerSysOpen
AerSysOpenEx
AerSysCloseAll

Example

Samples\Lib\AexSys.C
Samples\Lib\VisualBasic\RunPgm.vbp
21.3. AerSysCloseAll

AERERR_CODE AerSysCloseAll (HAERCTRL hAerCtrl);

Declare Function AerSysCloseAll Lib “AERSYS.DLL” (ByVal hAerCtrl As Long) As Long

**Parameters**

*hAerCtrl*  Handle to the axis processor card.

*AerSysCloseAll* closes all communications to a device regardless of the current reference count. The Aerotech handle is invalid after a call to this function.

---

**Use extreme caution if calling this function and using the handle elsewhere in a program that is relying on the reference counting (such as in multi-thread applications).**

---

**See Also**

*AerSysOpen*

*AerSysOpenEx*

*AerSysCloseAll*

**Example**

Samples\Lib\AexSys.C
21.4. AerSysDownLoad

AERERR_CODE AerSysDownLoad (HAERCTRL hAerCtrl);

Declare Function AerSysDownLoad Lib “AERSYS.DLL” (ByVal hAerCtrl As Long) As Long

Parameters
  hAerCtrl   Handle to the controller.

The AerSysInitSystem function should be used instead of this function because it will download all parameter files and configure the axes.

AerSysDownLoad downloads the firmware (image) to the controller. The names of the files to download are retrieved from the system registry. Refer to the AerReg functions for further details on how the registry is setup and configured.

This function will return an error code if the firmware is already executing on the axis processor. If the image file is to be downloaded, regardless of the current status, then AerSysReset should be called prior to AerSysDownLoad.

If the system is configured for a PSO card, then the PSO firmware is automatically downloaded in this call.

See Also
  AerSysReset
  AerSysInitSystem

Example
  Samples\Lib\AexSys.C
21.5. **AerSysFaultAck**

AERERR_CODE AerSysFaultAck (HAERCTRL hAerCtrl, AXISMASK mAxis,
                                     TASKMASK mTask);

Declare Function AerSysFaultAck Lib “AERSYS.DLL” (ByVal hAerCtrl As Long,
                                     ByVal mAxis As Long, ByVal mTask As Long) As Long

**Parameters**

- **hAerCtrl**  Handle to the axis processor card.
- **mAxis**  Mask of axes to acknowledge faults (see constants).
- **mTask**  Mask of tasks to acknowledge faults (see constants).

The `AerSysFaultAck` function attempts to clear the faults for the specified axes and tasks. The function sets the `FAULT` axis parameter to –1 for each specified axis and sets the TaskFault and TaskWarning task parameters to 0 for each specified task. Axis faults, such as feedback and drive faults, cannot be cleared by this function. See the `FAULT` axis parameter in the U600MMI.hlp file for more information.

**C Language and LabView Constants**

- `TASKMASK_#`
- `AXISMASK_#`

**VB Constants**

- `aerTaskMask#`
- `aerAxisMask#`
21.6. AerSysGetDeviceID

AERERR_CODE AER_DLLENTRY AerSysGetDeviceID (HAERCTRL hAerCtrl,
            PDWORD pdwDeviceID, PDWORD pdwCard);

Declare Function AerSysGetDeviceID Lib “AERSYS.DLL” (ByVal hAerCtrl As Long,
            ByVal pdwDeviceID As Long, ByVal pdwCard As Long) As Long

Parameters

  hAerCtrl     Handle to the axis processor card.
  pdwDeviceID  Pointer returning the device identification (see constants below).
  pdwCard      Pointer returning the card number (see constants below).

This function will return the device identification from the operating system registry.

C Language and LabView Constants

  AER_UNIDEX_XXXX
  AER_CARD_XXXX

VB Constants

  aerDeviceIDXXXX
  aerCardXXXX

See Also

  AerSysReset

Example

  Samples\Lib\AexSys.C
21.7. AerSysInitSystem

AERERR_CODE AerSysInitSystem (HAERCTRL hAerCtrl, BOOL bReset, AXISMASK mAxis, TASKMASK mTask);

Declare Function AerSysInitSystem Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal bReset As Long, ByVal mAxis As Long, ByVal mTask As Long) As Long

Parameters
- *hAerCtrl* Handle to the axis processor card.
- *bReset* Should the U600 be reset and firmware automatically downloaded?
- *mAxis* Mask of axes to download axis and machine parameters (see constants).
- *mTask* Mask of tasks to download task parameters (see constants).

This function initializes the U600 card with the firmware and all parameter files. It reads the INI\U600.INI file to determine the names of the files to use. The sequence of initializing the system is as follows:

```c
If bReset Then
    // reset the card if requested
    AerSysReset
End If

// Download the firmware
AerSysDownLoad

// Download the axis parameter file
AerParmAxisDownLoadFile

// Download the machine parameter file
AerParmMachineDownLoadFile

// Download the axis configuration file
AerConfigDownLoadFile

// Download the task parameter file
AerParmTaskDownLoadFile

// Download the global parameter file
AerParmGlobalDownLoadFile

// Download 2D calibration table (if present)
AerAxisCal2DfileDownload

// Acknowledge any faults
Aer SysFaultAck
```

C Language and LabView Constants

<table>
<thead>
<tr>
<th>TASKMASK_/#</th>
<th>AXISMASK_/#</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>VB Constants</th>
</tr>
</thead>
<tbody>
<tr>
<td>aerTaskMask#</td>
</tr>
<tr>
<td>aerAxisMask#</td>
</tr>
</tbody>
</table>

**Example**

Samples\Lib\VisualBasic\RunPgm.vbp
21.8. AerSysOpen

AERERR_CODE AerSysOpen (DWORD dwDeviceID, DWORD dwCard, PHAERCTRL phAerCtrl);

Declare Function AerSysOpen Lib “AERSYS.DLL” (ByVal dwDeviceID As Long,
ByVal dwCard As Long, ByRef phAerCtrl As Long) As Long

Parameters

dwDeviceID Device identifier (see constants).
dwCard Card identifier (see constants).
phAerCtrl Points to a memory location to hold a HAERCTRL.

This function establishes communications to the specified device by creating an Aerotech Handle (HAERCTRL). The Device ID and Card is searched for in the system registry to find the appropriate values to determine how to configure the hardware. Refer to Aerotech Registry for further details on how the registry is setup and configured.

All calls to AerSysOpen should be matched by a call to AerSysClose.

The HAERCTRL is a handle that contains information that is necessary to communicate with the device. It is an internally maintained structure and is passed to any function that requires communication to the hardware. This handle should not be directly manipulated.

C Language and LabView Constants

AER_UNIDEX_XXXX
AER_CARD_XXXX

VB Constants

aerDeviceIDXXXX
aerCardXXXX

See Also

AerSysOpenEx
AerSysClose

Example

Samples\Lib\AexSys.C
Samples\Lib\VisualBasic\RunPgm.vbp
21.9. **AerSysOpenEx**

AERERR_CODE AerSysOpenEx (HAERCTRL hAerCtrl);
Declare Function AerSysOpenEx Lib “AERSYS.DLL” (ByVal hAerCtrl As Long) As Long

**Parameters**
- **hAerCtrl** Handle to the axis processor card.

`AerSysOpenEx` increments a reference count for the `hAerCtrl`. The `hAerCtrl` will not actually be destroyed by a call to `AerSysClose` unless its reference count is 0. In multi-threaded applications where each thread has a copy of the `hAerCtrl` structure, the `AerSysOpenEx` can be called within each thread to prevent any one thread from destroying the data associated with `hAerCtrl`.

All calls to `AerSysOpenEx` should be matched by a call to `AerSysClose`.

**See Also**
- `AerSysOpen`
- `AerSysClose`
- `AerSysCloseAll`

**Example**
Samples\Lib\AexSys.C
21.10. AerSysReset

AERERR_CODE AerSysReset (HAERCTRL hAerCtrl);

Declare Function AerSysReset Lib “AERSYS.DLL” (ByVal hAerCtrl As Long) As Long

Parameters

    hAerCtrl Handle to the axis processor card.

AerSysReset resets the axis processor and stops execution of firmware. The firmware
must be reloaded using AerSysDownLoad.

See Also

    AerSysDownLoad

Example

    Samples\Lib\AexSys.C
CHAPTER 22:  TASK FUNCTIONS

22.1.  Introduction

These functions allow the user to execute and control CNC programs already residing on
the controller. They have two main purposes:

1).  AerTask*( ) functions
2).  AerTaskCallBack functions (allow the user to execute functions on the PC in
CNC programs running on the controller).

These functions do not allow the caller to read or write actual CNC program code to or
from the axis processor card. This functionality is packaged in the CNC compiler, and is
only available at the C library level through the AerCmplrxxxx functions.

There are four stages of relationship of a program to a task: none, associated, active, and
executing. One must pass through these phases sequentially (i.e. a program can not
become active unless it is already associated and cannot be disassociated if it is active).
The AerTaskProgramxxxx functions allow the user to move between these stages.

Only in the executing stage, is the program “running,” and advancing the active line
number as it finishes each program step. Though the user can also set the active line
number, this can only be done when the program is in the associated stage.

CNC programs need to be compiled and downloaded before execution. Please see
samples\lib\AexProg for an example on how to compile and run a program/line.
Here is a pseudo-code example of a possible string of legal actions:

```c
AerTaskProgramAbort() ; Abort current CNC program, if any
AerTaskProgramReset() ; reset current CNC program, if any
AerTaskProgramDeassociate() ; disassociate current CNC program, if any
AerTaskProgramAssociate(x) ; Associate program x to task
AerTaskProgramSetLineUser(1) ; Optional, sets program to start at line
   ; other than the 1st line
AerTaskProgramExecute() ; Makes program active AND executing
AerTaskProgramStop() ; Stops executing, now positioned at end of
   ; current line stopped at (optional).
AerTaskProgramReset() ; Makes program inactive (but still associated)
AerTaskProgramSetLineUser(1000) ; Sets program to restart at line 1000
AerTaskProgramExecute() ; Begins executing at line 1000
AerTaskProgramAbort() ; clean up to run new CNC program
AerTaskProgramReset() ; clean up to run new CNC program
AerTaskProgramDeassociate() ; clean up to run new CNC program
```

Many of these functions require a parameter of type AER_PROG_HANDLE. The programmer must define this structure before calling such functions. The main element of this structure is the program name, by which the axis processor identifies the program.

### 22.1.1. Task Callback Functions

A “task callback” describes the mechanism that a U600 CNC program uses to request information or an action from the host computer. The task callback has the following procedure:

1. The U600 recognizes the callback command and pauses program execution on that line.
2. The U600 card generates an interrupt.
3. The device driver detects the interrupt and signals an event to notify that a task callback has been requested.
4. A waiting process on the PC detects the signaled event and requests additional data concerning the callback command from the U600 card (AerTaskCallBackGetData).
5. The process that detected the event carries out the appropriate function.
6. The process tells the U600 to continue (AerTaskCallBackContinue).
7. The U600 executes the next program line.

The callback mechanism is complex, however it allows for a great deal of customizing of the U600 CNC language. This mechanism has been greatly simplified by the U600MMI and allows for much easier customization of the CNC language. The functions described in this section deal with steps 5 and 6.

Step 3 requires creating a “task callback event.” An event must be setup by using the Event functions. Steps 4 through 6 require that the programmer start a thread.

### 22.1.2. Extending the CNC with a CALLDLL Statement

The process of writing a function in a DLL and calling a function in the DLL from within the CNC program is described in Technical Note #4 in the U600MMI.hlp file.
22.2. **AerTaskCallBackContinue**

AERERR_CODE AerTaskCallBackContinue (HAERCTRL hAerCtrl, TASKINDEX iTask, PCALLBACK_VALUE pValue, AERERR_CODE e960Rc, AERERR_CODE eTaskRc, DOUBLE dCNCReturnValue);

**Parameters**

- **hAerCtrl**
  - Handle to the axis processor card.
- **iTask**
  - Task to retrieve callback information from (see constants).
- **pValue**
  - Pointer to structure holding callback data.
- **e960Rc**
  - Set to non-zero to generate a task fault, if desired.
- **eTaskRc**
  - Value to be set into Task Parameter ErrCode.
- **dCNCReturnValue**
  - Value to set into a returned CNC variable.

This function is for backward compatibility. This function is the same as calling `AerTaskCallBackReturnSetDouble` then `AerTaskCallBackContinueEx`. See these functions for more information.

**C Language and LabView Constants**

- TASKINDEX_

**See Also**

- AerTaskCallBackReturnSetDouble
- AerTaskCallBackReturnSetString
- AerTaskCallBackContinueEx
22.3. AerTaskCallBackContinueEx

AERERR_CODE AerTaskCallBackContinueEx (HAERCTRL hAerCtrl, TASKINDEX iTask, PCALLBACK_VALUE pValue, AERERR_CODE e960Rc, AERERR_CODE eTaskRc);

Parameters
- `hAerCtrl`: Handle to the controller.
- `iTask`: Task to retrieve callback information from (see constants below).
- `pValue`: Pointer to structure holding callback data.
- `e960Rc`: Set to non-zero to generate a task fault, if desired.
- `eTaskRc`: Value to be set into Task Parameter ErrCode.

This function tells the axis processor card to continue whenever a callback function is complete. The `pValue` parameter is the same parameter passed/returned from the `AerTaskCallBackGetData` function. The `e960Rc` is an error code that will generate a task fault if it is non-zero. The `eTaskRc` will be set into the task parameter RetCode.

To set a value into a return variable, the `AerTaskCallBackReturnSetxxxx` function needs to be called first.

**C Language and LabView Constants**

- `TASKINDEX_#`

**See Also**
- `AerTaskCallBackReturnSetDouble`
- `AerTaskCallBackReturnSetString`
22.4. AerTaskCallBackGetData

AERERR_CODE AerTaskCallBackGetData (HAERCTRL hAerCtrl, TASKINDEX iTask, PCALLBACK_VALUE pValue);

Parameters

- hAerCtrl: Handle to the axis processor card.
- iTask: Task to retrieve callback information from (see constants).
- pValue: Pointer to structure to hold callback data.

When a callback event has occurred, it must get the data from the U600 card. The U600 stores all necessary data in the CALLBACK_VALUE structure. Due to the nature of the callback, the data cannot be defined until runtime. The CALLBACK_VALUE structure is a variable-binary object that stores the various data types to the callback function. The data contained in the CALLBACK_VALUE structure should be accessed using the AerTaskCallBackValuexxxx functions.

C Language and LabView Constants

- TASKINDEX_

See Also

- AerTaskCallBackValuexxxx functions
22.5. **AerTaskCallBackReturnSetDouble**

AERERR_CODE AerTaskCallBackReturnSetDouble (HAERCCTRL hAerCtrl, PCALLBACK_VALUE pValue, DOUBLE dCNCReturnValue);

**Parameters**

- **hAerCtrl** Handle to the axis processor card.
- **pValue** Pointer to structure to holding callback data.
- **dCNCReturnValue** Numeric value to set into CNC return variable.

This function sets the value of the user-supplied CNC return variable. It is used to set a callback return value just prior to doing an `AerTaskCallBackContinueEx` call. If no return value was supplied in the callback statement, or if it is of the wrong type (non-numeric), then an error is returned.

**See Also**

- `AerTaskCallBackContinueEx`
- `AerTaskCallBackReturnSetString`

**Example**

If the callback was initiated with a CALLDLL as follows:

```
$GLOBAL5=CALLDLL "file.dll" "function A" ..., 
```

then this function could be used to set `$GLOBAL5`. 
22.6. **AerTaskCallBackReturnSetString**

AERERR_CODE AerTaskCallBackReturnSetString (HAERCTRL hAerCtrl, PCALLBACK_VALUE pValue, LPCTSTR pszCNCReturnValue);

**Parameters**
- `hAerCtrl` Handle to the axis processor card.
- `pValue` Pointer to structure to holding callback data.
- `pszCNCReturnValue` String value to set into CNC return variable.

This function sets the value of the user supplied CNC return variable. It is used to set a callback return value just prior to doing an `AerTaskCallBackContinueEx` call. If no return value was supplied in the callback command, or if it is of the wrong type (non-string), then an error is returned.

**See Also**
- `AerTaskCallBackContinueEx`
- `AerTaskCallBackReturnSetDouble`

**Example**

If the callback was initiated with a CALLDLL as follows:

```
$STRGLOBAL5=CALLDLL "file.dll" "function A" …,
```

then this function could be used to set string global variable 5.
22.7. AerTaskCallBackValueGetDouble

AERERR_CODE AerTaskCallBackValueGetDouble (HAERCTRL hAerCtrl, PCALLBACK_VALUE pValue, DWORD dwArg, PDOUBLE pdValue);

Parameters

- **hAerCtrl**: Handle to the axis processor card.
- **pValue**: Pointer to CALLBACK_VALUE data returned from AerTaskCallBackGetData.
- **dwArg**: Parameter number of parameter to retrieve (zero-based).
- **pdValue**: Pointer to hold the retrieved double value.

This function reads the double (numeric) value that was passed to the callback function. If a variable was passed as the argument, the current value of the variable is returned to the callback. This function will fail (return an error code) if a nonnumeric data type was passed to the callback, as the specified parameter.

The programmer should keep in mind that in a CALLDLL callback statement, the first two arguments are always the dll name and dll function name, respectively. Therefore, the remaining arguments always start at \( dwArg = 2 \).

See Also

- AerTaskCallBackValueGetDWORD
- AerTaskCallBackValueGetString
22.8. **AerTaskCallBackValueGetDWORD**

AERERR_CODE AerTaskCallBackValueGetDWORD (HAERCTRL hAerCtrl, PCALLBACK_VALUE pValue, DWORD dwArg, PDWORD pdwValue);

**Parameters**

- **hAerCtrl**: Handle to the axis processor card.
- **pValue**: Pointer to CALLBACK_VALUE data returned from *AerTaskCallBackGetData*.
- **dwArg**: Parameter number of parameter to retrieve (zero-based).
- **pdwValue**: Pointer to hold the retrieved DWORD (integer) value.

This function returns an integer value passed to the callback function. If a variable was passed as the argument, the current value of the variable is returned. Any floating-point value is truncated. This function will fail (return an error code) if a nonnumeric data type was passed, as the specified parameter.

The programmer should keep in mind that in a CALLDLL callback statement, the first two arguments are always the dll name and dll function name, respectively. Therefore, the remaining arguments always start at \( dwArg = 2 \).

**See Also**

- AerTaskCallBackValueGetDouble
- AerTaskCallBackValueGetString
22.9. AerTaskCallBackValueGetString

AERERR_CODE AerTaskCallBackValueGetString (HAERCTRL hAerCtrl, PCALLBACK_VALUE pValue, DWORD dwArg, LPTSTR pszValue, DWORD dwMaxBytes);

Parameters

- **hAerCtrl**: Handle to the axis processor card.
- **pValue**: Pointer to CALLBACK_VALUE data returned from AerTaskCallBackGetData.
- **dwArg**: Parameter number to retrieve (zero-based).
- **pszValue**: Pointer to hold the retrieved string value.
- **dwMaxBytes**: Maximum size of string to retrieve.

This function returns the string value passed to the callback function. If a variable was passed as the argument, the current value of the variable is returned. This function will fail (return an error code) if a non-string data type was passed as the specified parameter.

The programmer should keep in mind that in a CALLDLL callback statement, the first two arguments are always the dll name and dll function name, respectively. Therefore, the remaining arguments always start at `dwArg = 2`.

See Also

- AerTaskCallBackValueGetDouble
- AerTaskCallBackValueGetDWORD
22.10. AerTaskCallBackValueMakeString

AERERR_CODE AerTaskCallBackValueMakeString (HAERCTRL hAerCtrl, PCALLBACK_VALUE pValue, DWORD dwArg, LPTSTR pszValue, DWORD dwMaxBytes);

Parameters

- **hAerCtrl**  Handle to the axis processor card.
- **pValue**  Pointer to CALLBACK_VALUE data returned from AerTaskCallBackGetData.
- **dwArg**  Parameter number to retrieve (zero-based).
- **pszValue**  Pointer to hold the retrieved string value.
- **dwMaxBytes**  Maximum size of string to retrieve.

This function will generate one string from all arguments, starting with *dwArg*.

The programmer should keep in mind that in a CALLDLL callback statement, the first two arguments are always the dll name and dll function name, respectively. Therefore, the remaining arguments always start at *dwArg* = 2.

See Also

- AerTaskCallBackGetData
22.11. AerTaskCallBackValueSetDouble

AERERR_CODE AerTaskCallBackValueSetDouble (HAERCTRL hAerCtrl,
   PCALLBACK_VALUE pValue, DWORD dwArg, DOUBLE dValue);

Parameters
   hAerCtrl   Handle to the axis processor card.
   pValue     Pointer to CALLBACK_VALUE data returned from
               AerTaskCallBackGetData.
   dwArg      Parameter number of parameter to set (zero-based).
   dValue     Value to set into specified variable.

This function sets the double (numeric) value of the variable that was passed to the
callback function. The variable can either be a local, task, or global variable, as well as a
binary input/output or register input/output parameter, etc. This function will fail (return
an error code) if a double variable was not passed as the specified parameter.

The programmer should keep in mind that in a CALLDLL callback statement, the first
two arguments are always the dll name and dll function name, respectively. Therefore, the
remaining arguments always start at dwArg = 2.

See Also
   AerTaskCallBackGetData
   AerTaskCallBackValueSetDWORD
   AerTaskCallBackValueSetString
22.12. **AerTaskCallBackValueSetDWORD**

AERERR_CODE AerTaskCallBackValueSetDWORD (HAERCTRL hAerCtrl,
PCALLBACK_VALUE pValue, DWORD dwArg, DWORD dwValue);

**Parameters**

- **hAerCtrl**  Handle to the axis processor card.
- **pValue**  Pointer to CALLBACK_VALUE data returned from
  AerTaskCallBackGetData.
- **dwArg**  Parameter number to set (zero-based).
- **dwValue**  Value to set into specified variable.

This function sets the integer value to the variable that was passed to the callback function. The variable can either be a local, task, or global variable, as well as a binary input/output or register input/output parameter, etc. This function will fail (return an error code) if a DWORD variable was not passed as the specified parameter.

The programmer should keep in mind that in a CALLDLL callback statement, the first two arguments are always the dll name and dll function name, respectively. Therefore, the remaining arguments always start at \( dwArg = 2 \).

**See Also**

- AerTaskCallBackGetData
- AerTaskCallBackValueSetDouble
- AerTaskCallBackValueSetString
22.13. AerTaskCallBackValueSetString

AERERR_CODE AerTaskCallBackValueSetString (HAERCTRL hAerCtrl, PCALLBACK_VALUE pValue, DWORD dwArg, LPTSTR pszValue);

Parameters

- **hAerCtrl**: Handle to the axis processor card.
- **pValue**: Pointer to CALLBACK_VALUE data returned from AerTaskCallBackGetData.
- **dwArg**: Parameter number to set (zero-based).
- **pszValue**: Value to set into specified variable.

This function sets the string value of the variable that was passed to the callback function. The variable can be any valid string variable. This function will fail (return an error code) if a nonnumeric data type or constant was passed as the specified parameter.

The programmer should keep in mind that in a CALLDLL callback statement, the first two arguments are always the dll name and dll function name, respectively. Therefore, the remaining arguments always start at \( dwArg = 2 \).

See Also

- AerTaskCallBackGetData
- AerTaskCallBackValueSetDouble
- AerTaskCallBackValueSetDWORD

AERERR_CODE AerTaskCallBackValueValidateNumArgs (PCALLBACK_VALUE pValue, DWORD dwMin, DWORD dwMax);

**Parameters**

- *pValue*  
  Pointer to CALLBACK_VALUE data returned from *AerTaskCallBackGetData*.
- *dwMin*  
  Minimum number of parameters for callback.
- *dwMax*  
  Maximum number of parameters for callback.

This function is supplied so that the writer of a callback function can quickly validate the number of parameters that were passed to the function. This function simply checks the min and max against the actual number passed (the *dwArgs* member of the *pValue* structure). The actual number must be within the minimum and maximum values (inclusive) or an appropriate error is returned. The argument count does not include the DLL or function name.

**See Also**

*AerTaskCallBackGetData*
### 22.15. AerTaskGetPhysAxisFromTaskAxis

**C Function**

```c
AERERR_CODE AerTaskGetPhysAxisFromTaskAxis (HAERCTRL hAerCtrl, TASKINDEX iTask, TASKAXISINDEX iTaskAxis, PPHYSAXISINDEX piPhysAxis);
```

**VB Function**

```vb
Declare Function AerTaskGetPhysAxisFromTaskAxis Lib "AERSYS.DLL" (ByVal hAerCtrl As Long, ByVal iTask As Long, ByVal iTaskAxis As Double, ByRef piPhysAxis As Double) As Long
```

#### Parameters

- **hAerCtrl** Handle to an Aerotech Control.
- **iTask** Task Index (see constants below).
- **iTaskAxis** Task Axis Index to find physical axis (see constants).
- **piPhysAxis** Physical axis index (returned by this function) (see constants).

This function returns the physical axis number mapped to the given task axis number for the given task. Each task can have its own independent mapping. By default, for each task, the axes are mapped so the task and axes numbers are the same. Physical axis 0 is mapped to task axis 0, and so on.


#### C Language and LabView Constants

- **TASKAXISINDEX_#**
- **AXISINDEX_#**
- **PHYSAXISINDEX_#**

#### VB Constants

- **aerTaskIndex#**
- **aerTaskAxisIndex#**
- **aerPhysAxisIndex#**

#### See Also

- `AerTaskProgramGetTaskAxisFromPhysAxis`
- `AerTaskProgramImmediateMapAxis`
22.16. AerTaskGetTaskAxisFromPhysAxis

AERERR_CODE AerTaskGetTaskAxisFromPhysAxis (HAERCTRL hAerCtrl,
    TASKINDEX iTask, PHYSAXISINDEX iPhysAxis,
    PTASKAXISINDEX piTaskAxis);

Declare Function AerTaskGetTaskAxisFromPhysAxis Lib “AERSYS.DLL” (ByVal
    hAerCtrl As Long, ByVal iTask As Long, ByRef iPhysAxis As Double,
    ByRef piTaskAxis As Double) As Long

Parameters
    hAerCtrl          Handle to an Aerotech Control.
    iTask             Task index (see constants below).
    piTaskAxis        Task axis index mapped to this physical axis.
    iPhysAxis         Physical axis index to find the Task Index of.

This function returns the task axis number mapped to the given physical axis number. Each task can have its own independent mapping. By default, for each task, the axes are mapped so the task and axes numbers are the same: physical axis 0 is mapped to task axis 0, and so on.

Please see the U600 Series Users Guide manual, P/N EDU157, under Axis Arbitration for details on mapping axes.

C Language and LabView Constants
    TASKAXISINDEX_#
    AXISINDEX_#
    PHYSAXISINDEX_#

VB Constants
    aerTaskIndex#
    aerTaskAxisIndex#
    aerPhysAxisIndex#

See Also
    AerTaskProgramGetPhysAxisFromTaskAxis
    AerTaskProgramImmediateMapAxis
22.17. AerTaskImmediateBindAxis

AERERR_CODE AerTaskImmediateBindAxis (HAERCTRL hAerCtrl, TASKINDEX iTask, TASKAXISINDEX iTaskAxis);

Declare Function AerTaskImmediateBindAxis Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal iTask As Long, ByVal iTaskAxis As Double) As Long

Parameters

- **hAerCtrl**: Handle to an Aerotech Control.
- **iTask**: Task Index to bind the axis to (see constants below).
- **iTaskAxis**: Task Axis Index to bind.

This function binds the axis to the task. This function is functionally equivalent to executing the CNC command: BIND. Please see the *U600 Series Users Guide manual, P/N EDU157*, under Axis Arbitration for details on binding (although the axes are bound via the definitions in the \INI\AxisCfg.INI file).

**C Language and LabView Constants**

- TASKAXISINDEX_
- TASKINDEX_

**VB Constants**

- aerTaskIndex#
- aerTaskAxisIndex#

See Also

- AerTaskImmediateFreeAxis
- AerTaskImmediateCaptureAxis
- AerTaskImmediateMapAxis
22.18. AerTaskImmediateCaptureAxis

AERERR_CODE AerTaskImmediateCaptureAxis (HAERCTRL hAerCtrl, TASKINDEX iTask, TASKAXISINDEX iTaskAxis);

Declare Function AerTaskImmediateCaptureAxis Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal iTask As Long, ByVal iTaskAxis As Double) As Long

Parameters

  hAerCtrl  Handle to an Aerotech Control.
  iTask     Task Index to capture the axis (see constants below).
  iTaskAxis Task Axis Index of axis to capture (see constants below).

This function captures the axis on the given task. This function is equivalent to executing the CNC command: CAPTURE. Please see the U600 Series Users Guide manual, P/N EDU157, under CNC G Code Programming for details on capturing axes (although the axes are captured via the definitions in the \INI\AxisCfg.INI file).

C Language and LabView Constants

  TASKAXISINDEX_
  TASKINDEX_

VB Constants

  aerTaskIndex#
  aerTaskAxisIndex#

See Also

  AerTaskImmediateFreeAxis
  AerTaskImmediateBindAxis
  AerTaskImmediateMapAxis
22.19. AerTaskImmediateFreeAxis

AERERR_CODE AerTaskImmediateFreeAxis (HAERCTRL hAerCtrl, TASKINDEX iTask, TASKAXISINDEX iTaskAxis);

Declare Function AerTaskImmediateFreeAxis Lib "AERSYS.DLL" (ByVal hAerCtrl As Long, ByVal iTask As Long, ByVal iTaskAxis As Double) As Long

Parameters

- **hAerCtrl**: Handle to an Aerotech Control.
- **iTask**: Task index to free the axis from (see constants below).
- **iTaskAxis**: Task Axis Index of axis to free (see constants below).

This function frees the axis from the given task. This function is functionally equivalent to executing the CNC command: FREE. Please see the *U600 Series Users Guide manual, P/N EDU157*, under Axis Arbitration for details on freeing axes (if required by your application).

C Language and LabView Constants

- **TASKAXISINDEX_#**
- **TASKINDEX_#**

VB Constants

- **aerTaskIndex#**
- **aerTaskAxisIndex#**

See Also

- AerTaskImmediateBindAxis
- AerTaskImmediateCaptureAxis
- AerTaskImmediateMapAxis
22.20. **AerTaskImmediateMapAxis**

```c
AERERR_CODE AerTaskImmediateMapAxis (HAERCTRL hAerCtrl, TASKINDEX iTask, TASKAXISINDEX iTaskAxis, PHYSAXISINDEX iPhysAxis);
```

Declare Function `AerTaskImmediateMapAxis Lib “AERSYS.DLL”` (ByVal `hAerCtrl` As Long, ByVal `iTask` As Long, ByVal `iTaskAxis` As Double, ByVal `iPhysAxis` As Double) As Long

**Parameters**

- `hAerCtrl` Handle to an Aerotech Control.
- `iTask` Task Index to map axis to (see constants below).
- `iTaskAxis` Task Axis Index to map (see constants below).
- `iPhysAxis` Physical axis index to map to (see constants below).

This function maps an axis to a task. This function is functionally equivalent to executing the CNC command: MAP. Please see the *U600 Series Users Guide manual*, P/N EDU157, under Axis Arbitration for details on mapping axes (axes are mapped via the \INI\AxisCfg.INI file).

**C Language and LabView Constants**

- `TASKAXISINDEX_#`
- `AXISINDEX_#`
- `PHYSAXISINDEX_#`

**VB Constants**

- `aerTaskIndex#`
- `aerTaskAxisIndex#`
- `aerPhysAxisIndex#`

**See Also**

- `AerTaskImmediateFreeAxis`
- `AerTaskImmediateCaptureAxis`
- `AerTaskImmediateBind`
22.21. AerTaskModeGetName

AERERR_CODE AerTaskModeGetName (HAERCTRL hAerCtrl, DWORD dwBit, LPTSTR pszName);

Declare Function AerTaskModeGetName Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal dwBit As Long, ByVal pszName As String) As Long

Parameters

- `hAerCtrl`: Handle to an Aerotech Control.
- `dwBit`: Bit number (zero-based) to get the name of.
- `pszName`: String to return the mode name.

All string variables in MS Visual Basic, passed by reference (ByRef), must be declared as fixed length strings within your program, long enough to hold the string value returned by the function. Also, those string variables which are passed, with another parameter indicating the length of the string variable, must also be fixed length strings, otherwise, you would not be able to pass the length of the string. For example, to declare a fixed length string:

```
DIM sGlobStr as STRING * 50 ; 50 characters long
```

This function copies the name associated with the bit of the task mode word that the user indicated. The `pszName` must point to a buffer allocated to at least MAX_PARM_NAME_LEN+1 bytes.

The bit numbering is zero-based, with the zero bit corresponding to the ones placed in the binary mask representation of the mode word. There are 32 bits in the task mode word, indicated by the constants (below) for VB. The valid `dwBit` values are from 0 to 31. If the passed bit number is invalid, it copies the string "Invalid Bit" into the passed buffer.

C Language and LabView Constants

- No Constants

VB Constants

- `aerBit#`

See Also

- AerTaskStatusGetName
22.22. AerTaskProgramAbort

AERERR_CODE AerTaskProgramAbort (HAERCTRL hAerCtrl, TASKINDEX iTask);
Declare Function AerTaskProgramAbort Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal iTask As Long) As Long

**Parameters**
- *hAerCtrl* Handle to an Aerotech Control.
- *iTask* Task index (see constants below).

This function immediately stops execution of the CNC program. If the task is not associated, active, and executing at the time this function is called, then this function does nothing and returns no error code. If a program is executing at the time this function is called, the task will abort the current CNC statement and exit executing mode. However, the task will remain in active mode after execution of this function. This function will also halt any immediate command that is running on the task.

In contrast to AerTaskProgramStop this function does not increment the current line number to the next line. Therefore, if the user performs an AerTaskProgramExecute after executing this function (with no intervening AerTaskProgramSetLineUser), then the program will begin execution at the beginning of the line that was aborted.

This function will cause an abrupt stop of any current CNC G-code-type or synchronous motion without deceleration.

**C Language and LabView Constants**

- TASKINDEX_XXX

**VB Constants**

- AerTaskIndex#

**See Also**
- AerTaskProgramExecute
- AerTaskProgramStop
- AerMoveAbort
- AerMoveHalt

**Example**

Samples\Lib\VisualBasic\RunPgm.vbp
22.23. AerTaskProgramAssociate

AERERR_CODE AerTaskProgramAssociate (HAERCTRL haerCtrl, TASKINDEX iTask, PAER_PROG_HANDLE pHandle);

Declare Function AerTaskProgramAssociate Lib "AERSYS.DLL" (ByVal haerCtrl As Long, ByVal iTask As Long, ByVal pHandle As Long) As Long

Parameters

- haerCtrl: Handle to an Aerotech Control.
- iTask: Task index (see constants below).
- pHandle: Pointer to completed AER_PROG_HANDLE structure.
- pName: String containing the CNC program handle.

This function will associate a user CNC program with a task. Programs must be associated with a task before they can be made active, or be executed by the axis processor card. The programmer must provide the program name via the pName/pHandle parameters. Use AerTaskProgramDeAssociate to disassociate programs from a task.

C Language and LabView Constants

- TASKINDEX_XXXX

VB Constants

- AerTaskIndex#

See Also

- AerTaskProgramDeAssociate
- AerTaskProgramAbort
- AerTaskProgramReset

Example

- Samples\Lib\AexProg.C
- Samples\Lib\VisualBasic\RunPgm.vbp
22.24. AerTaskProgramDeAssociate

AERERR_CODE AerTaskProgramDeAssociate (HAERCTRL hAerCtrl, TASKINDEX iTask);

Declare Function AerTaskProgramDeAssociate Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal iTask As Long) As Long

Parameters

  hAerCtrl   Handle to the axis processor card.
  iTask      Index of task to use (see constants below).

This function de-associates a CNC program from the specified task. It will fail if the state of the program on the given task is executing or active.

C Language and LabView Constants

  TASKINDEX_XXXX

VB Constants

  AerTaskIndex#

See Also

  AerTaskProgramAbort
  AerTaskProgramAssociate
  AerTaskProgramReset

Example

  Samples\Lib\VisualBasic\RunPgm.vbp
### 22.25. AerTaskProgramExecute

AERERR_CODE AerTaskProgramExecute (HAERCTRL hAerCtrl, TASKINDEX iTask, DWORD dwType);

Declare Function AerTaskProgramExecute Lib "AERSYS.DLL" (ByVal hAerCtrl As Long, ByVal iTask As Long, ByVal dwType As Long) As Long

**Parameters**

- **hAerCtrl** Handle to an Aerotech Control.
- **iTask** Task index (see constants below).
- **dwType** See constants below.

This function begins executing an associated program and makes the program active. It begins executing at the current user line number. A program must be associated to the task before it can be executed. The program may be active. If the program is executing, no operation takes place, and no error is returned.

There is no *AerTaskProgramContinue* function; this function should be used for the initial execution of a program and for continuing execution after a stop or abort.

**C Language and LabView Constants**

- TASKINDEX_XXXX
- TASKEXEC_XXXX

**VB Constants**

- aerTaskIndex#
- aerTaskExecXXXX

**See Also**

- AerTaskProgramSetLineUser
- AerTaskProgramAbort
- AerTaskProgramStop
- AerTaskProgramAssociate
- AerTaskProgramContinue

**Example**

- Samples\Lib\AexProg.C
- Samples\Lib\VisualBasic\RunPgm.vbp
22.26. AerTaskProgramGetLineUser

AERERR_CODE AerTaskProgramGetLineUser (HAERCTRL hAerCtrl, TASKINDEX iTask, PDWORD pdwLineUser);

Declare Function AerTaskProgramGetLineUser Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal iTask As Long, ByRef pdwLineUser As Long) As Long

Parameters

- **hAerCtrl** Handle to the axis processor card.
- **iTask** Index of task to use (see constants below).
- **pdwLineUser** Pointer to variable to hold current line number.

This function returns the current user line number of the associated program that the task is executing.

**C Language and LabView Constants**

- **TASKINDEX_XXXX**

**VB Constants**

- **aerTaskIndex#**

See Also

- **AerTaskProgramSetLineUser**
### 22.27. AerTaskProgramReset

AERERR_CODE AerTaskProgramReset (HAERCTRL hAerCtrl, TASKINDEX iTask);

Declare Function AerTaskProgramReset Lib "AERSYS.DLL" (ByVal hAerCtrl As Long, ByVal iTask As Long) As Long

**Parameters**

- **hAerCtrl**: Handle to an Aerotech Control.
- **iTask**: Task index (see constants below).

This function resets a miscellany of data that may have been set due to program execution, back to their initial or nominal state:

1. The task is made inactive (if active and not executing).
2. The axis processor stops waiting for a “callback” response, if it was waiting.
3. The axis processor drops the “move interrupted” state, which may have been active if it was interrupted by an ONGOSUB statement, “Jog and Return,” or “Jog and Offset” commands.

The user can only perform an AerTaskProgramReset if the task is not executing a program. If a task is executing a program when this function is called, it does nothing and returns an error code. If the task is not associated to a program or active when this function is called, it does nothing and does not return an error code.

One common use of this function allows the user to use AerTaskProgramSetLineUser to reset the current line, after a program halt or abort.

The values of task variables are not altered by this function.

### C Language and LabView Constants

- **TASKINDEX_XXXX**

### VB Constants

- **AerTaskIndex#**

### See Also

- AerTaskProgramExecute
- AerTaskProgramSetLineUser
- AerTaskReset

### Example

```
Samples\Lib\VisualBasic\RunPgm.vbp
```
22.28. AerTaskProgramSetLineUser

AERERR_CODE AerTaskProgramSetLineUser (HAERCTRL hAerCtrl, TASKINDEX iTask, DWORD dwLineUser);

Declare Function AerTaskProgramSetLineUser Lib "AERSYS.DLL" (ByVal hAerCtrl As Long, ByVal iTask As Long, ByVal dwLineUser As Long) As Long

Parameters

hAerCtrl Handle to an Aerotech Control.
 iTask Task index (see constants below).
 dwLineUser Program user line index.

This function sets the “current” CNC program line to be executed. This is the line that will be the first one executed on the next call to AerTaskProgramExecute. In order to set the current user line number, a program must be associated to a task but not be active and not currently executing.

This function should be used prior to beginning a program execution or after a program stop or abort in order to change the current line before continuing execution again.

In order to use this function properly, the user must understand that the axis processor recognizes two line numbering systems called “960” and “user.” This function expects a user line number as its input.

The 960 line numbering system is zero-based and each incoming CNC program line is simply assigned unique consecutive numbers (for that program).

The user line number is part of the “code packet” passed down to the axis processor for each CNC line. The axis processor does not determine or make any checks on the validity of the user line number passed down in the packet. Therefore, there can be (and often are, if the Aerotech CNC compiler has downloaded the packet) multiple 960 lines in the same program with the same user line number. In performing an AerTaskProgramSetLineUser call, the axis processor will look for the first CNC line (with the lowest 960 line number) that has the given user line number.

The first time a program is downloaded to the controller, it is given a default user line number that corresponds to the first 960 line in the program. Therefore, the user does not need to call this function in order to begin execution at the first line. However, if another (or the same) program is downloaded again, the axis processor will not reset the current user line number. It remains at whatever value it had in the original program. Consequently, it is strongly advised that programmers call this function even before the first execution of a program.
The user should be cautioned when using this function: undesirable or unexpected results could occur if the line is changed into or out of a block statement. For example, suppose the user changes to a line in a subroutine, thereby bypassing the call statement. When the program reaches the return statement of the subroutine, it finds it has no place to return. Similar problems can happen when jumping into an if-endif, or while-endwhile block of statements.

C Language and LabView Constants

\textit{TASKINDEX\_XXXX}

\textbf{VB Constants}

\texttt{AerTaskIndex#}

\textbf{See Also}

\texttt{AerTaskProgramAssociate}

\texttt{AerTaskProgramExecute}

\texttt{AerTaskProgramStop}
22.29. AerTaskProgramStop

AERERR_CODE AerTaskProgramStop (HAERCTRL hAerCtrl, TASKINDEX iTask);

Declare Function AerTaskProgramStop Lib "AERSYS.DLL" (ByVal hAerCtrl As Long,
ByVal iTask As Long) As Long

Parameters
    hAerCtrl Handle to an Aerotech Control.
    iTask Task index (see constants below).

This function stops execution of the CNC program at the end of the current block. If the
task is not associated, active, and executing at the time this function is called, then this
function does nothing and returns no error code. If a program is executing at the time this
function is called, the task will finish the current CNC statement, set the current line
number to the next user line and exit executing mode. However, the task will remain in
active mode after execution of this function.

This function will not affect any current CNC G-code type motion, because the CNC G-
code block only finishes after the move is complete. Therefore, G-code movement will
complete normally before the program stops execution.

It is important to realize that all other forms of motion are not effected by a program stop.
If the programmer begins asynchronous motion during a program and then executes this
function, the asynchronous motion will continue on. The user must use the
AerTaskProgramAbort, AerMoveAbort, or AerMoveStop functions to stop asynchronous
motion.

C Language and LabView Constants
    TASKINDEX_XXXX

VB Constants
    AerTaskIndex#

See Also
    AerTaskProgramExecute
    AerTaskProgramAbort
    AerMoveAbort
    AerMoveHalt
22.30. AerTaskReset

AERERR_CODE AerTaskReset (HAERCTRL hAerCtrl, TASKINDEX iTask);

Declare Function AerTaskReset Lib "AERSYS.DLL" (ByVal hAerCtrl As Long, ByVal iTask As Long) As Long

Parameters
   hAerCtrl  Handle to an Aerotech Control.
   iTask     Task index (see constants below).

This function resets a miscellany of task data. It can be viewed as a harder reset than the AerTaskProgramReset function. The following is performed:

1. The task is de-associated from any program.
2. Any task fault is cleared.
3. The program call stack is cleared, along with any interrupted move data.
4. R-Theta transformations, if any, are removed.
5. The priority is reset to NORMAL (if an ONGOSUB statement raised it).
6. Any fixture offsets are removed.

The user can only perform a task reset if the task is not executing a program. If a task is executing a program when this function is called, it does nothing and returns an error code. If the task is active when this function is called, this function will first automatically perform an AerTaskProgramReset, and then perform as described above. If the task is not associated when this function is called, it does nothing and does not return an error code.

One common use of this function is to clear any residual settings due to previous program execution before associating and running another program.

C Language and LabView Constants
   TASKINDEX_XXXX

VB Constants
   AerTaskIndex#

See Also
   AerTaskProgramReset
22.31. AerTaskSetExecuteMode

AERERR_CODE AerTaskSetExecuteMode (HAERCTRL hAerCtrl, TASKINDEX iTask, DWORD dwMode);

Declare Function AerTaskSetExecuteMode Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal iTask As Long, ByVal dwMode As Long) As Long

Parameters
- hAerCtrl Handle to the axis processor card.
- iTask Index of task to use (see constants below).
- dwMode Mode of execution (see constants below).

The AerTaskSetExecuteMode function sets the default mode of execution for AerTaskProgramExecute.

C Language and LabView Constants
- TASKINDEX_XXXX
- TASKEXEC_XXXX

VB Constants
- aerTaskIndex#
- aerTaskExecXXXX

See Also
- AerTaskProgramExecute
22.32. AerTaskSetRetraceMode

AERERR_CODE AerTaskSetRetraceMode (HAERCTRL hAerCtrl, TASKINDEX iTask, DWORD dwMode);

Declare Function AerTaskSetRetraceMode Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal iTask As Long, ByVal dwMode As Long) As Long

Parameters

hAerCtrl Handle to the axis processor card.

iTask Index of task to use (see constants below).

dwMode Set to 0 to disable retrace mode, 1-enables retrace mode.

This function sets/resets retrace mode for the specified task. If a program is executing, a program stop command is issued. It is necessary to issue the execute command to get the program running again. See the U600MMI.hlp file for more information on the retrace mode.

C Language and LabView Constants

TASKINDEX_XXXX

VB Constants

AerTaskIndex#
22.33. AerTaskSetSlewMode

AERERR_CODE AerTaskSetSlewMode (HAERCTRL hAerCtrl, TASKINDEX iTask, DWORD dwMode);

Declare Function AerTaskSetSlewMode Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal iTask As Long, ByVal dwMode As Long) As Long

Parameters
- **hAerCtrl** Handle to the axis processor card.
- **iTask** Index of task to use (TASKINDEX_xxxx constant).
- **dwMode** Set 0 to turn off slew Mode, 1 to turn on.

This function enables the joystick slew mode on the specified task. By enabling slew mode, this function enables the joystick. The Joystick then works as configured by the following task parameters: JoyStickPort and SlewPair1 through SlewPair8. These parameters must be set appropriately before enabling slew mode. See the U600MMI.hlp file for more information on the joystick.

**C Language and LabView Constants**

`TASKINDEX_XXXX`

**VB Constants**

`aerTaskIndex#`
### 22.34. **AerTaskStatusGetName**

AERERR_CODE AerTaskStatusGetName (HAERCTRL hAerCtrl, DWORD dwBit, LPTSTR pszName);

Declare Function AerTaskStatusGetName Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal dwBit As Long, ByVal pszName As String) As Long

#### Parameters

- **hAerCtrl**
  - Handle to an Aerotech Control.

- **dwBit**
  - Bit number (zero-based) to get the name of (see constants).

- **pszName**
  - Pointer to the String to return the task name.

All string variables in MS Visual Basic, passed by reference (ByRef), must be declared as fixed length strings within your program, long enough to hold the string value returned by the function. Also, those string variables which are passed, with another parameter indicating the length of the string variable, must also be fixed length strings, otherwise, you would not be able to pass the length of the string. For example, to declare a fixed length string:

```vbnet
DIM sGlobStr as STRING * 50 ; 50 characters long
```

This function copies the name associated with the bit of the task status word that the user indicated. The `pszName` must point to a buffer allocated to at least MAX_PARM_NAME_LEN+1 bytes.

The bit numbering is zero-based, with the zero bit corresponding to the one’s place in the binary mask representation of the first status word (there are three task status words (Status1, Status2, and Status3), each 32 bits long). The second and third task status words are accessed by adding 32 or 64 respectively, to the desired bit number in that word. There are 32 bits in each task status word and the valid `dwBit` values are from 0 to 95. If the passed bit number is invalid, it copies the string “Invalid Bit” into the passed buffer. See the U600MMI.hlp file for more information on the task status parameters.

**C Language and LabView Constants**

- **No Constants**

**VB Constants**

- **aerBit#**

**See Also**

- **AerTaskModeGetName**
CHAPTER 23: TOOL FUNCTIONS

23.1. Introduction

The Tool Functions are responsible for managing both the Tool Files (AerToolFileXXX) and the Tool Tables (AerToolTableXXX).

The Tool Table is used in conjunction with the CNC programming language. Tool Tables are designed to work with the following groups GCODES/CNC commands: G1/G2/G3, G40/G41/G42, G43, G143/G144/G149, and the T-Word.

The Tool File and Table functions rely on the AER_TOOL structure:

```c
typedef struct tagAER_TOOL
{
    DWORD   mFlags;          // AERTOOL_F_xxx Constants
    DOUBLE  dCutterRadius;   // TaskParm: CutterRadius
    DOUBLE  dCutterLength;   // TaskParm: CutterLength
    DOUBLE  dCutterWear;     // TaskParm: CutterWear
    DOUBLE  dOffsetX;        // TaskParm: CutterOffsetX
    DOUBLE  dOffsetY;        // TaskParm: CutterOffsetY
    DOUBLE  dFWord;          // if <> 0 Then F=dFWord
    DOUBLE  dSWord;          // if <> 0 Then S=dSWord
    CHAR    szName[MAX_TOOL_NAME_LEN] ;// Tool Description
} AER_TOOL;
typedef AER_TOOL  *PAER_TOOL;
```
The following flag values are available:

**AERTOOL_F_ENGLISH**

All Cutter and Offset values are specified in Inches. Otherwise units are metric (mm).

**AERTOOL_F_INUSE**

A tool can be marked as “not in use”. The system won’t allow the Tool to be made active.

**AERTOOL_F_FORCE_UNITS**

The CNC program must be in the same units as the tool. If units are specified in English (AERTOOL_F_ENGLISH) and a CNC program is in metric (G71), then the system will not allow the Tool to become active if this value is set. If this bit is not set, the values will be converted from English units to Metric units and the tool will be made active.

**AERTOOL_F_USER_DIAMETER**

This item is for display/user input purposes only. The User expects values to be entered/displayed as diameters. The interface should multiply/divide the CutterRadius value by 2 when displaying/setting the value.

**AERTOOL_F_VALID**

This internal bit is set in the tool table after an individual tool has been successfully set with AerToolTableSetTool.

The Tool File functions manage the storing and retrieving of the AER_TOOL structure to a file. The Tool Table functions manage the tools on the controller.
23.2. AerToolFileDownload

AERERR_CODE AerToolFileDownload (HAERCTRL hAerCtrl, LPCTSTR pszFile)

Declare Function AerToolFileDownload Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal pszFile As String) As Long

Parameters

\begin{align*}
\textit{hAerCtrl} &\quad \text{Handle to the axis processor card.} \\
\textit{pszFile} &\quad \text{Tool File.}
\end{align*}

This function allocates memory on the controller and downloads the specified Tool File.

The name of the Tool file can be retrieved with \textit{AerRegGetFileName}.

The code for the function follows:

\begin{verbatim}
AERERR_CODE AER_DLLENTRY AerToolFileDownload (HAERCTRL hAerCtrl, LPCTSTR pszFile)
{
    AER_TOOL tTool;
    DWORD dwNumTools;
    DWORD dwTool;
    AERERR_CODE eRc;

    // get number of tools from file
    eRc = AerToolFileGetNumTools( pszFile, &dwNumTools );
    RETURN_ON_ERR( eRc );

    // free tool table on card
    eRc = AerToolTableFree( hAerCtrl );
    RETURN_ON_ERR( eRc );

    // allocate the number of tools on card
    eRc = AerToolTableAllocate( hAerCtrl, dwNumTools );
    RETURN_ON_ERR( eRc );

    for( dwTool = 0; dwTool < dwNumTools; dwTool++ )
    {
        // read the tool from the file
        eRc = AerToolFileGetTool( pszFile, dwTool, &tTool );
        BREAK_ON_ERR( eRc );

        // write the tool to the card
        eRc = AerToolTableSetTool( hAerCtrl, dwTool, &tTool );
        BREAK_ON_ERR( eRc );
    }

    return( eRc );
}
\end{verbatim}

See Also

\textit{AerRegGetFileName}
23.3. AerToolFileGetNumTools

AERERR_CODE AerToolFileGetNumTools (LPCTSTR pszFile, PDWORD pdwNumTools)

**Parameters**

- **pszFile**: Tool File.
- **pdwNumTools**: Number of Tools in the tool file.

This function returns the number of tools that are in the tool file.

The name of the Tool file can be retrieved with AerRegGetFileName (AERREGID_ToolFile).

**See Also**

- AerRegGetFileName
- AerToolFileDownload
23.4. AerToolFileGetTool

AERERR_CODE AerToolFileGetTool (LPCTSTR pszFile, DWORD dwTool, PAER_TOOL pTool)

Parameters

pszFile Tool File.
dwTool Which tool to retrieve from file (0-Based).
pTool Pointer to AER_TOOL structure to hold tool info.

This function returns the tool information for the given tool.

The name of the Tool file can be retrieved with AerRegGetFileName (AERREGID_ToolFile).

See Also

AerRegGetFileName
AerToolFileDownload
AerToolFileSetTool
23.5. AerToolFileSetTool

AERERR_CODE AerToolFileSetTool (LPCTSTR pszFile, DWORD dwTool, PAER_TOOL pTool)

Parameters

<table>
<thead>
<tr>
<th>pszFile</th>
<th>Tool File.</th>
</tr>
</thead>
<tbody>
<tr>
<td>dwTool</td>
<td>Which tool to retrieve from file (0-Based).</td>
</tr>
<tr>
<td>pTool</td>
<td>Pointer to AER_TOOL structure to hold tool info.</td>
</tr>
</tbody>
</table>

This function sets the information for the specified tool. An error will occur if `dwTool` is not a valid tool number. A valid tool number is 0..(NumTools-1). 0..(NumTools-1) will update the specified tool in the Tool File. If `dwTool` equals NumTools then the tool is added to the Tool file with `AerToolFileAddTool`.

The name of the Tool file can be retrieved with `AerRegGetFileName` (AERREGID_ToolFile).

See Also

- `AerRegGetFileName`
- `AerToolFileAddTool`
- `AerToolFileDownload`
- `AerToolFileGetTool`
23.6. **AerToolFileAddTool**

AERERR_CODE AerToolFileAddTool (LPCTSTR pszFile, DWORD dwTool, PAER_TOOL pTool)

**Parameters**

- *pszFile*  
  Tool File.
- *dwTool*  
  Which tool to retrieve from file (0-Based).
- *pTool*  
  Pointer to AER_TOOL structure to hold tool info.

This function adds a new tool to the Tool File.

The name of the Tool file can be retrieved with *AerRegGetFileName* (AERREGID_ToolFile).

**See Also**

- *AerRegGetFileName*
- *AerToolFileDownload*
- *AerToolFileRemoveTool*
23.7. AerToolFileRemoveTool

AERERR_CODE AerToolFileRemoveTool (LPCTSTR pszFile, DWORD dwTool)

**Parameters**

- **pszFile**: Tool File.
- **dwTool**: Which tool to retrieve from file (0-Based).

This function removes the specified tool from the Tool File.

The name of the Tool file can be retrieved with `AerRegGetFileName` (AERREGID_ToolFile).

**See Also**

- `AerRegGetFileName`
- `AerToolFileAddTool`
- `AerToolFileDownload`
23.8. **AerToolTableAllocate**

AERERR_CODE AerToolTableAllocate (HAERCTRL hAerCtrl, DWORD dwNumTools)

**Parameters**

- **hAerCtrl** Handle to the axis processor card.
- **dwNumTools** Number of tools that the table will hold.

This function allocates memory on the axis processor for the specified number of tools. All allocated tools are marked as invalid until the tool is properly set with `AerToolTableSetTool`.

**See Also**

- `AerToolFileDownload`
- `AerToolTableFree`
- `AerToolTableSetTool`
23.9. **AerToolTableFree**

AERERR_CODE AerToolTableFree (HAERCTRL hAerCtrl)

**Parameters**

\[ hAerCtrl \]

Handle to the axis processor card.

This function frees the memory on the axis processor associated with the tool table.

**See Also**

AerToolFileDownload
AerToolTableAllocate
23.10. AerToolTableGetNumTools

AERERR_CODE AerToolTableGetNumTools (HAERCTRL hAerCtrl, PDWORD pdwNumTools)

Parameters

\( hAerCtrl \)  Handle to the axis processor card.
\( pdwNumTools \) Number of tools that have been allocated in the tool table.

This function returns the number of tools that have been allocated in the tool table.

See Also

AerToolFileDownload
AerToolTableAllocate
23.11. AerToolTableSetTool

AERERR_CODE AerToolTableSetTool (HAERCTRL hAerCtrl, DWORD dwTool, PAER_TOOL pTool)

Parameters

  hAerCtrl          Handle to the axis processor card.
  dwTool           Which tool to retrieve from tool table (0-Based).
  pTool            Pointer to AER_TOOL structure to hold tool info.

This function sets the information for the specified tool. An error will occur if `dwTool` is
not a valid tool number. A valid tool number is 0..NumToolsAllocated-1.

A tool remains invalid until it has been set in the tool table. The `mFlag` parameter of
`pTool` will reflect AERTOOL_F_VALID on subsequent calls to AerToolTableGetTool.

C Language and LabView Constants

  AERTOOL_F_XXXX

See Also

  AerToolFileDownload
  AerToolTableAllocate
  AerToolTableGetTool
23.12. AerToolTableGetTool

AERERR_CODE AerToolTableGetTool (HAERCTRL hAerCtrl, DWORD dwTool, PAER_TOOL pTool)

Parameters

- **hAerCtrl**: Handle to the axis processor card.
- **dwTool**: Which tool to retrieve from tool table (0-Based).
- **pTool**: Pointer to AER_TOOL structure to hold tool info.

This function returns the information for the specified tool. An error will occur if `dwTool` is not a valid tool number. A valid tool number is 0..NumToolsAllocated-1.

See Also

- AerToolFileDownload
- AerToolTableAllocate
- AerToolTableSetTool
CHAPTER 24: TORQUE FUNCTIONS

In This Section:
- Introduction......................................................... 24-1
- AerTorqueGetMode............................................. 24-2
- AerTorqueSetMode............................................. 24-3

24.1. Introduction

The torque mode functions provide axis control seeking to maintain a constant torque on
an axis, independent of speed and thermal effects. This method of operation can be used
in winding applications where a constant tension must be applied to the material as it is
being collected onto a reel.
24.2. AerTorqueGetMode

AERERR_CODE AerTorqueGetMode (HAERCTRL hAerCtrl, AXISINDEX iAxis, WORD pwMode, WORD pwChannel, FLOAT pdAmpsVel, FLOAT pdAmpsTemp, WORD pwNomTemp);

Declare Function AerTorqueGetMode Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal iAxis As Long, ByRef pwMode As Integer, ByRef pwChannel As Integer, ByRef pdAmpsVel As Single, ByRef pdAmpsTemp As Single, ByRef pwNomTemp As Integer) As Long

Parameters

hAerCtrl Handle to the axis processor card.
iAxis Axis index (see constants below).
pwMode Returns 0/1 to disable/enable torque mode.
pwChannel Returns A/D channel for temperature sensor (see constants below).
pdAmpsVel Returns velocity compensation term expressed in D/A bits per count/sec.
pdAmpsTemp Returns temperature compensation term.
pwNomTemp Returns nominal operating temperature expressed in A/D bits.

This function returns the state of the constant torque mode of an axis. The parameters of the function are as follows:

pwMode 0/1 to indicates torque mode
pwChannel A/D channel for temperature sensor (see constants below)
  UNIDEX 600 card ch’s 1-4
  Encoder Exp. card 1 ch’s 5-8
  Encoder Exp. card 2 ch’s 9-12
  Encoder Exp. card 3 ch’s 13-16
pdAmpsVel Velocity compensation term expressed in D/A bits per count/sec.
  For example, given a full scale current output of 50 amps and a desired velocity compensation of 0.01 amp / (cnt/sec.), the following constant would be used:
  \[32767 \times 0.01 / 50 = 6.5534\]
pdAmpsTemp Temperature compensation in units of (% current increase / temp_change) / (100 * A/D_bits / degree_celsius).
  For example, given a bipolar 12 bit +/- 10 volt A/D, with 0 - 10 volts corresponding to a 125 degree increase in temperature and a change in the motor torque constant of 17% over a 75 degree increase in temperature:
  \[(17 / 75) / (100 \times 2047 / 125) = 0.000138414\]
pwNomTemp Nominal (room) operating temperature expressed in A/D bits. The wNomTemp value can be determined by reading the A/D card at the nominal operating temperature.

C Language and LabView Constants
AXISINDEX_#
ANALOGINDEX_#

VB Constants
aerAxisIndex#
aerAnalogIndex#

See Also
AerTorqueSetMode
24.3. AerTorqueSetMode

AERERR_CODE AerTorqueSetMode (HAERCTRL hAerCtrl, AXISINDEX iAxis,
WORD wMode, WORD wChannel, FLOAT dAmpsVel, FLOAT dAmpsTemp, WORD wNomTemp);

Declare Function AerTorqueSetMode Lib “AERSYS.DLL” (ByVal hAerCtrl As Long,
ByVal iAxis As Long, ByVal wMode As Integer, ByVal wChannel As Integer, ByVal dAmpsVel As Single,
ByVal dAmpsTemp As Single, ByVal wNomTemp As Integer) As Long

Parameters

- **hAerCtrl**: Handle to the axis processor card.
- **iAxis**: Axis index (see constants below).
- **wMode**: 0/1 to disable/enable torque mode.
- **wChannel**: A/D channel for temperature sensor (see constants below).
- **dAmpsVel**: Velocity compensation term expressed in D/A bits per count/sec.
- **dAmpsTemp**: Temperature compensation term.
- **wNomTemp**: Nominal operating temperature expressed in A/D bits.

This function allows a constant output torque to be applied to an axis by compensating for changes in the motor torque constant due to thermal effects and frictional losses. This function has applications in winding where a constant tension must be applied while winding. Setting the IMAX axis parameter of the desired axis attains the torque set point. The value of **IMAX** can be calculated as follows:

\[
IMAX = \frac{32767}{\text{full\_scale\_current}} \times \frac{\text{desired\_torque}}{\text{motor\_torque\_constant}}
\]

where:

- **full\_scale\_current** is the current output from the amplifier for a 10 volt command from the DAC. **Motor\_torque\_constant** is in units of torque per ampere of current.

The parameters of the function are as follows:

- **wMode**: 0/1 to disable/enable torque mode
- **wChannel**: A/D channel for temperature sensor
  - UNIDEX 600 card channels 1-4
  - Encoder Exp. card 1 channels 5-8
  - Encoder Exp. card 2 channels 9-12
  - Encoder Exp. card 3 channels 13-16

- **dAmpsVel**: Velocity compensation term expressed in D/A bits per count/sec
  
  For example, given a full scale current output of 50 amps and a desired velocity compensation of 0.01 amp / (count/sec.), the following constant would be used:

\[
32767 \times .01 / 50 = 6.5534
\]

- **dAmpsTemp**: Temperature compensation in units of (% current increase / temp\_change) / (100 * A/D\_bits / degree\_celsius).
For example given a bipolar 12 bit +/- 10 volt A/D, with 0 - 10 volts corresponding to a 125 degree increase in temperature and a change in the motor torque constant of 17% over a 75 degree increase in temperature:

\[
\frac{17}{75} \div \left( \frac{100 \times 2047}{125} \right) = 0.000138414
\]

\( w_{\text{NomTemp}} \) Nominal (room) operating temperature expressed in A/D bits. The \( w_{\text{NomTemp}} \) value can be determined by reading the A/D card at the nominal operating temperature.

### C Language and LabView Constants

- `AXISINDEX_#`
- `ANALOGINDEX_#`

### VB Constants

- `aerAxisIndex#`
- `aerAnalogIndex#`

### See Also

- `AerTorqueGetMode`
CHAPTER 25:  UTILITY FUNCTIONS

In This Section:
- Introduction ................................................................. 25-1
- AerUtilAlignMaster ....................................................... 25-2
- AerUtilHitWatchdogTimer .............................................. 25-3
- AerUtilMotorSet ............................................................ 25-4
- AerUtilSet1khzServoLoop ............................................. 25-5
- AerUtilSet4khzServoLoop ............................................. 25-6

25.1. Introduction

The AerUtil group serves as a miscellaneous category for Aerotech library functions. These functions accomplish a variety of tasks.
25.2. AerUtilAlignMaster

AERERR_CODE AerUtilAlignMaster (HAERCTRL hAerCtrl, AXISINDEX iAxis);

Declare Function AerUtilAlignMaster Lib “AERSYS.DLL” (ByVal hAerCtrl As Long,
ByVal iAxis As Long) As Long

Parameters
  hAerCtrl   Handle to the axis processor card.
iAxis       The axis number to align (see constants below).

This function sets the MASTERPOS axis parameter for the axis equal to its RAWPOS parameter.

C Language and LabView Constants
  AXISINDEX_XXXX

VB Constants
  aerAxisIndex#
25.3. **AerUtilHitWatchdogTimer**

AERERR_CODE AerUtilHitWatchdogTimer (HAERCTRL *hAerCtl);

Declare Function AerUtilHitWatchdogTimer Lib “AERSYS.DLL” (ByVal *hAerCtrl As Long) As Long

**Parameters**

* **hAerCtrl**  Handle to the axis processor card.

This function will strobe the watchdog timer preventing a fatal error from occurring and shutting down the axis processor card.
25.4. AerUtilMotorSet

AERERR_CODE AerUtilMotorSet (HAERCTRL hAerCtrl, AXISINDEX iAxis, WORD wSet);

Declare Function AerUtilMotorSet Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal iAxis As Long, ByVal wSet As Integer) As Long

Parameters
   hAerCtrl Handle to the axis processor card.
   iAxis An physical axis index, representing the torque vector to set (see constants below).
   wSet The electrical angle to output (360° = 1,024).

This function will enable the specified drive and set the Sin and Sin+120 current command outputs to the specified angle causing the AC brushless motor’s rotor to rotate to the specified electrical angle. The amplitude of the torque vector is determined by the setting of its IAVGLIMIT axis parameter. This parameter should be set before calling this function by the AerParmAxisSetValue function to a value less than or equal to the motor’s continuous current rating to ensure the motor is not damaged. This function is useful in determining a required commutation offset for proper phasing of the feedback device. The electrical vector may be removed by disabling the drive. This can be accomplished with the AerParmAxisSetValue function.

C Language and LabView Constants
   AXISINDEX_XXXX

VB Constants
   aerAxisIndex#

See Also
   AerConfig
   AerParmAxisSetValue
25.5. **AerUtilSet1khzServoLoop**

AERERR_CODE AerUtilSet1khzServoLoop (HAERCTRL hAerCtrl);

Declare Function AerUtilSet1khzServoLoop Lib “AERSYS.DLL” (ByVal hAerCtrl As Long) As Long

**Parameters**

*hAerCtrl*  Handle to a axis processor communication channel.

This function will define the servo loop update rate all of the axes to be 1,000 Hertz (1 msec.) in the same manner as the G130 RS274 command. This servo loop is responsible for sampling the feedback from the motor and calculating the new motor command from the axis gains.

See the Enable1kHzservo global parameter in the U600MMI.hlp file for more information.

**See Also**

*AerUtilSet4khzServoLoop*
25.6. **AerUtilSet4khzServoLoop**

AERERR_CODE AerUtilSet4khzServoLoop (HAERCTRL hAerCtrl);

Declare Function AerUtilSet4khzServoLoop Lib "AERSYS.DLL" (ByVal hAerCtrl As Long) As Long

**Parameters**

- **hAerCtrl** Handle to the axis processor card.

This function will define the servo loop update rate to be 4000 Hertz (.25 Msec.) in the same manner as the G131 RS274 command. This servo loop is responsible for sampling the feedback from the motor and calculating the new motor command from the axis gains.

See the Enable1kHzservo global parameter in the U600MMI.hlp file for more information.

**See Also**

- AerUtilSet1khzServoLoop

∇ ∇ ∇
CHAPTER 26: VARIABLE FUNCTIONS

26.1. Introduction

The Variable functions read and write to the user variables within the UNIDEX 600 Series controllers.
26.2. AerVarAnalogGetDouble

AERERR_CODE AerVarAnalogGetDouble (HAERCTRL hAerCtrl, ANALOGINDEX iAnalog, PDOUBLE pdValue);

Declare Function AerVarAnalogGetDouble Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal iAnalog As Long, ByRef pdValue As Double) As Long

Parameters

- **hAerCtrl** Handle to axis processor card.
- **iAnalog** Which analog input to read (see constants below).
- **pdValue** Pointer to variable to hold value of analog input.

This function reads the current value of the specified analog input from the UNIDEX 600 card. The U600 card has 4 analog inputs. Four additional analog inputs can be accessed on each 4-EN PC (encoder card) card.

C Language and LabView Constants

- ANALOGINDEX_XXXX

VB Constants

- aerAnalogIndex#
26.3. **AerVarGlobalGetDouble**

AERERR_CODE AerVarGlobalGetDouble (HAERCTRL hAerCtrl, DWORD dwNum, PDOUBLE pfdValue);

Declare Function AerVarGlobalGetDouble Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal dwNum As Long, ByRef pfdValue As Double) As Long

**Parameters**

- **hAerCtrl** Handle to the axis processor card.
- **dwNum** Index of global variables to read (zero-based).
- **pfdValue** Address of variable to receive value.

This function will return the value of a global double precision variable within the controller. The number of global variables is determined by the global parameter ‘NumGlobalDoubles’, which has a default value of 10. The range of global double variables is that of a double precision number, +/- 1.7E308. These global variables are numbered 0 through NumGlobalDoubles-1.

**See Also**

- AerVarGlobalSetDouble
26.4. AerVarGlobalGetString

AERERR_CODE AerVarGlobalGetString (HAERCTRL hAerCtrl, DWORD dwNum, LPSTR pszString);

Declare Function AerVarGlobalGetString Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal dwNum As Long, ByRef pszString As String) As Long

Parameters

- **hAerCtrl**: Handle to the axis processor card.
- **dwNum**: Index of global variables (zero-based).
- **pszString**: Address of variable to receive string.

All string variables in MS Visual Basic, passed by reference (ByRef), must be declared as fixed length strings within your program, long enough to hold the string value returned by the function. Also, those string variables which are passed, with another parameter indicating the length of the string variable, must also be fixed length strings, otherwise, you would not be able to pass the length of the string. For example, to declare a fixed length string:

```vbnet
DIM sGlobStr as STRING * 50 ; 50 characters long
```

This function will return the contents of a global string variable within the controller. The number of global string variables is determined by the global parameter ‘NumGlobalStrings’, which has a default value of 10. The maximum length of global string variables is 127 characters. These string variables are numbered 0 through NumGlobalStrings-1.

See Also

- AerVarGlobalSetString
26.5. **AerVarGlobalSetDouble**

AERERR_CODE AerVarGlobalSetDouble (HAERCTRL hAerCtrl, DWORD dwNum, DOUBLE fdValue);

Declare Function AerVarGlobalSetDouble Lib "AERSYS.DLL" (ByVal hAerCtrl As Long, ByVal dwNum As Long, ByVal fdValue As Double) As Long

**Parameters**

- **hAerCtrl**  
  Handle to the axis processor card.
- **dwNum**  
  Index of global variables to set (zero-based).
- **fdValue**  
  Double value to write.

This function will set the value of a global double precision variable within the controller. The number of global variables is determined by the global parameter ‘NumGlobalDoubles’, which has a default value of 10. The range of global double variables is that of a double precision number, +/- 1.7E308. These global variables are numbered 0 through NumGlobalDoubles-1.

**See Also**

- *AerVarGlobalGetDouble*
26.6. AerVarGlobalSetString

AERERR_CODE AerVarGlobalSetString (HAERCTRL hAerCtrl, DWORD dwNum, LPSTR pszString);

Declare Function AerVarGlobalSetString Lib "AERSYS.DLL" (ByVal hAerCtrl As Long, ByVal dwNum As Long, ByVal pszString As String) As Long

Parameters
- hAerCtrl: Handle to the axis processor card.
- dwNum: Index of global string variables (zero-based).
- pszString: Address of string to write.

This function will set the value of a global string variable within the controller. The number of global string variables is determined by the global parameter ‘NumGlobalStrings’, which has a default value of 10. The maximum length of global string variables is 127 characters. These string variables are numbered 0 through NumGlobalStrings-1.

See Also
- AerVarGlobalGetString
26.7. AerVarProgramGetDouble

AERERR_CODE AerVarProgramGetDouble (HAERCTRL hAerCtrl, TASKINDEX iTask, DWORD dwNum, PDOUBLE pfdValue);

Declare Function AerVarProgramGetDouble Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal iTask As Double, ByVal dwNum As Long, ByRef pfdValue As Double) As Long

Parameters

- **hAerCtrl**: Handle to the axis processor card.
- **iTask**: Task index (see constants below).
- **dwNum**: Index of variable (zero-based).
- **pfdValue**: Address of variable to receive value.

This function will return the value of a user-defined double precision variable, defined within a user CNC program. The number of variables is determined by the number defined by the user in their CNC program, by the DVAR command, which is limited by the amount of free memory on the controller. The range of variables is that of a double precision number, +/- 1.7E\(^{308}\). These variables are numbered 0 through the maximum number -1. These variables will be sequentially numbered in the order in which the user defined them.

C Language and LabView Constants

- **TASKINDEX_XXXX**

VB Constants

- **aerTaskIndex#**

See Also

- AerVarProgramSetDouble
- AerCompilerGetProgVarTotal
- AerCompilerGetProgVarByName
- AerCompilerGetProgVarByNumber
26.8. AerVarProgramSetDouble

AERERR_CODE AerVarProgramSetDouble (HAERCTRL hAerCtrl, TASKINDEX iTask, DWORD dwNum, DOUBLE fdValue);

Declare Function AerVarProgramSetDouble Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal iTask As Double, ByVal dwNum As Long, ByVal fdValue As Double) As Long

Parameters

- **hAerCtrl** Handle to the axis processor card.
- **iTask** Task index (see constants below).
- **dwNum** Index of program variable number to set (zero-based).
- **fdValue** Double value to write.

This function will set the value of a user-defined double precision variable, defined within a user CNC program. The number of variables is determined by the number defined by the user in their CNC program, by the DVAR command, which is limited by the amount of free memory on the axis processor. The range of variables is that of a double precision number, +/- 1.7E308. These variables are numbered 0 through the maximum number -1. These variables will be sequentially numbered in the order in which the user defined them.

C Language and LabView Constants

- **TASKINDEX_XXXX**

VB Constants

- aerTaskIndex#

See Also

- AerVarProgramGetDouble
- AerCompilerGetProgVarTotal
- AerCompilerGetProgVarByName
- AerCompilerGetProgVarByNumber
26.9. AerVarStackGetDouble

AERERR_CODE AerVarStackGetDouble (HAERCTRL hAerCtrl, TASKINDEX iTask,
CSPARMINDEX iCSParm, PDOUBLE pfdValue);

Declare Function AerVarStackGetDouble Lib “AERSYS.DLL” (ByVal hAerCtrl As
Long, ByVal iTask As Double, ByVal iCSParm As Double, ByRef
pfdValue As Double) As Long

Parameters
- **hAerCtrl**: Handle to the axis processor card.
- **iTask**: Task index (see constants below).
- **iCSParm**: Index of call stack parameters (zero-based).
- **pfdValue**: Address of variable to receive value.

This function will return the value of a stack double precision variable used by the user’s
CNC program for subroutine and program calls on the axis processor. The number of
stack variables is determined by the task parameter ‘MaxCallStack’, which has a default
value of 10. The range of stack double variables is that of a double precision number, +/-
1.7E308. These stack variables are numbered 0 through MaxCallStack-1.

C Language and LabView Constants
- **TASKINDEX_XXXX**

VB Constants
- **aerTaskIndex#**

See Also
- AerVarStackSetDouble
26.10. AerVarStackSetDouble

AERERR_CODE AerVarStackSetDouble (HAERCTRL hAerCtrl, TASKINDEX iTask, CSPARMINDEX iCSParm, DOUBLE fdValue);

Declare Function AerVarStackSetDouble Lib "AERSYS.DLL" (ByVal hAerCtrl As Long, ByVal iTask As Double, ByVal iCSParm As Double, ByVal fdValue As Double) As Long

Parameters

- **hAerCtrl** Handle to the axis processor card.
- **iTask** Task index (see constants below).
- **iCSParm** Index of call stack parameters (zero-based).
- **fdValue** Double value to write.

This function will set the value of a stack double precision variable used by the user’s CNC program for subroutine and program calls on the axis processor. The number of stack variables is determined by the task parameter ‘MaxCallStack’, which has a default value of 10. The range of stack double variables is that of a double precision number, +/-1.7E308. These stack variables are numbered 0 through MaxCallStack-1.

C Language and LabView Constants

- **TASKINDEX_XXXX**

VB Constants

- **aerTaskIndex#**

See Also

- AerVarStackGetDouble
26.11. AerVarTaskGetDouble

AERERR_CODE AerVarTaskGetDouble (HAERCTRL hAerCtrl, TASKINDEX iTask, 
DWORD dwNum, PDOUBLE pfdValue);

Declare Function AerVarTaskGetDouble Lib “AERSYS.DLL” (ByVal hAerCtrl As 
Long, ByVal iTask As Double, ByVal dwNum As Long, ByRef 
pfdValue As Double) As Long

Parameters
    hAerCtrl       Handle to the Axis Processor card.
    iTask          Task index (see constants below).
    dwNum          Index of task variables to read (zero-based).
    pfdValue       Address of variable to receive data.

This function will return the value of a task double precision variable on the specified 
task. The number of task variables is determined by the task parameter 
‘NumTaskDoubles’, which has a default value of 10. The range of task double variables is 
that of a double precision number, +/- 1.7E308. These task variables are numbered 0 
through NumTaskDoubles-1.

C Language and LabView Constants
    TASKINDEX_XXX

VB Constants
    aerTaskIndex#

See Also
    AerVarTaskSetDouble
26.12. AerVarTaskGetString

AERERR_CODE AerVarTaskGetString (HAERCTRL hAerCtrl, TASKINDEX iTask, DWORD dwNum, LPSTR pszString);

Declare Function AerVarTaskGetString Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal iTask As Double, ByVal dwNum As Long, ByRef pszString As String) As Long

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hAerCtrl</td>
<td>Handle to the axis processor card.</td>
</tr>
<tr>
<td>iTask</td>
<td>Task index (see constants below).</td>
</tr>
<tr>
<td>dwNum</td>
<td>Index of task string variables (zero-based).</td>
</tr>
<tr>
<td>pszString</td>
<td>Address of variable to receive string.</td>
</tr>
</tbody>
</table>

All string variables in MS Visual Basic, passed by reference (ByRef), must be declared as fixed length strings within your program, long enough to hold the string value returned by the function. Also, those string variables which are passed, with another parameter indicating the length of the string variable, must also be fixed length strings, otherwise, you would not be able to pass the length of the string. For example, to declare a fixed length string:

```
DIM sGlobStr as STRING * 50 ; 50 characters long
```

This function will return the contents of a task string variable used within a user CNC program. The number of task string variables is determined by the task parameter ‘NumTaskStrings’, which has a default value of 10. The maximum length of task string variables is 127 characters. These string variables are numbered 0 through NumTaskStrings-1.

See Also

AerVarTaskSetString
26.13. AerVarTaskSetDouble

AERERR_CODE AerVarTaskSetDouble (HAERCTRL hAerCtrl, TASKINDEX iTask,
   DWORD dwNum, DOUBLE fdValue);

Declare Function AerVarTaskSetDouble Lib “AERSYS.DLL” (ByVal hAerCtrl As Long,
   ByVal iTask As Double, ByVal dwNum As Long, ByVal fdValue As Double) As Long

Parameters
   hAerCtrl     Handle to the axis processor card.
   iTask        Task index (see constants below).
   dwNum        Index of task variables to set (zero-based).
   fdValue      Double value to write.

This function will set the value of a task double precision variable used on the specified
task. The number of task variables is determined by the task parameter
‘NumTaskDoubles’, which has a default value of 10. The range of task double variables is
that of a double precision number, +/- 1.7E308. These task variables are numbered 0
through NumTaskDoubles-1.

C Language and LabView Constants
   TASKINDEX_XXX

VB Constants
   aerTaskIndex#

See Also
   AerVarTaskGetDouble

AERERR_CODE AerVarTaskSetString (HAERCTRL hAerCtrl, TASKINDEX iTask, DWORD dwNum, LPSTR pszString);

Declare Function AerVarTaskSetString Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal iTask As Double, ByVal dwNum As Long, ByVal pszString As String) As Long

Parameters

  hAerCtrl      Handle to the axis processor card.
  iTask         Task index (see constants below).
  dwNum         Index of task string variables (zero-based).
  pszString     Address of string to write.

This function will set the value of a task string variable used within a user CNC program on the axis processor. The number of task string variables is determined by the task parameter ‘NumTaskStrings’, which has a default value of 10. The maximum length of task string variables is 127 characters. These string variables are numbered 0 through NumTaskStrings-1.

C Language and LabView Constants

  TASKINDEX_XXXX

VB Constants

  aerTaskIndex#

See Also

  AerVarTaskGetString
CHAPTER 27:  VERSION FUNCTIONS

27.1. Introduction

The version functions provide information about the current version of the UNIDEX 600 Series controller firmware and library (Aer Sys.DLL) executing on the user’s system.
27.2. AerVerGetBootVersion

AERERR_CODE AerVerGetBootVersion (HAERCTRL hAerCtrl, PAER_VERSION pVersion);

Parameters

  hAerCtrl   Handle to the axis processor card.
  pVersion   Pointer to a structure that version data is written to.

This function returns the identical result as AerVerGetImgVersion. The image (firmware) must be running for this function to return successfully.

See Also

  AerVerGetImgVersion
  AerVerGetCpuStatus
27.3. AerVerGetCpuStatus

AERERR_CODE AerVerGetCpuStatus (HAERCTRL hAerCtrl, PWORD pwStat);

Declare Function AerVerGetCpuStatus Lib “AERSYS.DLL” (ByVal hAerCtrl As Long,
ByRef pwStat As Short) As Long

Parameters
   - hAerCtrl Handle to the axis processor card.
   - pwStat Pointer to a WORD to receive status.

This function returns a status value indicating the current state of the axis processor
board. These states are constants as defined below.

C Language and LabView Constants
   AER_CPUSSTATXXX

VB Constants
   aerCPUStatusXXX

See Also
   AerSysDownload
27.4. AerVerGetDrvVersion

AERERR_CODE AerVerGetDrvVersion (PAER_VERSION pVersion);

AERERR_CODE AerVerGetDrvVersionEx (HAERCTRL hAerCtrl, PWORD pwUnidex, PWORD pwMajor, PWORD pwMinor, PWORD pwBuild);

Declare Function AerVerGetDrvVersionEx Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByRef pwUnidex As Integer, ByRef pwMajor As Integer, ByRef pwMinor As Integer, ByRef pwBuild As Integer) As Long

AERERR_CODE AerVerGetDrvVersionString (HAERCTRL hAerCtrl, LPTSTR pszVersion);

Declare Function AerVerGetDrvVersionString Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByRef pszVersion As String) As Long

Parameters

- pVersion: Pointer to a structure that version data is written.
- pwUnidex: Pointer to WORD to hold UNIDEX Number.
- pwMajor: Pointer to WORD to hold Major Number.
- pwMinor: Pointer to WORD to hold Minor Number.
- pwBuild: Pointer to WORD to hold Build Number.
- pszVersion: Pointer to string to hold Version string.

All string variables in MS Visual Basic, passed by reference (ByRef), must be declared as fixed length strings within your program, long enough to hold the string value returned by the function. Also, those string variables which are passed, with another parameter indicating the length of the string variable, must also be fixed length strings, otherwise, you would not be able to pass the length of the string. For example, to declare a fixed length string:

```
DIM sGlobStr as STRING * 50 ; 50 characters long
```

This function returns the version number of the device driver.

This may not be the same device driver as seen on the hard drive, unless a reboot has been performed since the new device driver was placed on the hard drive.

The “String” form of this function returns the version information as a NULL terminated string in the pszVersion parameter.

See Also
AerVerGetImgVersion

Example
Samples\Lib\AexSys.C
27.5. AerVerGetImgVersion

AERERR_CODE AerVerGetImgVersion (HAERCTRL hAerCtrl, PAER_VERSION pVersion);

AERERR_CODE AerVerGetImgVersionEx (HAERCTRL hAerCtrl, PWORD pwUnidex, PWORD pwMajor, PWORD pwMinor, PWORD pwBuild);

Declare Function AerVerGetImgVersionEx Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByRef pwUnidex As Integer, ByRef pwMajor As Integer, ByRef pwMinor As Integer, ByRef pwBuild As Integer) As Long

AERERR_CODE AerVerGetImgVersionString (HAERCTRL hAerCtrl, LPTSTR pszVersion);

Declare Function AerVerGetImgVersionString Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByRef pszVersion As String) As Long

Parameters

hAerCtrl Handle to the axis processor card.
pVersion Pointer to a structure that version data is written.
pwUnidex Pointer to WORD to hold UNIDEX Number.
pwMajor Pointer to WORD to hold Major Number.
pwMinor Pointer to WORD to hold Minor Number.
pwBuild Pointer to WORD to hold Build Number.
pszVersion Pointer to string to hold Version string.

All string variables in MS Visual Basic, passed by reference (ByRef), must be declared as fixed length strings within your program, long enough to hold the string value returned by the function. Also, those string variables which are passed, with another parameter indicating the length of the string variable, must also be fixed length strings, otherwise, you would not be able to pass the length of the string. For example, to declare a fixed length string;

```
DIM sGlobStr as STRING * 50 ; 50 characters long
```

This function returns version information about the axis processor image (firmware) code that is currently running on the axis processor.

This may not be the same image present on the hard drive, unless a download has been performed after the image was placed on the hard drive.

The “String” form of this function returns the version information as a (NULL terminated) string in the pszVersion parameter.

See Also

AerVerGetBootVersion

WARNING

The image must be running for this function to return successfully.
27.6. AerVerGetLibVersion

AERERR_CODE AerVerGetLibVersion (PAER_VERSION pVersion);
AERERR_CODE AerVerGetLibVersionEx (PWORD pwUnidex, PWORD pwMajor, PWORD pwMinor, PWORD pwBuild);

Declare Function AerVerGetLibVersionEx Lib "AERSYS.DLL" (ByRef pwUnidex As Integer, ByRef pwMajor As Integer, ByRef pwMinor As Integer, ByRef pwBuild As Integer) As Long
AERERR_CODE AerVerGetLibVersionString (LPTSTR pszVersion);
Declare Function AerVerGetLibVersionString Lib "AERSYS.DLL" (ByRef pszVersion As String) As Long

Parameters
- pVersion: Pointer to a structure that version data is written.
- pwUnidex: Pointer to WORD to hold UNIDEX Number.
- pwMajor: Pointer to WORD to hold Major Number.
- pwMinor: Pointer to WORD to hold Minor Number.
- pwBuild: Pointer to WORD to hold Build Number.
- pszVersion: Pointer to string to hold Version string.

All string variables in MS Visual Basic, passed by reference (ByRef), must be declared as fixed length strings within your program, long enough to hold the string value returned by the function. Also, those string variables which are passed, with another parameter indicating the length of the string variable, must also be fixed length strings, otherwise, you would not be able to pass the length of the string. For example, to declare a fixed length string:

```
DIM sGlobStr as STRING * 50 ; 50 characters long
```

This function returns the version number of the library (AerSys.DLL) that is currently running on the front-end processor.

This may not be the same DLL library that is on the hard drive, unless the application was started (and the DLL loaded into memory) after the library was installed on the hard drive.

The “String” form of this function returns the version information as a (NULL terminated) string in the pszVersion parameter.

See Also
AerVerGetImgVersion

Example
samples\lib\AexSys.C
27.7. **AerVerGetOS**

AERERR_CODE AerVerGetOS (PDWORD pdwOS);

Declare Function AerVerGetOS Lib “AERSYS.DLL” (ByRef pdwOS As Long) As Long

**Parameters**

- `pdwOS` Pointer to a word that will return a constant indicating the operating system (see constants below).

This function returns the current operating system (i.e. Windows 95, Windows NT). The value is an AEROS_xxxx constant.

**C Language and LabView Constants**

\[ \texttt{AEROS\_XXXX} \]

**VB Constants**

\[ \texttt{aerOS\_XXXX} \]

**Example**

Samples\Lib\AexSys.C
27.8. AerVerStringToVersion

AERERR_CODE AerVerStringToVersion (LPTSTR pszVersion, PAER_VERSION pVersion);

**Parameters**

- `pszVersion` String with version information.
- `pVersion` Pointer to structure to hold Aerotech version information.

This function converts a version string to the PAER_VERSION version structure. The string should be in an XXX.YYY.ZZ.UU format where X represents the UNINDEX number, Y represents the major number, Z represents the minor number and U represents the build number.
27.9. **AerVerToString**

AERERR_CODE AerVerToString (PAER_VERSION pVersion, LPTSTR pszVersion);

**Parameters**

- `pVersion` Pointer to version Aerotech information.
- `pszVersion` String to hold version information.

The `AerVerToString` function converts a PAER_VERSION version structure to a string. The format is XXX.YY.ZZ.UUU where X represents the UNIDEX Number, Y represents the Major Number, Z represents the Minor Number, and U represents the Build Number.

∇ ∇ ∇
CHAPTER 28: VIRTUAL I/O FUNCTIONS

28.1. Introduction

The Virtual I/O functions read and write to the virtual I/O of the UNIDEX 600 Series controllers. The virtual I/O space on the UNIDEX 600 card is comprised of four separate I/O spaces: 512 binary inputs, 512 binary outputs, 128 register inputs, and 128 register outputs. The U600 allows you to interact with these spaces in different block sizes using three function name styles: regular, “Ex”, and “Array.” For example, AerVirtGetBinaryInput() = regular, AerVirtGetInputByteEx() = “Ex”, and AerVirtSetRegisterInputArray() = “Array”.

For information on connecting Virtual I/O to hardware, refer to the U600 User’s Guide (Chapter 2, in the Digital I/O section).

A regular function (one that doesn’t end in “Ex” or “Array”) reads I/O by bit (also called binary) or register size (a block of 16 bits). The “Ex” functions give the user the limited ability to define the block size (Bytes, Words, or Dwords – 8, 16, or 32 bits, respectively). The “Array” function gives the user the ability to define even larger block sizes to get or set.

For example, AerVirtGetBinaryInput() reads a bit from the binary input space while AerVirtGetInputByteEx() reads from the same binary input space and returns a block of eight consecutive bits (a byte) instead. AerVirtGetRegisterInputArray returns a series of (16 bit) registers from the register input space. The number of registers being determined by the parameters to the function.
28.2. **AerVirtGetBinaryInput**

AERERR_CODE AerVirtGetBinaryInput (HAERCTRL hAerCtrl, DWORD dwNum, PBOOL pbValue);

Declare Function AerVirtGetBinaryInput Lib "AERSYS.DLL" (ByVal hAerCtrl As Long, ByVal dwNum As Long, ByRef pbValue As Long) As Long

**Parameters**

- **hAerCtrl** Handle to the axis processor card.
- **dwNum** Binary input bit number to read.
- **pbValue** Address of variable to receive value of the specified bit.

This function will read the state of a binary input (1 bit) from the UNIDEX 600 Series controller’s virtual I/O space. There are 512 binary inputs, numbered 0 through 511.

**See Also**

- AerVirtGetBinaryInputByteEx

**Example**

Samples\Lib\VisualBasic\RunPgm.vbp
28.3. AerVirtGetBinaryInputByteEx

AERERR_CODE AerVirtGetBinaryInputByteEx (HAERCTRL hAerCtrl, DWORD dwByte, PBYTE pbyData);

Declare Function AerVirtGetBinaryInputByteEx Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal dwByte As Long, ByRef pbyData As Byte) As Long

Parameters

- **hAerCtrl**  Handle to the axis processor card.
- **dwByte**  Binary input byte number.
- **pbyData**  Address of variable to receive value of the specified byte.

This function will read 8 bits (1 byte) of data from the UNIDEX 600 Series controller’s virtual I/O space. There are 512 binary inputs, numbered 0 through 511, contained in bytes 0 through 64. Byte 0 would return the values of binary inputs 0 through 7, byte 1 would return the values of binary inputs 8 through 15, etc.

See Also

- AerVirtGetBinaryInput
## 28.4. AerVirtGetBinaryInputDWordEx

AERERR_CODE AerVirtGetBinaryInputDWordEx (HAERCTRL hAerCtrl, DWORD dwDWord, PDWORD pdwData);

Declare Function AerVirtGetBinaryInputDWordEx Lib "AERSYS.DLL" (ByVal hAerCtrl As Long, ByVal dwDWord As Long, ByRef pdwData As Long) As Long

**Parameters**

- `hAerCtrl` Handle to axis processor card.
- `dwDWord` Which binary input DWORD (32 bit quantity) to read.
- `pdwData` Pointer to a DWORD to hold 32 bits of data.

This function reads 32-bits (1 DWORD) of data from the virtual input space.

DWORD 0 would return the values of binary inputs 0 – 31.
DWORD 1 would return the values of binary inputs 32 – 63, etc.
### 28.5. AerVirtGetBinaryInputWordArray

AERERR_CODE AerVirtGetBinaryInputWordArray (HAERCTRL hAerCtrl, DWORD dwStartWord, PWORD pwArray, DWORD dwNumWords);

Declare Function AerVirtGetBinaryInputWordArray Lib "AERSYS.DLL" (ByVal hAerCtrl As Long, ByVal dwStartWord As Long, ByRef pwArray As Integer, ByVal dwNumWords As Long) As Long

#### Parameters
- **hAerCtrl**: Handle to axis processor card
- **dwStartWord**: Which binary input WORD (16 bit quantity) to read from.
- **pwArray**: An array of 16 bit data words to hold the binary inputs.
- **dwNumWords**: The number of 16 bit data words to read.

This function reads the specified number of binary input words from the Virtual Input space into the given array space.

WORD 0 would return the values of binary inputs 0 – 15.
WORD 1 would return the values of binary inputs 16 – 31, etc.
28.6. AerVirtGetBinaryInputWordEx

AERERR_CODE AerVirtGetBinaryInputWordEx (HAERCTRL hAerCtrl, DWORD dwWord, PWORD pwData);

Declare Function AerVirtGetBinaryInputWordEx Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal dwWord As Long, ByRef pwData As Integer) As Long

Parameters

- **hAerCtrl**: Handle to axis processor card.
- **dwWord**: Which binary input WORD (16 bit quantity) to read.
- **pwData**: Pointer to a WORD to hold 16 bits of data.

This function reads 16 bits (1 WORD) of data from the virtual input space. The valid binary input words are 0-31 (where 0 returns the value of bits 0-15, 1 returns the value of bits 16-31, 2 returns the value of bits 32-47, etc.).
28.7. AerVirtGetBinaryIO

AERERR_CODE AerVirtGetBinaryIO (HAERCTRL hAerCtrl,
  PAERVIRT_BINARY_DATA pInputs,
  PAERVIRT_BINARY_DATA pOutputs);

Parameters

  hAerCtrl    Handle to axis processor card.
  pInputs     Pointer to AERVIRT_BINARY_DATA to hold the binary input data.
  pOutputs    Pointer to AERVIRT_BINARY_DATA to hold the binary output data.

This function retrieves the current state of all the virtual binary inputs and outputs on the
U600 card. One of the parameters, pInputs or pOutputs, may be NULL if both are not
desired.
28.8. **AerVirtGetBinaryOutput**

AERERR_CODE AerVirtGetBinaryOutput (HAERCTRL hAerCtrl, DWORD dwNum, PBOOL pbValue);

Declare Function AerVirtGetBinaryOutput Lib "AERSYS.DLL" (ByVal hAerCtrl As Long, ByVal dwNum As Long, ByRef pbValue As Long) As Long

**Parameters**

- **hAerCtrl** Handle to the axis processor card.
- **dwNum** Binary output bit number to read.
- **pbValue** Address of variable to receive value of the specified bit.

This function will read the state of a binary output (1 bit) from the UNIDEX 600 Series controller’s virtual I/O space. There are 512 binary outputs, numbered 0 through 511.

**See Also**

- *AerVirtGetBinaryOutputByteEx*
28.9. AerVirtGetBinaryOutputByteEx

AERERR_CODE AerVirtGetBinaryOutputByteEx (HAERCTRL hAerCtrl, DWORD dwByte, PBYTE pbyData);

Declare Function AerVirtGetBinaryOutputByteEx Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal dwByte As Long, ByRef pbyData As Byte) As Long

Parameters
  hAerCtrl       Handle to the axis processor card.
  dwByte         Binary output byte number.
  pbyData        Address of variable to receive value of the specified byte.

This function will read 8 bits (1 byte) of data from the UNIDEX 600 Series controller’s virtual I/O space. There are 512 binary outputs, numbered 0 through 511, contained in bytes 0 through 64. Byte 0 would return the values of binary outputs 0 through 7, byte 1 would return the values of binary outputs 8 through 15, etc.

See Also
  AerVirtGetBinaryOutput
28.10. AerVirtGetBinaryOutputDWordEx

AERERR_CODE AerVirtGetBinaryOutputDWordEx (HAERCTRL hAerCtrl, DWORD dwDWord, PDWORD pdwData);

Declare Function AerVirtGetBinaryOutputDWordEx Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal dwDWord As Long, ByRef pdwData As Long) As Long

Parameters

- **hAerCtrl**: Handle to axis processor card.
- **dwDWord**: Which binary output DWORD (32 bit quantity) to read.
- **pdwData**: Pointer to a DWORD to hold 32 bits of data.

This function reads 32 bits (1 DWORD) of data from the Virtual Output space. DWORD 0 would return the values of binary outputs 0 – 31. DWORD 1 would return the values of binary outputs 32 – 63, etc.
28.11. AerVirtGetBinaryOutputWordArray

AERERR_CODE AerVirtGetBinaryOutputWordArray (HAERCTRL hAerCtrl, DWORD dwStartWord, PWORD pwArray, DWORD dwNumWords);

Declare Function AerVirtGetBinaryOutputWordArray Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal dwStartWord As Long, ByRef pwArray As Integer, ByVal dwNumWords As Long) As Long

Parameters

- **hAerCtrl**: Handle to axis processor card.
- **dwStartWord**: Which binary output WORD (16 bit quantity) to start reading from.
- **pwArray**: An array of 16 bit data WORDS to hold the binary inputs.
- **dwNumWords**: The number of 16 bit data WORDS to read.

This function reads the specified number of binary output words from the virtual output space into the given array space.

WORD 0 would return the values of binary outputs 0 – 15.
WORD 1 would return the values of binary outputs 16 – 31, etc.

AERERR_CODE AerVirtGetBinaryOutputWordEx (HAERCTRL hAerCtrl, DWORD dwWord, PWORD pwData);

Declare Function AerVirtGetBinaryOutputWordEx Lib "AERSYS.DLL" (ByVal hAerCtrl As Long, ByVal dwWord As Long, ByRef pwData As Integer) As Long

Parameters

- **hAerCtrl**  Handle to axis processor card.
- **dwWord**  Which binary output WORD (16 bit quantity) to read.
- **pwData**  Pointer to a WORD to hold 16 bits of data.

This function reads 16-bits (1 WORD) of data from the virtual output space. The valid binary output words are 0 – 31 (where 0 returns the value of bits 0 – 15; 1 returns the value of bits 16 – 31; 2 returns the values of bits 32 – 47, etc.).
28.13. AerVirtGetRegisterInput

AERERR_CODE AerVirtGetRegisterInput (HAERCTRL hAerCtrl, DWORD dwNum, PWORD pwValue);

Declare Function AerVirtGetRegisterInput Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal dwNum As Long, ByRef pwValue As Integer) As Long

Parameters
- **hAerCtrl**  Handle to the axis processor card.
- **dwNum**  Register number to read.
- **pwValue**  Address of variable to receive value from the specified register.

This function will read a register from within the UNIDEX 600 Series controller’s virtual input register space. There are 128, 16 bit registers, numbered 0 through 127.

See Also
- *AerVirtGetRegisterInputEx*

AERERR_CODE AerVirtGetRegisterInputArray (HAERCTRL hAerCtrl, DWORD dwStartReg, PWORD pwArray, DWORD dwNumReg);

Declare Function AerVirtGetRegisterInputArray Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal dwStartReg As Long, ByRef pwArray As Integer, ByVal dwNumReg As Long) As Long

Parameters

- **hAerCtrl**: Handle to axis processor card.
- **dwStartReg**: Which register input WORD (16 bit quantity) to start reading from.
- **pwArray**: An array of WORDS to read the registers into.
- **dwNumReg**: The number of 16 bit register WORDS to read.

The *AerVirtGetRegisterInputArray* function reads the specified number of register input WORDS from the virtual register input space into the given array. There are 128 register inputs, numbered 0 – 127.
28.15. AerVirtGetRegisterInputEx

AERERR_CODE AerVirtGetRegisterInputEx (HAERCTRL hAerCtrl, DWORD dwReg, PWORD pwData);

Declare Function AerVirtGetRegisterInputEx Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal dwReg As Long, ByRef pwData As Integer) As Long

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hAerCtrl</td>
<td>Handle to the axis processor card.</td>
</tr>
<tr>
<td>dwReg</td>
<td>Register number to read.</td>
</tr>
<tr>
<td>pwData</td>
<td>Address of variable to receive value from the specified register.</td>
</tr>
</tbody>
</table>

This function will read a register from within the UNIDEX 600 Series controller's virtual input register space. There are 128, 16 bit registers, numbered 0 through 127.

See Also

AerVirtGetRegisterInput
28.16. AerVirtGetRegisterIO

AERERR_CODE AerVirtGetRegisterIO (HAERCTRL hAerCtrl, 
    PAERVIRT_REGISTER_DATA pInputs, 
    PAERVIRT_REGISTER_DATA pOutputs);

Parameters

- **hAerCtrl**: Handle to axis processor card.
- **pInputs**: Pointer to AERVIRT_REGISTER_DATA to hold the register input data (may be NULL).
- **pOutputs**: Pointer to AERVIRT_REGISTER_DATA to hold the register output data (may be NULL).

This function retrieves the current state of all the virtual register inputs and outputs on the U600 card. One of the parameters, *pInputs* and *pOutputs*, may be NULL if both types of register data is not required.
28.17. **AerVirtGetRegisterOutput**

AERERR_CODE AerVirtGetRegisterOutput (HAERCTRL hAerCtrl, DWORD dwNum, PWORD pwValue);

Declare Function AerVirtGetRegisterOutput Lib "AERSYS.DLL" (ByVal hAerCtrl As Long, ByVal dwNum As Long, ByRef pwValue As Integer) As Long

**Parameters**

- `hAerCtrl` Handle to the axis processor card.
- `dwNum` Register number to read.
- `pwValue` Address of variable to receive value from the specified register.

This function will read a register from within the UNIDEX 600 Series controllers virtual output register space. There are 128, 16 bit registers, numbered 0 through 127.

**See Also**

* AerVirtGetRegisterOutputEx
28.18. AerVirtGetRegisterOutputArray

AERERR_CODE AerVirtGetRegisterOutputArray (HAERCTRL hAerCtrl, DWORD dwStartReg, PWORD pwArray, DWORD dwNumReg);

Declare Function AerVirtGetRegisterOutputArray Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal dwStartReg As Long, ByRef pwArray As Integer, ByVal dwNumReg As Long) As Long

Parameters

- **hAerCtrl**: Handle to axis processor card.
- **dwStartReg**: Which register output WORD (16 bit quantity) to start reading from.
- **pwArray**: An array of WORDS to read the registers into.
- **dwNumReg**: The number of 16 bit register WORDS to read.

This function reads the specified number of register output WORDS from the virtual register output space into the given array. There are 128 register outputs numbered 0 – 127.
28.19. AerVirtGetRegisterOutputEx

AERERR_CODE AerVirtGetRegisterOutputEx (HAERCTRL hAerCtrl, DWORD dwReg, PWORD pwData);

Declare Function AerVirtGetRegisterOutputEx Lib "AERSYS.DLL" (ByVal hAerCtrl As Long, ByVal dwReg As Long, ByRef pwData As Integer) As Long

Parameters
- `hAerCtrl` Handle to the axis processor card.
- `dwReg` Register number to read.
- `pwData` Address of variable to receive value from the specified register.

This function will read a register from within the UNIDEX 600 Series controllers virtual output register space. There are 128, 16 bit registers, numbered 0 through 127.

See Also
- `AerVirtGetRegisterOutput`
28.20. AerVirtSetBinaryInput

AERERR_CODE AerVirtSetBinaryInput (HAERCTRL hAerCtrl, DWORD dwNum, BYTE bValue);

Declare Function AerVirtSetBinaryInput Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal dwNum As Long, ByVal bValue As Long) As Long

Parameters
- hAerCtrl: Handle to the axis processor card.
- dwNum: Binary input bit number to set.
- bValue: Value to write.

This function will write data to the UNIDEX 600 Series controller’s virtual binary input space. There are 512 binary inputs, numbered 0 through 511. Writing to any virtual inputs that are mapped to physical inputs will be over-written by the state of the physical input on the next update of the virtual I/O by the axis processor card.

See Also
- AerVirtSetBinaryInputByteEx
28.21. AerVirtSetBinaryInputByteEx

AERERR_CODE AerVirtSetBinaryInputByteEx (HAERCTRL hAerCtrl, DWORD dwByte, BYTE byData);

Declare Function AerVirtSetBinaryInputByteEx Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal dwByte As Long, ByVal byData As Byte) As Long

Parameters

  hAerCtrl       Handle to the axis processor card.
  dwByte        Binary output byte number to set.
  byData         Byte of data to write.

This function will write 8 bits (1 byte) of data to the UNIDEX 600 Series controller’s virtual binary input space. There are 512 binary inputs, numbered 0 through 511, contained in bytes 0 through 64. Byte 0 would return the values of binary outputs 0 through 7, byte 1 would return the values of binary outputs 8 through 15, etc. Writing to any virtual inputs that are mapped to physical inputs will be over-written by the state of the physical input on the next update of the virtual I/O by the axis processor card.

See Also

  AerVirtSetBinaryInput
28.22. **AerVirtSetBinaryInputDWordEx**

AERERR_CODE AerVirtSetBinaryInputDWordEx (HAERCTRL hAerCtrl, DWORD dwDWord, DWORD dwData);

 Declare Function AerVirtSetBinaryInputDWordEx Lib “AERSYS.DLL” (ByVal \
 hAerCtrl As Long, ByVal dwDWord As Long, ByVal dwData As Long) As Long

**Parameters**

  - *hAerCtrl* Handle to axis processor card.
  - *dwDWord* Which binary input DWORD (32 bit quantity) to write to.
  - *dwData* A 32 bit data WORD to write.

This function writes 32 bits (1 DWORD) of data to the virtual input space.

DWORD 0 would write to binary inputs 0 – 31.
DWORD 1 would write to binary inputs 32 – 63, etc.
28.23. **AerVirtSetBinaryInputWordArray**

AERERR_CODE AerVirtSetBinaryInputWordArray (HAERCTRL hAerCtrl, DWORD dwStartWord, PWORD pwArray, DWORD dwNumWords);

Declare Function AerVirtSetBinaryInputWordArray Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal dwStartWord As Long, ByRef pwArray As Integer, ByVal dwNumWords As Long) As Long

**Parameters**
- **hAerCtrl**: Handle to axis processor card.
- **dwStartWord**: Which binary input WORD (16 bit quantity) to start writing to.
- **pwArray**: An array of 16 bit data WORDS to write.
- **dwNumWords**: The number of 16 bit data WORDS to write.

This function writes the specified number of binary input WORDS to the virtual input space from the given array.

WORD 0 would write to binary inputs 0 – 15.
WORD 1 would write to binary inputs 16 – 31, etc.
28.24. AerVirtSetBinaryInputWordEx

AERERR_CODE AerVirtSetBinaryInputWordEx (HAERCTRL hAerCtrl, DWORD dwWord, WORD wData);

Declare Function AerVirtSetBinaryInputWordEx Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal dwWord As Long, ByVal wData As Integer) As Long

Parameters
  hAerCtrl       Handle to axis processor card.
  dwWord         Which binary input WORD (16 bit quantity) to write.
  wData          A 16 bit data WORD to write.

This function writes 16 bits (1 WORD) of data to the virtual input space.

WORD 0 would write to binary inputs 0 – 15.
WORD 1 would write to binary inputs 16 – 31, etc.
28.25. AerVirtSetBinaryIO

AERERR_CODE AerVirtSetBinaryIO (HAERCTRL hAerCtrl,
    PAERVIRT_BINARY_DATA pInputs,
    PAERVIRT_BINARY_DATA pOutputs);

Parameters

    hAerCtrl  Handle to axis processor card.
    pInputs   Pointer to AERVIRT_BINARY_DATA with the binary input data.
    pOutputs  Pointer to AERVIRT_BINARY_DATA with the binary output data.

This function sets the current state of all the virtual binary inputs and outputs on the U600 card. One of the parameters, *pInputs* and *pOutputs*, may be NULL if both types do not need to be written.
28.26. **AerVirtSetBinaryOutput**

AERERR_CODE AerVirtSetBinaryOutput (HAERCCTRL hAerCtrl, DWORD dwNum, BOOL bValue);

Declare Function AerVirtSetBinaryOutput Lib "AERSYS.DLL" (ByVal hAerCtrl As Long, ByVal dwNum As Long, ByVal bValue As Long) As Long

**Parameters**

- **hAerCtrl**
  - Handle to the axis processor card.
- **dwNum**
  - Binary output bit number to set.
- **bValue**
  - Value to write.

This function will write data to the UNIDEX 600 Series controller’s virtual binary output space. There are 512 binary outputs, numbered 0 through 511.

**See Also**

- AerVirtSetBinaryOutputByteEx

**Example**

Samples\Lib\VisualBasic\RunPgm.vbp
28.27. AerVirtSetBinaryOutputByteEx

AERERR_CODE AerVirtSetBinaryOutputByteEx (HAERCTRL hAerCtrl, DWORD dwByte, BYTE byData);

Declare Function AerVirtSetBinaryOutputByteEx Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal dwByte As Long, ByVal byData As Byte) As Long

Parameters
  hAerCtrl Handle to the axis processor card.
  dwByte Binary output byte number to set.
  byData Byte of data to write.

This function will write 8 bits (1 byte) of data to the UNIDEX 600 Series controllers virtual binary output space. There are 512 binary outputs, numbered 0 through 511, contained in bytes 0 through 64. Byte 0 would set the values of binary outputs 0 through 7, byte 1 would set the values of binary outputs 8 through 15, etc.

See Also
  AerVirtSetBinaryOutput
28.28. AerVirtSetBinaryOutputDWordEx

AERERR_CODE AerVirtSetBinaryOutputDWordEx (HAERCTRL hAerCtrl, DWORD dwDWord, DWORD dwData);

Declare Function AerVirtSetBinaryOutputDWordEx Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal dwDWord As Long, ByVal wData As Long) As Long

Parameters

- **hAerCtrl** Handle to axis processor card.
- **dwWord** Which binary output DWORD (32 bit quantity) to write to.
- **wData** A 32 bit data WORD to write.

This function writes 32 bits (1 DWORD) of data to the virtual output space.

DWORD 0 would write to binary inputs 0 – 31.
DWORD 1 would write to binary inputs 32 – 63, etc.
28.29. AerVirtSetBinaryOutputWordArray

AERERR_CODE AerVirtSetBinaryOutputWordArray (HAERCTRL hAerCtrl, DWORD dwStartWord, PWORD pwArray, DWORD dwNumWords);

Declare Function AerVirtSetBinaryOutputWordArray Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal dwStartWord As Long, ByRef pwArray As Integer, ByVal dwNumWords As Long) As Long

Parameters

- hAerCtrl: Handle to axis processor card.
- dwStartWord: Which binary output WORD (16 bit quantity) to start writing to.
- pwArray: An array of 16 bit data WORDS to write.
- dwNumWords: The number of 16 bit data WORDS to write.

This function writes the specified number of binary output WORDS to the virtual output space from the given array.

WORD 0 would write to binary inputs 0 – 15.
WORD 1 would write to binary inputs 16 – 31, etc.
28.30. AerVirtSetBinaryOutputWordEx

AERERR_CODE AerVirtSetBinaryOutputWordEx (HAERCTRL hAerCtrl, DWORD dwWord, WORD wData);

Declare Function AerVirtSetBinaryOutputWordEx Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal dwWord As Long, ByVal wData As Integer) As Long

Parameters

- hAerCtrl Handle to axis processor card.
- dwWord Which binary output WORD (16 bit quantity) to write.
- wData A 16 bit data WORD to write.

This function writes 16 bits (1 WORD) of data to the virtual output space.

WORD 0 would write to binary inputs 0 – 15.
WORD 1 would write to binary inputs 16 – 31, etc.
28.31. **AerVirtSetRegisterInput**

AERERR_CODE AerVirtSetRegisterInput (HAERCTRL haerCtrl, DWORD dwNum, WORD wValue);

Declare Function AerVirtSetRegisterInput Lib “AERSYS.DLL” (ByVal haerCtrl As Long, ByVal dwNum As Long, ByVal wValue As Integer) As Long

**Parameters**

- **haerCtrl**: Handle to the axis processor card.
- **dwReg**: Register number to write.
- **wValue**: Value to write.

This function will write a value to a register within the UNIDEX 600 Series controller’s virtual input register space. There are 128, 16 bit registers, numbered 0 through 127.

**See Also**

- AerVirtSetRegisterInputEx
28.32. AerVirtSetRegisterInputArray

AERERR_CODE AerVirtSetRegisterInputArray (HAERCTRL hAerCtrl, DWORD dwStartReg, PWORD pwArray, DWORD dwNumReg);

Declare Function AerVirtSetRegisterInputArray Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal dwStartReg As Long, ByRef pwArray As Integer, ByVal dwNumReg As Long) As Long

Parameters

- \( hAerCtrl \) Handle to axis processor card.
- \( dwStartReg \) Which register input WORD (16 bit quantity) to start writing to.
- \( pwArray \) An array of 16 bit register WORDS to write.
- \( dwNumReg \) The number of 16 bit register WORDS to write.

This function writes the specified number of register input WORDS to the virtual register input space from the given array space. There are 128 (16 bit registers) numbered 0 – 127.
28.33. AerVirtSetRegisterInputEx

AERERR_CODE AerVirtSetRegisterInputEx (HAERCTRL hAerCtrl, DWORD dwReg, WORD wData);

Declare Function AerVirtSetRegisterInputEx Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal dwReg As Long, ByVal pwData As Integer) As Long

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hAerCtrl</td>
<td>Handle to the axis processor card.</td>
</tr>
<tr>
<td>dwReg</td>
<td>Register number to write.</td>
</tr>
<tr>
<td>wData</td>
<td>Value to write.</td>
</tr>
</tbody>
</table>

This function will write a value to a register within the UNIDEX 600 Series controller’s virtual input register space. There are 128, 16 bit registers, numbered 0 through 127.

See Also

AerVirtSetRegisterInput
28.34. AerVirtSetRegisterIO

AERERR_CODE AerVirtSetRegisterIO (HAERCTRL hAerCtrl, 
PAERVIRT_REGISTER_DATA pInputs, 
PAERVIRT_REGISTER_DATA pOutputs);

Parameters

- **hAerCtrl**
  Handle to axis processor card.

- **pInputs**
  Pointer to AERVIRT_REGISTER_DATA with the register input data (may be NULL).

- **pOutputs**
  Pointer to AERVIRT_REGISTER_DATA with the register output data (may be NULL).

This function sets the current state of all the virtual register inputs and outputs on the U600 card. One of the parameters, *pInputs* and *pOutputs*, may be NULL if both types of data are not required.
28.35. AerVirtSetRegisterOutput

AERERR_CODE AerVirtSetRegisterOutput (HAERCTRL hAerCtrl, DWORD dwNum, WORD wValue);

Declare Function AerVirtSetRegisterOutput Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal dwNum As Long, ByVal wValue As Integer) As Long

Parameters

- **hAerCtrl**  
  Handle to the axis processor card.

- **dwReg**  
  Register number to write to.

- **wData**  
  Data to write to the specified output register.

This function will write a value to an output register within the UNIDEX 600 Series controllers virtual output register space. There are 128, 16 bit registers, numbered 0 through 127.

See Also

- AerVirtSetRegisterOutputEx
28.36. AerVirtSetRegisterOutputArray

AERERR_CODE AerVirtSetRegisterOutputArray (HAERCTRL hAerCtrl, DWORD dwStartReg, PWORD pwArray, DWORD dwNumReg);

Declare Function AerVirtSetRegisterOutputArray Lib "AERSYS.DLL" (ByVal hAerCtrl As Long, ByVal dwStartReg As Long, ByRef pwArray As Integer, ByVal dwNumReg As Long) As Long

Parameters
  hAerCtrl Handle to axis processor card.
  dwStartReg Which register output WORD (16 bit quantity) to start writing to.
  pwArray An array of 16 bit WORDS to write.
  dwNumReg The number of 16 bit register WORDS to write.

This function writes the specified number of values to the virtual register output space from the given array.
**28.37. AerVirtSetRegisterOutputEx**

AERERR_CODE AerVirtSetRegisterOutputEx (HAERCTRL hAerCtrl, DWORD dwReg, WORD wData);

Declare Function AerVirtSetRegisterOutputEx Lib “AERSYS.DLL” (ByVal hAerCtrl As Long, ByVal dwReg As Long, ByVal wData As Integer) As Long

**Parameters**
- **hAerCtrl** Handle to the axis processor card.
- **dwReg** Register number to write to.
- **wData** Data to write to the specified output register.

This function will write a value to an output register within the UNIDEX 600 Series controllers virtual output register space. There are 128, 16 bit registers, numbered 0 through 127.

**See Also**
- `AerVirtSetRegisterOutput`
APPENDIX A: CONSTANTS FOR C LANGUAGE AND LABVIEW USERS

A.1. Description

The following section provides a quick reference of all the constants used within the library’s (AerSys.DLL, AerErr.DLL, and AerCmplr.DLL).
A.2. AER_BP

These constants are used for defining breakpoints, on, off and toggling the break point.

**Constants**

- AER_BP_TOGGLE  (DWORD) -1
- AER_BP_OFF     0
- AER_BP_ON      1
A.3. AER_CARD

These constants are used to register a card with the system and to establish communications with the card during device initialization. The Aerotech device driver can communicate with up to four cards simultaneously.

**Constants**

- AER_CARD_NONE: -1
- AER_CARD_DEFAULT: 0
- AER_CARD_1: 1
- AER_CARD_2: 2
- AER_CARD_3: 3
- AER_CARD_4: 4

**Used by**

- AerRegxxxx
- AerSysOpen
### A.4. AER_CPUSTAT

These indicate the state of the controller.

**Constants**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AER_CPUSTAT_BOOT_EXECUTE</td>
<td>1</td>
<td>The CPU is booting itself up</td>
</tr>
<tr>
<td>AER_CPUSTAT_IMG_EXECUTE</td>
<td>2</td>
<td>The CPU is currently running an image file</td>
</tr>
<tr>
<td>AER_CPUSTAT_FATAL</td>
<td>3</td>
<td>The CPU is in an erroneous or unknown state</td>
</tr>
</tbody>
</table>

**Used by**

*AerVerGetCpuStatus*
A.5. AER_DRV600_DEFAULT

These constants are used for defining the default values of the device driver to communicate with the controller. They are AT Window, 0xdc00 0000, I/O base address 0x220, IRQ 5 and DRAM Size Code, 2 Megabytes.

**Constants**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AER_DRV600_DEFAULT_ATWIN</td>
<td>0xdc00</td>
<td>AT window</td>
</tr>
<tr>
<td>AER_DRV600_DEFAULT_IO</td>
<td>0x220</td>
<td>I/O address</td>
</tr>
<tr>
<td>AER_DRV600_DEFAULT_IRQ</td>
<td>5</td>
<td>IRQ 5</td>
</tr>
<tr>
<td>AER_DRV600_DEFAULT_DSC</td>
<td>2</td>
<td>2 megabytes</td>
</tr>
</tbody>
</table>
A.6. **AER_EVENT**

These constants are used to determine the state of the active tasks on the Axis Processor card.

**Constants**

- **AER_EVENT_TASK_FAULT** 0x00008002 // A task related fault has occurred (i.e. bad CNC line seen)
- **AER_EVENT_AXIS_FAULT** 0x00008003 // An axis related fault has occurred (i.e. position error)
- **AER_EVENT_TASK_CALLBACK** 0x00008006 // Axis processor requests a 'task operation'
- **AER_EVENT_VIRTIO_UPDATE** 0x00008005 // The controller wants to change the value of an external output.
- **AER_EVENT_SERIAL** 0x00008004 // Serial port on Axis processor input seen
- **AER_EVENT_AXIS_FAIL** // Interrupt seen, but no data found.
- **AER_EVENT_IRQ2_TIMER** // Secondary loop interrupt (U631 only)

**Used by**

_AerEventCreateEvent_
A.7. AER_FBTYPE_XXXX

An axis feedback type definition.

Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AER_FBTYPE_NULL</td>
<td>0</td>
<td>no feedback</td>
</tr>
<tr>
<td>AER_FBTYPE_RESOLVER</td>
<td>1</td>
<td>resolver or inductosyn</td>
</tr>
<tr>
<td>AER_FBTYPE_ENCODER</td>
<td>2</td>
<td>encoder feedback</td>
</tr>
<tr>
<td>AER_FBTYPE_ENCODER_HALL</td>
<td>3</td>
<td>brushless motor with encoder/hall</td>
</tr>
<tr>
<td>AER_FBTYPE_HPVM</td>
<td>4</td>
<td>HP VME laser feedback</td>
</tr>
<tr>
<td>AER_FBTYPE_NEW_EXT_ENCODER</td>
<td>5</td>
<td>spare</td>
</tr>
<tr>
<td>AER_FBTYPE_RESOLVER_HALL</td>
<td>6</td>
<td>resolver with hall</td>
</tr>
<tr>
<td>AER_FBTYPE_STEPPER</td>
<td>7</td>
<td>open loop stepper motor</td>
</tr>
<tr>
<td>AER_FBTYPE_HALL_POLE</td>
<td>10</td>
<td>Encoder Hall (pole pairs)</td>
</tr>
</tbody>
</table>

Used by

AerConfigXXX
A.8.  AER_IOTYPE_XXXX

An axis IO type definition.

Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AER_IOTYPE_NULL</td>
<td>0</td>
<td>no IO type</td>
</tr>
<tr>
<td>AER_IOTYPE_D2A</td>
<td>1</td>
<td>IO type is D2A</td>
</tr>
</tbody>
</table>

Used by

AerConfigXXXX
A.9. AER_PROBESTATUS_XXXX

These constants are used to determine the status of the probe.

**Constants**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AER_PROBESTATUS_ARMED</td>
<td>0x01</td>
<td>Probe is armed</td>
</tr>
<tr>
<td>AER_PROBESTATUS_VALIDPOS</td>
<td>0x02</td>
<td>Position is valid</td>
</tr>
<tr>
<td>AER_PROBESTATUS_INPUT</td>
<td>0x04</td>
<td>Has been configured</td>
</tr>
<tr>
<td>AER_PROBESTATUS_HIGHSPEED</td>
<td>0x08</td>
<td>Using High speed position</td>
</tr>
<tr>
<td></td>
<td></td>
<td>// Using High speed position latch</td>
</tr>
</tbody>
</table>

**Used By**

_Probe Functions_
A.10. AER_MFBTYPE_XXXX

An axis master feedback type definition.

**Constants**

<table>
<thead>
<tr>
<th>Definition</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AER_MFBTYPE_NULL</td>
<td>0</td>
<td>No master feedback</td>
</tr>
<tr>
<td>AER_MFBTYPE_RESOLVER</td>
<td>1</td>
<td>Resolver master feedback</td>
</tr>
<tr>
<td>AER_MFBTYPE_ENCODER</td>
<td>2</td>
<td>Encoder master feedback</td>
</tr>
<tr>
<td>AER_MFBTYPE_VIRTUAL</td>
<td>3</td>
<td>Virtual master feedback</td>
</tr>
<tr>
<td>AER_MFBTYPE_AUXENCODER</td>
<td>4</td>
<td>Auxiliary encoder feedback</td>
</tr>
</tbody>
</table>

**Used by**

AerConfigXXXX
A.11. **AER_SPARETYPE1_XXXX**

A spare axis feedback type definition.

**Constants**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AER_SPARETYPE1_NULL</td>
<td>0</td>
<td>Spare NULL feedback</td>
</tr>
<tr>
<td>AER_SPARETYPE1_ENCODER</td>
<td>3</td>
<td>Spare encoder feedback</td>
</tr>
<tr>
<td>AER_SPARETYPE1_ENCODER_SLAVE</td>
<td>4</td>
<td>Spare encoder slave</td>
</tr>
<tr>
<td>AER_SPARETYPE1_RESOLVER</td>
<td>5</td>
<td>Spare resolver feedback</td>
</tr>
</tbody>
</table>

**Used by**

*AerConfigXXXX*
A.12. AER_SPARETYPE2_XXXX

A spare feedback type definition, defined as 0.

Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AER_SPARETYPE2_NULL</td>
<td>0</td>
</tr>
<tr>
<td>AER_SPARETYPE2_ANALOG</td>
<td>1</td>
</tr>
</tbody>
</table>

Used by

AerConfigXXX
A.13. AER_UNIDEX

These constants are used to specify to the device driver, the type of Aerotech controller hardware that it will be communicating with. They are used when registering the system in the Win32 registry.

**Constants**

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AER_UNIDEX_NONE</td>
<td>-1</td>
<td>Reserved for Aerotech use</td>
</tr>
<tr>
<td>AER_UNIDEX_DEFAULT</td>
<td>0</td>
<td>Default machine</td>
</tr>
<tr>
<td>AER_UNIDEX_600PC</td>
<td>600</td>
<td>U600 (AT bus)</td>
</tr>
</tbody>
</table>

**Used by**

- AerRegxxxx
- AerSysOpen
A.14. AERCAM

These constants are used by the various camming functions.

Constants

<table>
<thead>
<tr>
<th>Constant Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AERCAM_POINT_LINEAR</td>
<td>0</td>
<td>Linear interpolation</td>
</tr>
<tr>
<td>AERCAM_POINT_CUBIC</td>
<td>1</td>
<td>Cubic interpolation</td>
</tr>
</tbody>
</table>

Used by

- AerCamTableSetPoint
- AerCamTableGetPoint

Constants

<table>
<thead>
<tr>
<th>Constant Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AERCAM_MODE_OFF</td>
<td>0</td>
<td>Turn off synchronization</td>
</tr>
<tr>
<td>AERCAM_MODE_RELATIVE</td>
<td>1</td>
<td>Relative camtable mode</td>
</tr>
<tr>
<td>AERCAM_MODE_ABSOLUTE</td>
<td>2</td>
<td>Absolute camtable mode</td>
</tr>
<tr>
<td>AERCAM_MODE_VELOCITY</td>
<td>3</td>
<td>Velocity camtable mode</td>
</tr>
</tbody>
</table>

Used by

- AerCamTableGetStatus
- AerCamTableGetMode

Constants

<table>
<thead>
<tr>
<th>Constant Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AERCAM_STAT_ALLOCATED</td>
<td>1</td>
<td>Cam table allocated</td>
</tr>
<tr>
<td>AERCAM_STAT_COEFFS_DONE</td>
<td>2</td>
<td>Coefficients have been calculated</td>
</tr>
<tr>
<td>AERCAM_STAT_COEFFS_BUSY</td>
<td>4</td>
<td>Coefficients being calculated</td>
</tr>
</tbody>
</table>

Used by

- AerCamTableGetStatus
A.15. AERCMPLR

These constants control the CNC compilation, combine the desired flags together passing as the `dwMode` parameter to `AerCompilerCompileLine` or `AerCompilerCompileFile`. Some combinations may not be legal (in these cases an error will be returned) or some options may be ignored (if `AERCMPL_PREPROC_ONLY` is set, `AERCMPLR_NO_USE_OBJ` is ignored).

**Constants**

- `AERCMPLR_DEFAULT` 0x0 // Uses object files, only
  - recompile when necessary,
  - don’t make listing
- `AERCMPLR_PREPROC_ONLY` 0x1 // Only run preprocessor (use
  - with MAKE_LISTING)
- `AERCMPLR_NO_READ_OBJS` 0x2 // Always recompile it, don’t
  - write obj.
- `AERCMPLR_NO_USE_SRC` 0x4 // Read object (no source code
  - available)
- `AERCMPLR_MAKE_LISTING` 0x8 // Puts preprocessor output in file
- `AERCMPLR_SRC_WITH_OBJ` 0x10
- `AERCMPLR_WRITE_OBJ` 0x20

**Used by**

- `AerCompilerCompileLine`
- `AerCompilerCompileFile`
A.16. AERERR_TYPE

These constants specify the type of warning message (Message Only, Warning Message, Error Message or No Message).

Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AERERR_TYPE_MSG</td>
<td>0</td>
</tr>
<tr>
<td>AERERR_TYPE_WARN</td>
<td>1</td>
</tr>
<tr>
<td>AERERR_TYPE_ERROR</td>
<td>2</td>
</tr>
<tr>
<td>AERERR_TYPE_NONE</td>
<td>3</td>
</tr>
</tbody>
</table>
A.17. AERREGID

Constants

// these values are used to get values (AerRegGet/SetFilename())

AERREGID_AxisCfgFile  0  // <iniDir>\AxisCfg.ini
AERREGID_AxisWizFile  1  // <iniDir>\AxisCfg.wiz
AERREGID_AxisParmFile 2  // <iniDir>\AxisParm.ini
AERREGID_MachParmFile 3  // <iniDir>\MachParm.ini
AERREGID_TaskParmFile 4  // <iniDir>\TaskParm.ini
AERREGID_GlobParmFile 5  // <iniDir>\GlobParm.ini
AERREGID_ParmDefFile  6  // <iniDir>\ParmDef.ini
AERREGID_InstallDir   7  // c:\u600(created by setup)
AERREGID_ProgramDir   8  // <installDir>\Programs
AERREGID_InitDir      9  // <installDir>\Ini
AERREGID_BinDir       10 // <installDir>\Bin
AERREGID_ImageFile    11 // <binDir>\PC960.img
AERREGID_BootImageFile 12 // <binDir>\PC960BT.img
AERREGID_SymbolicName 13 // U600/<binDir>\U600.vxd
AERREGID_Cal2DFile    14 // 2D Calibration File
AERREGID_AutomationFile 15 // U600auto.ini (Program
                            // Automation File)
AERREGID_ToolFile     16 // tool.ini (ToolTable file)
AERREGID_VirtIOFile   17 // VirtIO file
AERREGID_U600IniFile  18 // U600mmi.ini (U600 init File)
AERREGID_U600PosFile  19 // U600pos.ss (U600 positions
                            // display file)
AERREGID_U600File     20 // U600.ini (U600 filenames)
AERREGID_Max          21

Used By

AerRegGetFilename()
AerRegSetFilename()
A.18. AERTOOL

Constants

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AERTOOL_F_ENGLISH</td>
<td>0x00000001</td>
<td>units are english</td>
</tr>
<tr>
<td>AERTOOL_F_INUSE</td>
<td>0x00000002</td>
<td>tool can be used</td>
</tr>
<tr>
<td>AERTOOL_F_FORCE_UNITS</td>
<td>0x00000004</td>
<td>when tool is activated, it must be in proper units</td>
</tr>
<tr>
<td>AERTOOL_F_USER_DIAMETER</td>
<td>0x00080000</td>
<td>used by Interface/Display only. Values (Radius/Wear) are diameter</td>
</tr>
<tr>
<td>AERTOOL_F_VALID</td>
<td>0x80000000</td>
<td>internal - the tool has been loaded</td>
</tr>
</tbody>
</table>
A.19. ALT_STAT

To determine the other (alternate) motion status words of an axis.

Constants

// Alternate Status word returned by
ALT_STAT_LAST_MVE 0x00000001 // mask for final move into
// home
ALT_STAT_ALT_HME 0x00000002 // mask for alternate home
ALT_STAT_PWR_HME 0x00000004 // mask for mask for home start
// in limit
ALT_STAT_FND_HME 0x00000008 // mask for home limit detected
ALT_STAT_TRQ_SET 0x00000010 // mask for valid torque
// parameters
ALT_STAT_TRQ_EN 0x00000020 // mask for torque enable
ALT_STAT_HALL_COMM 0x00000040 // mask for hall effect
// commutation
ALT_STAT_HALL_EDGE 0x00000080 // mask for first edge received
ALT_STAT_DUAL_LOOP 0x00000100 // mask for dual loop feedback
ALT_STAT_VME_JOG 0x00000200 // mask for vme jog mode
ALT_STAT_LST_DIR 0x00000400 // mask for ccw motion in hall
// commutation
ALT_STAT_JOG_DIR 0x00000800 // mask for positive jog
// direction
ALT_STAT_EXT_RES 0x00001000 // mask for external resolver
// feedback
ALT_STAT_REV_ERROR_ON 0x00002000 // reversal error enabled
ALT_STAT_LAST_REV_POS 0x00004000 // last reversal in pos dir
ALT_STAT_LAST_REV_NEG 0x00008000 // last reversal in neg dir
ALT_STAT_SAFE_ENABLE 0x00010000 // safe zone enabled
ALT_STAT_SAFE_OUTSIDE 0x00020000 // safe zone is outside region
ALT_STAT_SIMULATION 0x00040000 // feedrate override
ALT_STAT_HOME_VELOCITY 0x00080000 // home off the velocity
// transducer
ALT_STAT_ACCEL_RATE 0x00100000 // accel rate
ALT_STAT_DECEL_RATE 0x00200000 // decel rate
ALT_STAT_PHASE_MODE 0x00400000 // mask for phase advance
ALT_STAT_HOME_COMPLETE 0x00800000 //
ALT_STAT_LIMIT_OVERRIDE 0x01000000 // always check soft limits if
// reset
ALT_STAT_SLAVE_ENCODER 0x02000000 // slave encoder
ALT_STAT_QEC_CAM 0x04000000 // cam table queued
ALT_STAT_HOME_NOLIM 0x08000000 // home without home limit
ALT_STAT_SPEED_OUTPUT 0x10000000 //
ALT_STAT_STEP 0x20000000 // mask for stepper motor
ALT_STAT_NORMAL_OUT 0x40000000 // set for negative current
// output
ALT_STAT_MFB_ENCODER 0x80000000 // set for encoder for master
// feedback

Used by

Axis parameter ALT_STATUS
A.20. ANALOGINDEX

Specifies an analog channel.

Constants

ANALOGINDEX_xxx is an analog index in the range of 1-16, having a value of 0 to 15.

Used by

ANALOGINDEX variable
A.21. AXISINDEX

Physical axis index numbers.

Constants
   AXISINDEX_n where n is the axis number in the range of 1-16, having a value of 0 to 15.

Used by
   AerParmAxisGetValue, ...
A.22. AXISMASK

These constants specify multiple axes. They are added (summed) together to produce one parameter specifying multiple axes within one parameter.

Constants

<table>
<thead>
<tr>
<th>AXISMASK_n</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AXISMASK_1</td>
<td>0x00000001L</td>
</tr>
<tr>
<td>AXISMASK_2</td>
<td>0x00000002L</td>
</tr>
<tr>
<td>AXISMASK_3</td>
<td>0x00000004L</td>
</tr>
<tr>
<td>AXISMASK_4</td>
<td>0x00000008L</td>
</tr>
<tr>
<td>AXISMASK_5</td>
<td>0x00000010L</td>
</tr>
<tr>
<td>AXISMASK_6</td>
<td>0x00000020L</td>
</tr>
<tr>
<td>AXISMASK_7</td>
<td>0x00000040L</td>
</tr>
<tr>
<td>AXISMASK_8</td>
<td>0x00000080L</td>
</tr>
<tr>
<td>AXISMASK_9</td>
<td>0x00000100L</td>
</tr>
<tr>
<td>AXISMASK_10</td>
<td>0x00000200L</td>
</tr>
<tr>
<td>AXISMASK_11</td>
<td>0x00000400L</td>
</tr>
<tr>
<td>AXISMASK_12</td>
<td>0x00000800L</td>
</tr>
<tr>
<td>AXISMASK_13</td>
<td>0x00001000L</td>
</tr>
<tr>
<td>AXISMASK_14</td>
<td>0x00002000L</td>
</tr>
<tr>
<td>AXISMASK_15</td>
<td>0x00004000L</td>
</tr>
<tr>
<td>AXISMASK_16</td>
<td>0x00008000L</td>
</tr>
</tbody>
</table>
### A.23. AXISPARM

Axis Parameter numbers

<table>
<thead>
<tr>
<th>Constants</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AXISPARM_STATUS</td>
<td>1</td>
</tr>
<tr>
<td>AXISPARM_POS</td>
<td>2</td>
</tr>
<tr>
<td>AXISPARM_ECHO</td>
<td>3</td>
</tr>
<tr>
<td>AXISPARM_CLOCK</td>
<td>4</td>
</tr>
<tr>
<td>AXISPARM_RESOLVER</td>
<td>5</td>
</tr>
<tr>
<td>AXISPARM_RAWPOS</td>
<td>6</td>
</tr>
<tr>
<td>AXISPARM_POSCMD</td>
<td>7</td>
</tr>
<tr>
<td>AXISPARM_POSERR</td>
<td>8</td>
</tr>
<tr>
<td>AXISPARM_POSTOGO</td>
<td>9</td>
</tr>
<tr>
<td>AXISPARM_POSTARGET</td>
<td>10</td>
</tr>
<tr>
<td>AXISPARM_IVEL</td>
<td>11</td>
</tr>
<tr>
<td>AXISPARM_AVGVEL</td>
<td>12</td>
</tr>
<tr>
<td>AXISPARM_AVGVELTIME</td>
<td>13</td>
</tr>
<tr>
<td>AXISPARM_RESERVED2</td>
<td>15</td>
</tr>
<tr>
<td>AXISPARM Ki</td>
<td>16</td>
</tr>
<tr>
<td>AXISPARM_KP</td>
<td>17</td>
</tr>
<tr>
<td>AXISPARM_PGAIN</td>
<td>18</td>
</tr>
<tr>
<td>AXISPARM_VFF</td>
<td>19</td>
</tr>
<tr>
<td>AXISPARM_DRIVE</td>
<td>20</td>
</tr>
<tr>
<td>AXISPARM_AUX</td>
<td>21</td>
</tr>
<tr>
<td>AXISPARM_RESERVED3</td>
<td>22</td>
</tr>
<tr>
<td>AXISPARM_RESERVED4</td>
<td>23</td>
</tr>
<tr>
<td>AXISPARM_KFACTOR</td>
<td>24</td>
</tr>
<tr>
<td>AXISPARM_RESERVED5</td>
<td>25</td>
</tr>
<tr>
<td>AXISPARM_KD</td>
<td>26</td>
</tr>
<tr>
<td>AXISPARM_RESERVED6</td>
<td>27</td>
</tr>
<tr>
<td>AXISPARM_RESERVED38</td>
<td>28</td>
</tr>
<tr>
<td>AXISPARM_REVERSALMODE</td>
<td>29</td>
</tr>
<tr>
<td>AXISPARM_REVERSALVALUE</td>
<td>30</td>
</tr>
<tr>
<td>AXISPARM_IAVG</td>
<td>31</td>
</tr>
<tr>
<td>AXISPARM_IMAX</td>
<td>32</td>
</tr>
<tr>
<td>AXISPARM_IAVGLIMIT</td>
<td>33</td>
</tr>
<tr>
<td>AXISPARM_IAVGTIME</td>
<td>34</td>
</tr>
<tr>
<td>AXISPARM_ICMD</td>
<td>35</td>
</tr>
<tr>
<td>AXISPARM_PSEERRLIMIT</td>
<td>36</td>
</tr>
<tr>
<td>AXISPARM_INPOSLIMIT</td>
<td>37</td>
</tr>
<tr>
<td>AXISPARM_CWEOT</td>
<td>38</td>
</tr>
<tr>
<td>AXISPARM_CCWEOT</td>
<td>39</td>
</tr>
<tr>
<td>AXISPARM_RESERVED36</td>
<td>40</td>
</tr>
<tr>
<td>AXISPARM_RESERVED37</td>
<td>41</td>
</tr>
<tr>
<td>AXISPARM_VELCMDTRAP</td>
<td>42</td>
</tr>
<tr>
<td>AXISPARM_VELOCITY</td>
<td>43</td>
</tr>
<tr>
<td>AXISPARM_FBWINDOW</td>
<td>44</td>
</tr>
<tr>
<td>AXISPARM_SAFEZONECW</td>
<td>45</td>
</tr>
<tr>
<td>AXISPARM_SAFEZONECCW</td>
<td>46</td>
</tr>
<tr>
<td>AXISPARM_SAFEZONEMODE</td>
<td>47</td>
</tr>
<tr>
<td>AXISPARM_SIMULATION</td>
<td>48</td>
</tr>
<tr>
<td>AXISPARM_ACCEL</td>
<td>49</td>
</tr>
<tr>
<td>AXISPARM_ACCELMODE</td>
<td>50</td>
</tr>
<tr>
<td>AXISPARM_DECEL</td>
<td>51</td>
</tr>
<tr>
<td>AXISPARM_DECELMODE</td>
<td>52</td>
</tr>
<tr>
<td>AXISPARM_FEEDRATE</td>
<td>53</td>
</tr>
<tr>
<td>AXISPARM_ACCELERATE</td>
<td>54</td>
</tr>
<tr>
<td>AXISPARM_DECELERATE</td>
<td>55</td>
</tr>
<tr>
<td>AXISPARM_RESERVED8</td>
<td>56</td>
</tr>
<tr>
<td>AXISPARM_HOMESWITCHPOS</td>
<td>57</td>
</tr>
<tr>
<td>AXISPARM_HOMESWITCHTOL</td>
<td>58</td>
</tr>
<tr>
<td>AXISPARM_SYSTEMCLOCK</td>
<td>59</td>
</tr>
</tbody>
</table>
AXISPARM_RESERVED9 59
AXISPARM_RESERVED10 60
AXISPARM_RESERVED11 61
AXISPARM_RESERVED12 62
AXISPARM_RESERVED13 63
AXISPARM_FAULT 64
AXISPARM_FAULTMASK 65
AXISPARM_DISABLEMASK 66
AXISPARM_INTMASK 67
AXISPARM_AUXMASK 68
AXISPARM_HALT MASK 69
AXISPARM_IOLEVEL 70
AXISPARM_AUXOFFSET 71
AXISPARM_ABORTMASK 72
AXISPARM_RESERVED14 73
AXISPARM_RESERVED15 74
AXISPARM_RESERVED16 75
AXISPARM_RESERVED17 76
AXISPARM_RESERVED18 77
AXISPARM_CAMPOSITION 78
AXISPARM_CAMPoint 79
AXISPARM_MASTERPOS 80
AXISPARM_MASTERRES 81
AXISPARM_MASTERLEN 82
AXISPARM_RESERVED37 83
AXISPARM_CAMOFFSET 84
AXISPARM_SYNCSPEED 85
AXISPARM_RESERVED19 86 // PROFILETIME
AXISPARM_PROFQDEPTH 87
AXISPARM_PROFQSIZE 88
AXISPARM_MOVEQDEPTH 89
AXISPARM_MOVEQSIZE 90
AXISPARM_RESERVED20 91
AXISPARM_RESERVED21 92
AXISPARM_RESERVED22 93
AXISPARM_RESERVED23 94
AXISPARM_OFFSET 95
AXISPARM_RESERVED24 96 // ALIGN
AXISPARM_MOTIONSTATUS 97
AXISPARM_SERVOSTATUS 98
AXISPARM_RESERVED25 99
AXISPARM_RESERVED26 100
AXISPARM_RESERVED27 101
AXISPARM_RESERVED28 102
AXISPARM_RESERVED29 103
AXISPARM_RESERVED30 104
AXISPARM_RESERVED31 105
AXISPARM_RESERVED32 106
AXISPARM_CSUM 107
AXISPARM_RESERVED34 108 // CSUM_WRITE
AXISPARM_EXTR2DSCL 109
AXISPARM_BASE_SPEED 110
AXISPARM_MAX_PHASE 111
AXISPARM_PHASE_SPEED 112
AXISPARM_ALT_STATUS 113
AXISPARM_RESERVED35 114 // FORCE_LINES
AXISPARM_SOFTLIMITMODE 115
AXISPARM_HOMEVELMULT 116
AXISPARM_CAMADVANCE 117
AXISPARM_VGAIN 118
AXISPARM_ALPHA 119
AXISPARM_VELTIMECONST 120
AXISPARM_MAXCAMACCEL 121
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>AXISPARM_HOMEOFFSET</td>
<td>122</td>
</tr>
<tr>
<td>AXISPARM_ICMDPOLARITY</td>
<td>123</td>
</tr>
<tr>
<td>AXISPARM_BRAKEMASK</td>
<td>124</td>
</tr>
<tr>
<td>AXISPARM_DACOFFSET</td>
<td>125</td>
</tr>
<tr>
<td>AXISPARM_B0</td>
<td>126</td>
</tr>
<tr>
<td>AXISPARM_B1</td>
<td>127</td>
</tr>
<tr>
<td>AXISPARM_B2</td>
<td>128</td>
</tr>
<tr>
<td>AXISPARM_A1</td>
<td>129</td>
</tr>
<tr>
<td>AXISPARM_A2</td>
<td>130</td>
</tr>
<tr>
<td>AXISPARM_POSTOGOIRQ</td>
<td>131</td>
</tr>
<tr>
<td>AXISPARM_AUXDELAY</td>
<td>132</td>
</tr>
<tr>
<td>AXISPARM_SCALEPGAIN</td>
<td>133</td>
</tr>
<tr>
<td>AXISPARM_POSMODULO</td>
<td>134</td>
</tr>
<tr>
<td>AXISPARM_NEGMODULO</td>
<td>135</td>
</tr>
<tr>
<td>AXISPARM_MODULOMODE</td>
<td>136</td>
</tr>
<tr>
<td>AXISPARM_IMAXPLUS</td>
<td>137</td>
</tr>
<tr>
<td>AXISPARM_IMAXMINUS</td>
<td>138</td>
</tr>
<tr>
<td>AXISPARM_AUXLEVEL</td>
<td>139</td>
</tr>
<tr>
<td>AXISPARM_GEARSLAVE</td>
<td>140</td>
</tr>
<tr>
<td>AXISPARM_GEARMASTER</td>
<td>141</td>
</tr>
<tr>
<td>AXISPARM_GEARMODE</td>
<td>142</td>
</tr>
<tr>
<td>AXISPARM_AUXVELCMD</td>
<td>143</td>
</tr>
<tr>
<td>AXISPARM_POSITION_TOL</td>
<td>144</td>
</tr>
<tr>
<td>AXISPARM_POSITIONTOL_TIME</td>
<td>145</td>
</tr>
<tr>
<td>AXISPARM_GANTRYMODE</td>
<td>146</td>
</tr>
<tr>
<td>AXISPARM_GANTRYOFFSET</td>
<td>147</td>
</tr>
<tr>
<td>AXISPARM_GANTRYMASTER</td>
<td>148</td>
</tr>
<tr>
<td>AXISPARM_PHASEAOFFSET</td>
<td>149</td>
</tr>
<tr>
<td>AXISPARM_PHASEBOFFSET</td>
<td>150</td>
</tr>
<tr>
<td>AXISPARM_DATAPLOT_MODE</td>
<td>151</td>
</tr>
<tr>
<td>AXISPARM_BB0</td>
<td>152</td>
</tr>
<tr>
<td>AXISPARM_BB1</td>
<td>153</td>
</tr>
<tr>
<td>AXISPARM_BB2</td>
<td>154</td>
</tr>
<tr>
<td>AXISPARM_AA1</td>
<td>155</td>
</tr>
<tr>
<td>AXISPARM_AA2</td>
<td>156</td>
</tr>
<tr>
<td>MAX_AXISPARMS</td>
<td>157</td>
</tr>
</tbody>
</table>

**Used by**

_AerParmAxisxxxx_
A.24. AXISTYPE

These constants specify the type of axis.

**Constants**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AXISTYPE_LINEAR</td>
<td>0</td>
<td>Axis is linear</td>
</tr>
<tr>
<td>AXISTYPE_ROTARY_MODULO</td>
<td>1</td>
<td>Axis is rotary with rollover</td>
</tr>
<tr>
<td>AXISTYPE_ROTARY_NONMODULO</td>
<td>2</td>
<td>Axis is rotary</td>
</tr>
</tbody>
</table>

**Used by**

*Machine parameter – Type*
A.25. CSPARMINDEX

Call stack parameter index numbers.

Constants

CSPARMINDEX_n where n is the call stack parameter number in the range of 1-26 having a value of 0 to 25. These numbers correspond to the parameters passed by calls in the CNC program.

Used by

CSPARMINDEX variables
Compiler
A.26. FLT

To determine which fault is active.

Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLT_POSITION_ERROR</td>
<td>0x00000001</td>
<td>excess position error</td>
</tr>
<tr>
<td>FLT_I_AVERAGE</td>
<td>0x00000002</td>
<td>excess RMS current error</td>
</tr>
<tr>
<td>FLT_CW_HARD_LIMIT</td>
<td>0x00000004</td>
<td>CW hardware limit error</td>
</tr>
<tr>
<td>FLT_CCW_HARD_LIMIT</td>
<td>0x00000008</td>
<td>CCW hardware limit error</td>
</tr>
<tr>
<td>FLT_CW_SOFT_LIMIT</td>
<td>0x00000010</td>
<td>CW software limit error</td>
</tr>
<tr>
<td>FLT_CCW_SOFT_LIMIT</td>
<td>0x00000020</td>
<td>CCW software limit error</td>
</tr>
<tr>
<td>FLT_DRIVE</td>
<td>0x00000040</td>
<td>drive fault error</td>
</tr>
<tr>
<td>FLT_FEEDBACK</td>
<td>0x00000080</td>
<td>feedback failure error</td>
</tr>
<tr>
<td>FLT_PROGRAMMING</td>
<td>0x00000100</td>
<td>programming error</td>
</tr>
<tr>
<td>FLT_MASTER_FEEDBACK</td>
<td>0x00000200</td>
<td>master feedback failure</td>
</tr>
<tr>
<td>FLT_HOMING</td>
<td>0x00000400</td>
<td>homing fault</td>
</tr>
<tr>
<td>FLT_USER</td>
<td>0x00000800</td>
<td>user trigger</td>
</tr>
<tr>
<td>FLT_VEL_TRAP</td>
<td>0x00001000</td>
<td>velocity trap</td>
</tr>
<tr>
<td>FLT_VEL_COMMAND_TRAP</td>
<td>0x00002000</td>
<td>velocity command trap</td>
</tr>
<tr>
<td>FLT_HOME_TOLERANCE</td>
<td>0x00004000</td>
<td>home fault tolerance</td>
</tr>
<tr>
<td>FLT_PROBE</td>
<td>0x00008000</td>
<td>probe input fault</td>
</tr>
<tr>
<td>FLT_TASK</td>
<td>0x00010000</td>
<td>Task fault</td>
</tr>
<tr>
<td>FLT_EXTERNAL_FEEDBACK</td>
<td>0x00020000</td>
<td>external feedback failure</td>
</tr>
<tr>
<td>FLT_SAFEZONE</td>
<td>0x00040000</td>
<td>safe zone fault</td>
</tr>
</tbody>
</table>

Used by

Axis parameter FAULT
A.27. GLOBPARAM

Global parameter names.

Constants

- GLOBPARAM_AvgPollTimeSec 0
- GLOBPARAM_Version 1
- GLOBPARAM_NumGlobalDoubles 2
- GLOBPARAM_NumGlobalStrings 3
- GLOBPARAM_NumGlobalAxisPts 4
- GLOBPARAM_EstopEnabled 5
- GLOBPARAM_CallBackTimeoutSec 6
- GLOBPARAM_Interrupt2TimeSec 7
- GLOBPARAM_Enable1KhzServo 8
- GLOBPARAM_BuidlNumber 9
- GLOBPARAM_UserMode 10
- GLOBPARAM_ThrowTaskWarningsAsFaults 11
- GLOBPARAM_SystemStatus 12
- GLOBPARAM_Enable2Dcalibration 13
- GLOBPARAM_NumCannedFunctions 14
- GLOBPARAM_CompatibilityMode 15
- GLOBPARAM_NumDecimalsCompare 16
- GLOBPARAM_MeasurementMode 17

Used by

AerParmGlobalxxxx
A.28. HOMETYPE

Specifies the type of home routine to perform by the CNC home command.

Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HOMETYPE_NORMAL</td>
<td>0</td>
<td>Specifies a normal home cycle.</td>
</tr>
<tr>
<td>HOMETYPE.Reverse</td>
<td>1</td>
<td>Specifies a reverse to marker home cycle.</td>
</tr>
<tr>
<td>HOMETYPE_NOLIMIT</td>
<td>2</td>
<td>Specifies a home cycle with no limits.</td>
</tr>
<tr>
<td>HOMETYPE_QUICK</td>
<td>3</td>
<td>Specifies a quick home cycle.</td>
</tr>
<tr>
<td>HOMETYPE_VIRTUAL</td>
<td>4</td>
<td>Home at correct position</td>
</tr>
</tbody>
</table>

Used by

MACHPARM.HomeType
A.29. MACHPARM

Machine Parameter names.

Constants

- MACHPARM_Type
- MACHPARM_CntsPerInch
- MACHPARM_CntsPerDeg
- MACHPARM_MaxFeedRateIPM
- MACHPARM_MaxFeedRateRPM
- MACHPARM_RapidFeedRateIPM
- MACHPARM_RapidFeedRateRPM
- MACHPARM_HomeType
- MACHPARM_HomeDirection
- MACHPARM_HomeFeedRateIPM
- MACHPARM_HomeFeedRateRPM
- MACHPARM_HomeOffsetInch
- MACHPARM_HomeOffsetDeg
- MACHPARM_NumDecimalsEnglish
- MACHPARM_NumDecimalsMetric
- MACHPARM_AxisState
- MACHPARM_ControllingTask
- MACHPARM_PositionUnits
- MACHPARM_PositionCmdUnits
- MACHPARM_PresetCmdUnits
- MACHPARM_AvgVelUnits
- MACHPARM_FixtureOffset
- MACHPARM_FixtureOffset2
- MACHPARM_FixtureOffset3
- MACHPARM_FixtureOffset4
- MACHPARM_FixtureOffset5
- MACHPARM_FixtureOffset6
- MACHPARM_JogDistanceInch
- MACHPARM_JogDistanceDeg
- MACHPARM_JogVelocityIPM
- MACHPARM_JogVelocityRPM
- MACHPARM_UnusedAxis
- MACHPARM_ReverseSlewDir

Used by

AerParmMachinexxxx
A.30. MAX

These constants specify various maximum lengths of data used by the functions.

Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAX_PATH</td>
<td>261</td>
</tr>
<tr>
<td>MAX_TEXT_LEN</td>
<td>255</td>
</tr>
<tr>
<td>MAX_PARM_NAME_LEN</td>
<td>32</td>
</tr>
<tr>
<td>MAX_TOOL_NAME_LEN</td>
<td>32</td>
</tr>
</tbody>
</table>

// maximum parameter name lengths
// maximum tool name lengths
A.31. MOTION

To determine the motion status of an axis.

Constants

// Alternate Status word returned

MOTION_STAT_MOVE_DIR 0x00000001  // move direction
MOTION_STAT_MOVING 0x00000002  // moving
MOTION_STAT_ACCEL 0x00000004  // axis in accel phase
MOTION_STAT_DECEL 0x00000008  // axis in decel phase
MOTION_STAT_HOMING 0x00000010  // axis homing
MOTION_STAT_FEED_OVERRIDE 0x00000020  // feedrate override
MOTION_STAT_PROFILE 0x00000040  // axis in profile mode
MOTION_STAT_SYNC 0x00000080  // axis in sync mode
MOTION_STAT_CAM_TABLE 0x00000100  // cam table enabled
MOTION_STAT_HOME_DIR 0x00000200  // home direction
MOTION_STAT_CONT_MOVE 0x00000400  // continuous move
MOTION_STAT_QUEUE 0x00000800  // motion queue active
MOTION_STAT_HOLD 0x00001000  // hold active
MOTION_STAT_AUX_MODE 0x00002000  // aux mode
MOTION_STAT_BLOCK 0x00004000  // block motion
MOTION_STAT_HOLD_QUEUE 0x00008000  // hold queue
MOTION_STAT_DISABLE 0x00010000  // disable command
MOTION_STAT_HALT 0x00020000  // halt command
MOTION_STAT_ABORT 0x00040000  // abort command
MOTION_STAT_ACCEL_ON 0x00080000  // accel command
MOTION_STAT_DECEL_ON 0x00100000  // decel enabled
MOTION_STAT_ACCEL_SIGN 0x00200000  // accel sign used for dir change
MOTION_STAT_CONST_ACCEL 0x00400000  // linear/1-cosine accel flag
MOTION_STAT_CONST_DECEL 0x00800000  // linear/1-cosine decel flag
MOTION_STAT_BOUNDED 0x01000000  // bounded i.e., use soft limits
MOTION_STAT_SETUP_PEND 0x02000000  // setup command pending
MOTION_STAT_CHKRR_FLAG 0x04000000  // set along with setup_pend & cleared when checker runs
MOTION_STAT_QUICK_HOME 0x08000000  // quick home active
MOTION_STAT_IRQ_PENDING 0x10000000  // interrupt pending
MOTION_STAT_PENDANT_JOG 0x20000000  // pendant jog mode active
MOTION_STAT_MKRR_ARMED 0x40000000  // marker armed
MOTION_STAT_JOG 0x80000000  // Jog

Used by

Axis parameter MOTIONSTATUS
A.32. PARM_ATTR

These values can be binary “anded” with the attribute mask returned by the AerParmxxxxGetInfo functions, to determine if the conditions listed in the comments below are true or not.

Constants

<table>
<thead>
<tr>
<th>CONSTANT</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PARM_ATTR_VALID</td>
<td>0x0001</td>
<td>This is a valid parameter</td>
</tr>
<tr>
<td>PARM_ATTR_READ</td>
<td>0x0002</td>
<td>This parameter can be read (Get can be done)</td>
</tr>
<tr>
<td>PARM_ATTR_WRITE</td>
<td>0x0004</td>
<td>This parameter can be written (Set can be done)</td>
</tr>
<tr>
<td>PARM_ATTR_UPDATE</td>
<td>0x0008</td>
<td>The axis processor may change this value. For example, CLOCK is updated continuously</td>
</tr>
<tr>
<td>PARM_ATTR_NOLIMIT</td>
<td>0x0200</td>
<td>This parameter has no minimum and maximum values</td>
</tr>
<tr>
<td>PARM_ATTR_TEST</td>
<td>0x0020</td>
<td>Reserved for internal use, this bit has no meaning to the customer.</td>
</tr>
<tr>
<td>PARM_ATTR_INTEGER</td>
<td>0x0040</td>
<td>This parameter is a DWORD or double word integer (other wise its a double, or floating point)</td>
</tr>
<tr>
<td>PARM_ATTR_ENGLISH</td>
<td>0x0100</td>
<td>Value is in units of English units (inches) as opposed to metric (mil). This mask is only relevant for position, velocities etc. and it is also zero if the metric/Eng designation is not relevant to the parameter (i.e. NUM_DOUBLES)</td>
</tr>
<tr>
<td>PARM_ATTR_UNSIGNED</td>
<td>0x0010</td>
<td>Value is an unsigned quantity</td>
</tr>
<tr>
<td>PARM_ATTR_BITMASK</td>
<td>0x0400</td>
<td>display as bit mapped</td>
</tr>
<tr>
<td>PARM_ATTR_CHOICE</td>
<td>0x0800</td>
<td>display as choice</td>
</tr>
<tr>
<td>PARM_ATTR_HEX</td>
<td>0x1000</td>
<td>display as a HEX value</td>
</tr>
</tbody>
</table>

Used by

AerParmAxisGetInfo
AerParmMachineGetInfo
AerParmGlobalGetInfo
AerParmTaskGetInfo
### A.33. PHYSAXISINDEX

These constants specify a physical axis index.

**Constants**

<table>
<thead>
<tr>
<th>PHYSAXISINDEX_1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>PHYSAXISINDEX_2</td>
<td>1</td>
</tr>
<tr>
<td>PHYSAXISINDEX_3</td>
<td>2</td>
</tr>
<tr>
<td>PHYSAXISINDEX_4</td>
<td>3</td>
</tr>
<tr>
<td>PHYSAXISINDEX_5</td>
<td>4</td>
</tr>
<tr>
<td>PHYSAXISINDEX_6</td>
<td>5</td>
</tr>
<tr>
<td>PHYSAXISINDEX_7</td>
<td>6</td>
</tr>
<tr>
<td>PHYSAXISINDEX_8</td>
<td>7</td>
</tr>
<tr>
<td>PHYSAXISINDEX_9</td>
<td>8</td>
</tr>
<tr>
<td>PHYSAXISINDEX_10</td>
<td>9</td>
</tr>
<tr>
<td>PHYSAXISINDEX_11</td>
<td>10</td>
</tr>
<tr>
<td>PHYSAXISINDEX_12</td>
<td>11</td>
</tr>
<tr>
<td>PHYSAXISINDEX_13</td>
<td>12</td>
</tr>
<tr>
<td>PHYSAXISINDEX_14</td>
<td>13</td>
</tr>
<tr>
<td>PHYSAXISINDEX_15</td>
<td>14</td>
</tr>
<tr>
<td>PHYSAXISINDEX_16</td>
<td>15</td>
</tr>
</tbody>
</table>
A.34. PROGTYPE

Specifies the type of program to allocate.

Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROGTYPE_NORMAL</td>
<td>0</td>
<td>Normal program. All the program lines and labels must be loaded before the program can be executed. This type of program can be called by other programs or subroutines. There are limitations on the types of program lines allowed. The program remains intact on the controller until it is freed.</td>
</tr>
<tr>
<td>PROGTYPE_QUEUE</td>
<td>1</td>
<td>Queue program. This type of program can only be executed on one task at a time. No calls can be made to a queue program. There are limitations on the type of program lines allowed. Once executed the program line is lost. This type of program is used most often for MDI, serial/ethernet communication, or extremely large programs that cannot be completely allocated on the controller.</td>
</tr>
</tbody>
</table>

Used by

AerProgramAllocate
A.35. PSO_CHANNEL_MASK

These constants are used for defining the channels that the PSO-PC card will track for generating the firing pulse, etc.

Constants

<table>
<thead>
<tr>
<th>Constant Name</th>
<th>Value</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSO_CHANNEL_MASK1</td>
<td>0x01</td>
<td>PSO channel 1</td>
</tr>
<tr>
<td>PSO_CHANNEL_MASK2</td>
<td>0x02</td>
<td>PSO channel 2</td>
</tr>
<tr>
<td>PSO_CHANNEL_MASK3</td>
<td>0x04</td>
<td>PSO channel 3</td>
</tr>
<tr>
<td>PSO_CHANNEL_MASK4</td>
<td>0x08</td>
<td>PSO channel 4</td>
</tr>
</tbody>
</table>
A.36. PSO_MODE

These constants are used for defining the mode of the PSO-PC card.

Constants

- PSO_MODE_IMMEDIATE 0x00  // write output value NOW
- PSO_MODE_VELTRACKING 0x01  // track velocity
- PSO_MODE_POSTRACKING 0x02  // track position
A.37. **PSO_TABLE**

These constants are used for defining the mode of the PSO-PC table based commands.

**Constants**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSO_TABLE_INCDIST</td>
<td>0x00</td>
<td>incremental distances</td>
</tr>
<tr>
<td>PSO_TABLE_ABSDIST</td>
<td>0x01</td>
<td>absolute distances</td>
</tr>
<tr>
<td>PSO_TABLE_INVALID</td>
<td>0xFFFF</td>
<td>invalid firing table type</td>
</tr>
</tbody>
</table>
A.38. **PULSE_**

These constants are used for defining the mode of the PSO CNC command.

**Constants**

- `PULSE_WIDTH_INCDIST` 0x00 // define pulse width in mSec (PSOP 0)
- `PULSE_WIDTH_ABSDIST` 0x01 // define pulse width in uSec (PSOP 4)
- `PULSE_DEFINE_INVALID` 0xFFFF // define 3 phase pulse (PSOP 1)
- `PULSE_DEFINE_RAMP` 0xFFFF // define 3 phase pulse w/ ramp (PSOP 2)
- `PULSE_TOGGLE` 0xFFFF // (PSOP 5)
### A.39. SERVO

These constants are used to test the state of the SERVOSTAT axis parameter, by sequentially (as required) ANDing one of these constants with the value of the parameter and testing for the result to be TRUE.

#### Constants

// Servo Status word

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SERVO_STAT_DRIVE</td>
<td>0x00000001</td>
<td>drive on, off</td>
</tr>
<tr>
<td>SERVO_STAT_AUX</td>
<td>0x00000002</td>
<td>auxiliary output on, off</td>
</tr>
<tr>
<td>SERVO_STAT_CW_LIMIT</td>
<td>0x00000004</td>
<td>cw limit switch on, off</td>
</tr>
<tr>
<td>SERVO_STAT_CCW_LIMIT</td>
<td>0x00000008</td>
<td>ccw limit switch on, off</td>
</tr>
<tr>
<td>SERVO_STAT_HOME</td>
<td>0x00000010</td>
<td>home switch on, off</td>
</tr>
<tr>
<td>SERVO_STAT_DRIVE_FLT</td>
<td>0x00000020</td>
<td>drive fault status</td>
</tr>
<tr>
<td>SERVO_STAT_ATHOME</td>
<td>0x00000040</td>
<td>axis at home position</td>
</tr>
<tr>
<td>SERVO_STAT_DONE</td>
<td>0x00000080</td>
<td>motion done</td>
</tr>
<tr>
<td>SERVO_STAT_INPOS</td>
<td>0x00000100</td>
<td>axis in position</td>
</tr>
<tr>
<td>SERVO_STAT_FAULTED</td>
<td>0x00000200</td>
<td>axis is faulted</td>
</tr>
<tr>
<td>SERVO_STAT_PROBE_INPUT</td>
<td>0x00000400</td>
<td>probe input active</td>
</tr>
<tr>
<td>SERVO_STAT_MARKER</td>
<td>0x00000800</td>
<td>marker</td>
</tr>
<tr>
<td>SERVO_STAT_HALL1</td>
<td>0x00001000</td>
<td>hall input 1</td>
</tr>
<tr>
<td>SERVO_STAT_HALL2</td>
<td>0x00002000</td>
<td>hall input 2</td>
</tr>
<tr>
<td>SERVO_STAT_HALL3</td>
<td>0x00004000</td>
<td>hall input 3</td>
</tr>
<tr>
<td>SERVO_STAT_HALL4</td>
<td>0x00008000</td>
<td>hall input 4</td>
</tr>
<tr>
<td>SERVO_STAT_INEG_LIMIT</td>
<td>0x00010000</td>
<td>in integral negative limit</td>
</tr>
<tr>
<td>SERVO_STAT_IPOS_LIMIT</td>
<td>0x00020000</td>
<td>in integral positive limit</td>
</tr>
<tr>
<td>SERVO_STAT_VFF</td>
<td>0x00040000</td>
<td>VFF Enabled</td>
</tr>
<tr>
<td>SERVO_STAT_BRAKE</td>
<td>0x00080000</td>
<td>brake output active</td>
</tr>
<tr>
<td>SERVO_STAT_ALIVE</td>
<td>0x01000000</td>
<td>axis has been configured</td>
</tr>
<tr>
<td>SERVO_STAT_VVF_0ATC</td>
<td>0x02000000</td>
<td>VFF or position loop zero</td>
</tr>
<tr>
<td>SERVO_STAT_FEEDBACK_IN</td>
<td>0x04000000</td>
<td>Feedback fault input</td>
</tr>
<tr>
<td>SERVO_STAT_MFEEDBACK_IN</td>
<td>0x08000000</td>
<td>Mst Feedback fault input</td>
</tr>
<tr>
<td>SERVO_STAT_HPVMERGE_LASER</td>
<td>0x10000000</td>
<td>HP VME Laser</td>
</tr>
<tr>
<td>SERVO_STAT_SCALE_PGAiN</td>
<td>0x20000000</td>
<td>SCALPGAIN axis parameter</td>
</tr>
<tr>
<td>SERVO_STAT_AC</td>
<td>0x40000000</td>
<td>AC motor selected</td>
</tr>
<tr>
<td>SERVO_STAT_MSET</td>
<td>0x80000000</td>
<td>Axis in MSET mode</td>
</tr>
<tr>
<td>SERVO_STAT_HOMED</td>
<td>0x10000000</td>
<td>Axis has been homed</td>
</tr>
<tr>
<td>SERVO_STAT_ENCODER</td>
<td>0x20000000</td>
<td>Axis has encoder feedback</td>
</tr>
<tr>
<td>SERVO_STAT_ERROR_MAP</td>
<td>0x40000000</td>
<td>Error mapping enabled</td>
</tr>
<tr>
<td>SERVO_STAT_PLOOP_ONLY</td>
<td>0x80000000</td>
<td>position loop only</td>
</tr>
</tbody>
</table>

#### Used by

Axis parameter SERVOSTATUS
A.40. SPINDLEINDEX

Spindle index numbers.

Constants

SPINDLEINDEX\_n where n is the spindle number in the range of 1-4, having a value of 0 to 3.

Used by

SPINDLEINDEX variables

Compiler
A.41. STAT

To determine the state of the axis processor.

Constants

---

// Axis status bits defined
STAT_DRIVE 0x00000001 // drive on, off
STAT_AUX 0x00000002 // auxiliary output on, off
STAT_CW_LIMIT 0x00000004 // cw limit switch on, off
STAT_CCW_LIMIT 0x00000008 // ccw limit switch on, off
STAT_HOME 0x00000010 // home switch on, off
STAT_DRIVE_FLT 0x00000020 // drive fault status
STAT_ATHOME 0x00000040 // axis at home position
STAT_DONE 0x00000080 // motion done
STAT_INPOS 0x00000100 // axis in position
STAT_FAULTED 0x00000200 // axis is faulted
STAT_PROBE_INPUT 0x00000400 // probe input active
STAT_MARKER 0x00000800 // marker
STAT_HALL1 0x00001000 // hall input 1
STAT_HALL2 0x00002000 // hall input 2
STAT_HALL3 0x00004000 // hall input 3
STAT_HALL4 0x00008000 // hall input 4
STAT_MOVE_DIR 0x00010000 // move direction
STAT_MOVING 0x00020000 // moving
STAT_ACCEL 0x00040000 // axis in accel phase
STAT_DECEL 0x00080000 // axis in decel phase
STAT_HOMING 0x00100000 // axis homing
STAT_FEED_OVER 0x00200000 // feedrate override
STAT_PROFILE 0x00400000 // axis in profile mode
STAT_SYNC 0x00800000 // axis in sync mode
STAT_CAM_TABLE 0x01000000 // cam table enabled
STAT_HOME_DIR 0x02000000 // home direction
STAT.Continue 0x04000000 // continuous move
STAT_QUEUE 0x08000000 // motion queue a active
STAT_HOLD 0x10000000 // hold active
STAT_AUX_MODE 0x20000000 // aux mode
STAT_BLOCK_MOTION 0x40000000 // block motion
STAT_HOLD_QUEUE 0x80000000 // hold queue
---

STAT_DONE is set true after the commanded move is complete. This does not indicate that the actual velocity is zero due to the following errors, which may be present (see the POSERR axis parameter).

STAT_INPOS is set true after the completion of the move and when the axis is within the range defined by the INPOSLIMIT axis parameter.

Used by

Axis parameter STATUS
A.42. STRIP_IOPOSLATCH

These constants are position latch triggers.

Constants

// trigger mode (wMode)
STRIP_IOPOSLATCH_DISABLE 0 // IO position latch disable
STRIP_IOPOSLATCH_ONESHOT 1 // One-shot operation
STRIP_IOPOSLATCH_CONTINUOUS 2 // Continuous trigger

// type of I/O to trigger (wType)
STRIP_IOPOSLATCH_AXISIO 0 // Use axis I/O
STRIP_IOPOSLATCH_VIRTUALIO 1 // Use virtual I/O bit

// axis I/O bit number(wBit)
STRIP_IOPOSLATCH_BITCCW 0 // Use CCW input
STRIP_IOPOSLATCH_BITCW 1 // Use CW input
STRIP_IOPOSLATCH_BITHOME 2 // Use home input
STRIP_IOPOSLATCH_BITENCFLT 3 // Use encoder fault input
STRIP_IOPOSLATCH_BITFLT 4 // Use drive fault input
STRIP_IOPOSLATCH_BITHALLA 5 // Use Hall A input
STRIP_IOPOSLATCH_BITHALLB 6 // Use Hall B input
STRIP_IOPOSLATCH_BITHALLC 7 // Use Hall C input

Used by

AerSripSetIOPosLatch
A.43. STRIP_STATUS

These constants are used to determine the status of the strip chart.

**Constants**

- STRIP_STATUS_ALLOCATED 0x01 // Allocated
- STRIP_STATUS_ARMED 0x02 // Armed (See AerStripSetTrigger)
- STRIP_STATUS_TRIGGERED 0x04 // Triggered (collecting)
- STRIP_STATUS_DONE 0x08 // Done collecting
- STRIP_STATUS_OVERFLOW 0x10 // Strip chart overflow

**Used By**

*AerStripGetStatus*
A.44. STRIP_TRIGGER

These constants specify the different modes for triggering the strip chart.

**Constants**

- **STRIP_TRIGGER_IMMEDIATE** 0 // Enables immediate trigger
  // (collection starts now)
- **STRIP_TRIGGER_MASTER_POS** 1 // Master position trigger (starts
  // if master position is above or
  // below a value)
- **STRIP_TRIGGER_POINT0** 2 // Trigger with table
- **STRIP_TRIGGER_TORQUE** 3 // Trigger with torque
- **STRIP_TRIGGER_EXTERNAL** 4 // Program trigger

**Used By**

*AerStripSetTrigger*
A.45. STRIPGLOBAL_MODE

These constants specify the different modes for triggering the global strip chart.

**Constants**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STRIPGLOBAL_MODE_TIME</td>
<td>0</td>
<td>Start immediately</td>
</tr>
<tr>
<td>STRIPGLOBAL_MODE_TIME_SPEED_OVERRIDE</td>
<td>1</td>
<td>Use analog input</td>
</tr>
<tr>
<td>STRIPGLOBAL_MODE_POSITION</td>
<td>2</td>
<td>Trigger on position</td>
</tr>
<tr>
<td>STRIPGLOBAL_MODE_QUEUE</td>
<td>3</td>
<td>Continuous data</td>
</tr>
<tr>
<td>STRIPGLOBAL_MODE_QUEUE_HOLD</td>
<td>4</td>
<td>Wait for release</td>
</tr>
<tr>
<td>STRIPGLOBAL_MODE_EVENT_COUNT</td>
<td>5</td>
<td>Triggered by Probe</td>
</tr>
<tr>
<td>STRIPGLOBAL_MODE_EVENT_QUEUE</td>
<td>6</td>
<td>Continuous data, triggered by Probe</td>
</tr>
<tr>
<td>STRIPGLOBAL_MODE_IO</td>
<td>7</td>
<td>Triggered by virtual input</td>
</tr>
<tr>
<td>STRIPGLOBAL_MODE_IO_QUEUE</td>
<td>8</td>
<td>Continuous data, triggered by virtual input</td>
</tr>
</tbody>
</table>

Used by

AerStripGlobalSetTrigger
A.46. STRIPGLOBAL_STATUS

Used to determine the status of the global strip chart.

**Constants**

- `STRIPGLOBAL_STATUS_ALLOCATED`: 0x0001 // Allocated (see AerStripGlobalAllocate)
- `STRIPGLOBAL_STATUS_ARMED`: 0x0002 // Armed (see AerStripGlobalSetTrigger)
- `STRIPGLOBAL_STATUS_TRIGGERED`: 0x0004 // Triggered (collecting)
- `STRIPGLOBAL_STATUS_DONE`: 0x0008 // Done collecting
- `STRIPGLOBAL_STATUS_OVERFLOW`: 0x0010 // Strip chart overflow
- `STRIPGLOBAL_STATUS_FR_MODE`: 0x0020 // Sample time based on analog input #8
- `STRIPGLOBAL_STATUS_HOLD`: 0x0040 // Queue is currently held
- `STRIPGLOBAL_STATUS_TABLE_MODE`: 0x0080 // Triggered (in list mode)
- `STRIPGLOBAL_STATUS_TRIG_ALLOCATED`: 0x0100 // Trigger list is allocated
- `STRIPGLOBAL_STATUS_QUEUE_MODE`: 0x0200 // In queue mode (see AerStripGlobalSetTrigger)
- `STRIPGLOBAL_STATUS_ABORTED`: 0x0400
- `STRIPGLOBAL_STATUS_EVENT_MODE`: 0x0800
- `STRIPGLOBAL_STATUS_UPLOADING`: 0x1000
- `STRIPGLOBAL_STATUS_LOGIC_TIME`: 0x2000
- `STRIPGLOBAL_STATUS_LOGIC_LEVEL`: 0x4000
- `STRIPGLOBAL_STATUS_4KHZ`: 0x8000

**Used by**

- AerStripGlobalGetStatus
A.47. TASKAXISINDEX

Task axis index numbers.

Constants

TASKAXISINDEX_xxxx where xxxx is the task axis index number in the range of 1-16, having a value of 0 to 15. Task axes are mapped to physical axes by the MAP CNC command. These constants are used for TASKAXIS variables.

Used by

TASKAXISINDEX variables
A.48. TASKEXEC

Specifies the type of program execution to begin.

Constants

<table>
<thead>
<tr>
<th>TASKEXEC_RUN / TASKEXEC_RUN_INT O</th>
<th>0 // The task will execute // the program continuously.</th>
</tr>
</thead>
<tbody>
<tr>
<td>TASKEXEC_STEP_IN T O</td>
<td>1 // The task will execute one // user line of code. Calls // are treated as multiple // lines. Program execution // will stop after entering // the called subroutine.</td>
</tr>
<tr>
<td>TASKEXEC_STEP_OVER</td>
<td>2 // The task will execute one // user line of code. Calls // are treated as a single // line. Program execution // will continue after // entering the called // subroutine until // program flow returns to // the original call // statement.</td>
</tr>
<tr>
<td>TASKEXEC_RUN_OVER</td>
<td>3 // Will execute subroutines // as though they were a // single command, displaying // only the call subroutine // command line</td>
</tr>
</tbody>
</table>

Used by

_AerTaskProgramExecute_
A.49. TASKINDEX

Task index numbers.

**Constants**

TASKINDEX\_n where n is the task number in the range of 1-4, having a value of 0 to 3.

**Used by**

- TASKINDEX variables
- AerTaskxxx functions
A.50. TASKMODE

To determine which task mode is active. Parentheses indicate the interpretation of the mode when it is not active.

Constants

<table>
<thead>
<tr>
<th>Constant Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TASKMODE1_English</td>
<td>0</td>
<td>English units (metric) G70 (G71)</td>
</tr>
<tr>
<td>TASKMODE1_Absolute</td>
<td>1</td>
<td>Absolute mode (incremental) G90 (G91)</td>
</tr>
<tr>
<td>TASKMODE1_AccelModeLinear</td>
<td>2</td>
<td>Linear accel/decel (sinusoidal) G64 (G63)</td>
</tr>
<tr>
<td>TASKMODE1_AccelModeRate</td>
<td>3</td>
<td>Rate based accel/decel (time) G68 (G67)</td>
</tr>
<tr>
<td>TASKMODE1_RotaryDominant</td>
<td>4</td>
<td>Rotary feedrate dominant (linear) G98 (G99)</td>
</tr>
<tr>
<td>TASKMODE1_MotionContinuous</td>
<td>5</td>
<td>Continuous motion between blocks (decel) G108 (G109)</td>
</tr>
<tr>
<td>TASKMODE1_InverseCircular</td>
<td>6</td>
<td>Inversed circular direction (normal) G111 (G110)</td>
</tr>
<tr>
<td>TASKMODE1_SpindleShutDown</td>
<td>7</td>
<td>Enable shutdown spindle on program halt G101 (G100)</td>
</tr>
<tr>
<td>TASKMODE1_BlockDelete</td>
<td>8</td>
<td>Program block deletes are used (ignored)</td>
</tr>
<tr>
<td>TASKMODE1_OptionalStop</td>
<td>9</td>
<td>Program optional stops are used (ignored)</td>
</tr>
<tr>
<td>TASKMODE1_BreakPoint</td>
<td>10</td>
<td>Program break points are used (ignored)</td>
</tr>
<tr>
<td>TASKMODE1_MFOLock</td>
<td>11</td>
<td>MFO is locked - can not be changed M48 (locked) M49</td>
</tr>
<tr>
<td>TASKMODE1_MSOLock</td>
<td>12</td>
<td>MSO is locked - can not be changed M50 (locked) M51</td>
</tr>
<tr>
<td>TASKMODE1_DryRunFeedRate</td>
<td>13</td>
<td>Vector feedrate is limited by axes</td>
</tr>
<tr>
<td>TASKMODE1_Retrace</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>TASKMODE1_AutoMode</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>TASKMODE1_ProgramFeedRateMPU</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>TASKMODE1_ProgramFeedRateUPR</td>
<td>17</td>
<td></td>
</tr>
<tr>
<td>TASKMODE1_ProgramFeedRateSurf1</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>TASKMODE1_ProgramFeedRateSurf2</td>
<td>19</td>
<td></td>
</tr>
<tr>
<td>TASKMODE1_ProgramFeedRateSurf3</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>TASKMODE1_ProgramFeedRateSurf4</td>
<td>21</td>
<td></td>
</tr>
<tr>
<td>TASKMODE1_BlockDelete2</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>TASKMODE1_RunOverMode</td>
<td>23</td>
<td></td>
</tr>
<tr>
<td>TASKMODE1_MultiBlockLookAhead</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>TASKMODE1_MachineLock</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>TASKMODE1_HighSpeedBuffering</td>
<td>26</td>
<td></td>
</tr>
<tr>
<td>TASKMODE1_WaitForInPos</td>
<td>27</td>
<td></td>
</tr>
</tbody>
</table>

Used by

AerTaskGetModeName
AER_TASK_MODE
A.51. TASKPARAM

Task Parameter names.

Constants

- TASKPARAM_Number
- TASKPARAM_TaskFault
- TASKPARAM_TaskWarning
- TASKPARAM_MaxCallStack
- TASKPARAM_MaxModeStack
- TASKPARAM_NumTaskDoubles
- TASKPARAM_NumTaskStrings
- TASKPARAM_NumTaskAxisPts
- TASKPARAM_EstopInput
- TASKPARAM_FeedHoldInput
- TASKPARAM_FeedHoldEdgeInput
- TASKPARAM_S1_Index
- TASKPARAM_S1_RPM
- TASKPARAM_S2_Index
- TASKPARAM_S2_RPM
- TASKPARAM_S3_Index
- TASKPARAM_S3_RPM
- TASKPARAM_S4_Index
- TASKPARAM_S4_RPM
- TASKPARAM_RotateX
- TASKPARAM_RotateY
- TASKPARAM_RotateAngleDeg
- TASKPARAM_RthetaX
- TASKPARAM_RthetaY
- TASKPARAM_RthetaR
- TASKPARAM_RthetaT
- TASKPARAM_RthetaRadiusInch
- TASKPARAM_RthetaEnabled
- TASKPARAM_UpdateTimeSec
- TASKPARAM_AccelTimeSec
- TASKPARAM_DecelTimeSec
- TASKPARAM_AccelRateIPS2
- TASKPARAM_DecelRateIPS2
- TASKPARAM_AccelRateDPS2
- TASKPARAM_DecelRateDPS2
- TASKPARAM_LinearFeedRate
- TASKPARAM_RotaryFeedRate
- TASKPARAM_MFO
- TASKPARAM_MSO
- TASKPARAM_Coord1X
- TASKPARAM_Coord1Y
- TASKPARAM_Coord1Z
- TASKPARAM_Coord1Plane
- TASKPARAM_Coord2X
- TASKPARAM_Coord2Y
- TASKPARAM_Coord2Z
- TASKPARAM_Coord2Plane
- TASKPARAM_CutterX
- TASKPARAM_CutterY
- TASKPARAM_CutterRadiusInch
- TASKPARAM_NormalcyX
- TASKPARAM_NormalcyY
- TASKPARAM_NormalcyAxis
- TASKPARAM_UserFeedRateMode
- TASKPARAM_MaxMonitorData
- TASKPARAM_MaxOnGosubData
- TASKPARAM_AnalogMFOInput
TASKPARM_FeedHold  57
TASKPARM_MaxRadiusError  58
TASKPARM_Status1  59
TASKPARM_Status2  60
TASKPARM_Status3  61
TASKPARM_Model  62
TASKPARM_AnalogMSOInput  63
TASKPARM_ErrCode  64
TASKPARM_GlobalEStopDisable  65
TASKPARM_LinearFeedRateActual  66
TASKPARM_RotaryFeedRateActual  67
TASKPARM_HaltTaskOnAxisFault  68
TASKPARM_InterruptMotion  69
TASKPARM_InterruptMotionReturnType  70
TASKPARM_S2_AnalogMSOInput  71
TASKPARM_S3_AnalogMSOInput  72
TASKPARM_S4_AnalogMSOInput  73
TASKPARM_S2_MSO  74
TASKPARM_S3_MSO  75
TASKPARM_S4_MSO  76
TASKPARM_ROReq1  77
TASKPARM_RIAction1  78
TASKPARM_RIActionParm1  79
TASKPARM_JoyStickPort  80
TASKPARM_SlewPair1  81
TASKPARM_SlewPair2  82
TASKPARM_SlewPair3  83
TASKPARM_SlewPair4  84
TASKPARM_SlewPair5  85
TASKPARM_SlewPair6  86
TASKPARM_SlewPair7  87
TASKPARM_SlewPair8  88
TASKPARM_RIActionOpCode  89
TASKPARM_RIActionAxis  90
TASKPARM_RIActionParm1  91
TASKPARM_RIActionParm2  92
TASKPARM_S1_SpindleRadialAxis  93
TASKPARM_S2_SpindleRadialAxis  94
TASKPARM_S3_SpindleRadialAxis  95
TASKPARM_S4_SpindleRadialAxis  96
TASKPARM_BlendMaxAccelLinearIPS2  97
TASKPARM_BlendMaxAccelRotaryDPS2  98
TASKPARM_BlendMaxAccelCircleIPS2  99
TASKPARM_UNUSED1  100
TASKPARM_ActiveFixtureOffset  101
TASKPARM_ExecuteNumLines  102
TASKPARM_JogPair1EnableIn  103
TASKPARM_JogPair1Mode  104
TASKPARN_JogPair1Axis1  105
TASKPARN_JogPair1Axis1PlusIn  106
TASKPARN_JogPair1Axis1MinusIn  107
TASKPARN_JogPair1Axis2  108
TASKPARN_JogPair1Axis2PlusIn  109
TASKPARN_JogPair1Axis2MinusIn  110
TASKPARN_JogPair2EnableIn  111
TASKPARN_JogPair2Axis1  112
TASKPARN_JogPair2Axis2  113
TASKPARN_JogPair2Axis1PlusIn  114
TASKPARN_JogPair2Axis1MinusIn  115
TASKPARN_JogPair2Axis2PlusIn  116
TASKPARN_JogPair2Axis2MinusIn  117
TASKPARN_DryRunLinearFeedRateIPM  119
### TASKPARM List

<table>
<thead>
<tr>
<th>TASKPARM Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>TASKPARM_DryRunRotaryFeedRateRPM</td>
<td>120</td>
</tr>
<tr>
<td>TASKPARM_MaxLookAheadMoves</td>
<td>121</td>
</tr>
<tr>
<td>TASKPARM_LineNumberUser</td>
<td>122</td>
</tr>
<tr>
<td>TASKPARM_LineNumber960</td>
<td>123</td>
</tr>
<tr>
<td>TASKPARM_CannedFunctionID</td>
<td>124</td>
</tr>
<tr>
<td>TASKPARM_DecelOnProgramAbortMask</td>
<td>125</td>
</tr>
<tr>
<td>TASKPARM_IgnoreAxesMask</td>
<td>126</td>
</tr>
<tr>
<td>TASKPARM_ChordicalToleranceInch</td>
<td>127</td>
</tr>
<tr>
<td>TASKPARM_ROReq1Mask</td>
<td>128</td>
</tr>
<tr>
<td>TASKPARM_NormalcyToleranceDeg</td>
<td>129</td>
</tr>
<tr>
<td>TASKPARM_ChordicalSlowdownMsec</td>
<td>130</td>
</tr>
<tr>
<td>TASKPARM_CommandVelocityVariance</td>
<td>131</td>
</tr>
<tr>
<td>TASKPARM_CutterToleranceDeg</td>
<td>132</td>
</tr>
<tr>
<td>TASKPARM_CutterZ</td>
<td>133</td>
</tr>
<tr>
<td>TASKPARM_CutterLength</td>
<td>134</td>
</tr>
<tr>
<td>TASKPARM_CutterWear</td>
<td>135</td>
</tr>
<tr>
<td>TASKPARM_CutterOffsetX</td>
<td>136</td>
</tr>
<tr>
<td>TASKPARM_CutterOffsetY</td>
<td>137</td>
</tr>
<tr>
<td>TASKPARM_CutterRadius</td>
<td>138</td>
</tr>
<tr>
<td>TASKPARM_CutterActive</td>
<td>139</td>
</tr>
<tr>
<td>TASKPARM_RhethetaRadius</td>
<td>140</td>
</tr>
<tr>
<td>TASKPARM_AccelRate</td>
<td>141</td>
</tr>
<tr>
<td>TASKPARM_DecelRate</td>
<td>142</td>
</tr>
<tr>
<td>TASKPARM_MaxRadiusAdjust</td>
<td>143</td>
</tr>
<tr>
<td>TASKPARM_UpdateNumEntries</td>
<td>144</td>
</tr>
<tr>
<td>TASKPARM_MaxCornerRoundErr</td>
<td>145</td>
</tr>
<tr>
<td>TASKPARM_Group1GcodeMode</td>
<td>146</td>
</tr>
<tr>
<td>TASKPARM_ExecuteNumMonitors</td>
<td>147</td>
</tr>
<tr>
<td>TASKPARM_ExecuteNumSpindles</td>
<td>148</td>
</tr>
<tr>
<td>TASKPARM_ProfileAxesZeroVel</td>
<td>149</td>
</tr>
<tr>
<td>TASKPARM_ProfileAxesInpos</td>
<td>150</td>
</tr>
</tbody>
</table>

**Used by**

AerParmTaskxxxx
A.52. TASKSTATUS

To determine which task status is active. TaskStatus, which is a result of a G-code, is identified with the appropriate status bit.

**Constants**

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TASKSTATUS1_ProgramAssociated</td>
<td>0 // At least one program is with the task</td>
</tr>
<tr>
<td>TASKSTATUS1_ProgramActive</td>
<td>1 // The task is active - program execution was started</td>
</tr>
<tr>
<td>TASKSTATUS1_ProgramExecuting</td>
<td>2 // The task is currently executing a program</td>
</tr>
<tr>
<td>TASKSTATUS1_ImmediateCodeExecuting</td>
<td>3 // The task is currently executing an immediate code</td>
</tr>
<tr>
<td>TASKSTATUS1_ReturnMotionExecuting</td>
<td>4 // The task is currently executing a return motion</td>
</tr>
<tr>
<td>TASKSTATUS1_NotUsed</td>
<td>5 //</td>
</tr>
<tr>
<td>TASKSTATUS1_SingleStepInto</td>
<td>6 // The task is executing a STEP_INTO</td>
</tr>
<tr>
<td>TASKSTATUS1_SingleStepOver</td>
<td>7 // The task is executing a STEP_OVER</td>
</tr>
<tr>
<td>TASKSTATUS1_InterruptFaultPending</td>
<td>8 // A fault interrupt is pending</td>
</tr>
<tr>
<td>TASKSTATUS1_InterruptCallBackPending</td>
<td>9 // A program callback interrupt is pending</td>
</tr>
<tr>
<td>TASKSTATUS1_EStopInputActive</td>
<td>10 // The task estop is on</td>
</tr>
<tr>
<td>TASKSTATUS1_FeedHoldInputActive</td>
<td>11 // The task feedhold is on</td>
</tr>
<tr>
<td>TASKSTATUS1_CallBackHoldActive</td>
<td>12 // The task is waiting for a program callback to complete</td>
</tr>
<tr>
<td>TASKSTATUS1_CallBackResponding</td>
<td>13 // The front-end is responding to a program callback</td>
</tr>
<tr>
<td>TASKSTATUS1_ProgramCleanup</td>
<td>14 // The task is cleaning up the currently executing program due to an abort or interrupt</td>
</tr>
<tr>
<td>TASKSTATUS1_ProgramCodeCleanup</td>
<td>15 // The task is cleaning up the currently executing program line due to an abort or interrupt</td>
</tr>
<tr>
<td>TASKSTATUS1_OnGosubPending</td>
<td>16 // An interrupt due to a ONGOSUB is pending</td>
</tr>
<tr>
<td>TASKSTATUS1_FeedHoldInputLatch</td>
<td>17</td>
</tr>
<tr>
<td>TASKSTATUS1_ProbeCycle</td>
<td>18</td>
</tr>
<tr>
<td>TASKSTATUS1_Retrace</td>
<td>19</td>
</tr>
<tr>
<td>TASKSTATUS1_InsertLinkMove</td>
<td>20</td>
</tr>
<tr>
<td>TASKSTATUS1_InterruptActive</td>
<td>21</td>
</tr>
<tr>
<td>TASKSTATUS1_SlewActive</td>
<td>22</td>
</tr>
<tr>
<td>TASKSTATUS1_CornerRounding</td>
<td>23</td>
</tr>
<tr>
<td>TASKSTATUS1_ROReq1Active</td>
<td>24</td>
</tr>
<tr>
<td>TASKSTATUS1_CannedFunctionPending</td>
<td>25</td>
</tr>
<tr>
<td>TASKSTATUS1_CannedFunctionActive</td>
<td>26</td>
</tr>
<tr>
<td>TASKSTATUS1_CannedFunctionExecuting</td>
<td>27</td>
</tr>
<tr>
<td>TASKSTATUS1_ProgramReset</td>
<td>28</td>
</tr>
<tr>
<td>TASKSTATUS2_SpindleActive1</td>
<td>0 // The first task spindle is active</td>
</tr>
<tr>
<td>TASKSTATUS2_SpindleActive2</td>
<td>1 // The second task spindle is active</td>
</tr>
<tr>
<td>TASKSTATUS2_SpindleActive3</td>
<td>2 // The third task spindle is active</td>
</tr>
</tbody>
</table>
// active
TASKSTATUS2_SpindleActive4 3 // The fourth task spindle is active
TASKSTATUS2_MSOW41Change 4 // The MSO has changed
TASKSTATUS2_SpindleFeedHoldActive 5 // The spindle feedhold is on
TASKSTATUS2_AsynchronousFeedHoldActive 6 // The asynch feedhold is on
TASKSTATUS2_CutterEnabling 7
TASKSTATUS2_CutterDisabling 8
TASKSTATUS2_CutterOffsetsEnablingPos 9
TASKSTATUS2_CutterOffsetsEnablingNeg 10
TASKSTATUS2_CutterOffsetsDisabling 11

TASKSTATUS3_RotationActive 0 // Parts rotation is active
TASKSTATUS3_RthetapolarActive 1 // Polar R-Theta
TASKSTATUS3_RthetacylindricalActive 2 // Cylindrical R-Theta
TASKSTATUS3_OffsetPresetActive 3 // Preset offsets are active
TASKSTATUS3_OffsetFixtureActive 4 // Fixture offsets are active
TASKSTATUS3_OffsetManualActive 5 // Manual offsets are active
TASKSTATUS3_MotionType1 6 // (G0) - Point to Point
TASKSTATUS3_MotionType2 7 // (G1) - Contour Motion
TASKSTATUS3_MotionActive 8 // Contour/Pt-to-Pt motion is executing
TASKSTATUS3_MotionContinuous 9 // Continuous motion is active
TASKSTATUS3_MotionFeedHoldActive 11 // The contour feedhold is on
TASKSTATUS3_CutterEnabling 12 // Cutter compensation is being enabled on the current move
TASKSTATUS3_CutterActiveLeft 13 // Cutter compensation is active on the left
TASKSTATUS3_CutterActiveRight 14 // Cutter compensation is active on the right
TASKSTATUS3_CutterDisabling 15 // Cutter compensation is being disabled on the current move
TASKSTATUS3_NormalcyActiveLeft 16 // Normalcy is active on the left (G21)
TASKSTATUS3_NormalcyActiveRight 17 // Normalcy is active on the right (G22)
TASKSTATUS3_NormalcyAlignment 18 // The normalcy axis is being aligned
TASKSTATUS3_ProgramFeedRateMPU 19 // Feedrates are in minutes per unit G94 (If ProgramFeedrateMPU and ProgramFeedrateUPR are off, then G93 is active)
TASKSTATUS3_ProgramFeedRateUPR 20 // Feedrates are in units per minute G95 (If ProgramFeedrateMPU and ProgramFeedrateUPR are off, then G93 is active)
TASKSTATUS3_LimitFeedRateActive 21 // The current feedrate is
// being limited
// by the axis maximums
TASKSTATUS3_LimitMFOActive 22 // The MFO is being limited
// to account for feedrate
// limits

TASKSTATUS3_Coord1Plane1Active 23 // (G17) Plane specified by
// Coord1X & Coord1Y is
// active

TASKSTATUS3_Coord1Plane2Active 24 // (G18) Plane specified by
// Coord1Y & Coord1Z is
// active

TASKSTATUS3_Coord1Plane3Active 25 // (G19) Plane specified by
// Coord1Z & Coord1X is
// active

TASKSTATUS3_Coord2Plane1Active 26 // (G27) Plane specified by
// Coord2X & Coord2Y is
// active

TASKSTATUS3_Coord2Plane2Active 27 // (G28) Plane specified by
// Coord2Y & Coord2Z is
// active

TASKSTATUS3_Coord2Plane3Active 28 // (G29) Plane specified by
// Coord2Z & Coord1X is
// active

TASKSTATUS3_MotionNoAccel 29
TASKSTATUS3_MirrorActive 30

Used by
AerTaskGetStatusName
AER_TASK_STATUS

∇ ∇ ∇
APPENDIX B: VISUAL BASIC CONSTANTS

In This Section:
• Description...........................................B-1

B.1. Description

The following section provides a quick reference of all the visual basic constants used within the libraries (AerSys.DLL, AerErr.DLL, and AerCmplr.DLL).

All of these constants are available with the MS VB programming environment by pressing the F2 key (“Object Browser”), and selecting the Aerotech System Library. If they are not present, see Section 1.4., Visual Basic Programming Quick Start.
B.2. aerAnalogIndex

Specifies an analog channel.

Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>aerAnalogIndex1</td>
<td>0</td>
</tr>
<tr>
<td>aerAnalogIndex2</td>
<td>1</td>
</tr>
<tr>
<td>aerAnalogIndex3</td>
<td>2</td>
</tr>
<tr>
<td>aerAnalogIndex4</td>
<td>3</td>
</tr>
<tr>
<td>aerAnalogIndex5</td>
<td>4</td>
</tr>
<tr>
<td>aerAnalogIndex6</td>
<td>5</td>
</tr>
<tr>
<td>aerAnalogIndex7</td>
<td>6</td>
</tr>
<tr>
<td>aerAnalogIndex8</td>
<td>7</td>
</tr>
<tr>
<td>aerAnalogIndex9</td>
<td>8</td>
</tr>
<tr>
<td>aerAnalogIndex10</td>
<td>9</td>
</tr>
<tr>
<td>aerAnalogIndex11</td>
<td>10</td>
</tr>
<tr>
<td>aerAnalogIndex12</td>
<td>11</td>
</tr>
<tr>
<td>aerAnalogIndex13</td>
<td>12</td>
</tr>
<tr>
<td>aerAnalogIndex14</td>
<td>13</td>
</tr>
<tr>
<td>aerAnalogIndex15</td>
<td>14</td>
</tr>
<tr>
<td>aerAnalogIndex16</td>
<td>15</td>
</tr>
<tr>
<td>aerMaxAnalogs</td>
<td>16</td>
</tr>
</tbody>
</table>

Used by

`ANALOGINDEX` variable
### B.3. aerAnalogMask

These constants specify an analog input channel. To specify multiple channels, add them together (i.e., aerAnalogMask1 + aerAnalogMask2).

**Constants**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>aerAnalogMask1</td>
<td>&amp;H0001</td>
<td>-</td>
</tr>
<tr>
<td>aerAnalogMask2</td>
<td>&amp;H0002</td>
<td>U600</td>
</tr>
<tr>
<td>aerAnalogMask3</td>
<td>&amp;H0004</td>
<td>4EN configured as board #1</td>
</tr>
<tr>
<td>aerAnalogMask4</td>
<td>&amp;H0008</td>
<td>4EN configured as board #2</td>
</tr>
<tr>
<td>aerAnalogMask5</td>
<td>&amp;H0010</td>
<td>4EN configured as board #3</td>
</tr>
<tr>
<td>aerAnalogMask6</td>
<td>&amp;H0020</td>
<td></td>
</tr>
<tr>
<td>aerAnalogMask7</td>
<td>&amp;H0040</td>
<td></td>
</tr>
<tr>
<td>aerAnalogMask8</td>
<td>&amp;H0080</td>
<td></td>
</tr>
<tr>
<td>aerAnalogMask9</td>
<td>&amp;H0100</td>
<td></td>
</tr>
<tr>
<td>aerAnalogMask10</td>
<td>&amp;H0200</td>
<td></td>
</tr>
<tr>
<td>aerAnalogMask11</td>
<td>&amp;H0400</td>
<td></td>
</tr>
<tr>
<td>aerAnalogMask12</td>
<td>&amp;H0800</td>
<td></td>
</tr>
<tr>
<td>aerAnalogMask13</td>
<td>&amp;H1000</td>
<td></td>
</tr>
<tr>
<td>aerAnalogMask14</td>
<td>&amp;H2000</td>
<td></td>
</tr>
<tr>
<td>aerAnalogMask15</td>
<td>&amp;H4000</td>
<td></td>
</tr>
<tr>
<td>aerAnalogMask16</td>
<td>&amp;H4000</td>
<td></td>
</tr>
<tr>
<td>aerAnalogMask</td>
<td>&amp;HFFFF</td>
<td></td>
</tr>
</tbody>
</table>
B.4. aerAutoProgSystem

Specifies either a system file or a user’s file within a program automation.

Constants

- aerAutoProgSystemSysFile 1
- aerAutoProgSystemUserFile 2

Used by

AerAutoProgXXX
B.5. aerAutoProgType

Specifies the type of program to allocate.

**Constants**

- aerAutoProgTypeInclude 0
- aerAutoProgTypeDownloadOnly 1
- aerAutoProgTypeRunSilent 2
- aerAutoProgTypeRun 3
- aerAutoProgTypeRunImmediate 4
- aerAutoProgTypeLoad 5

**Used by**

*AerProgramAllocate*
B.6. aerAxisIndex

Physical axis index numbers.

Constants

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>aerAxisIndex1</td>
<td>0</td>
</tr>
<tr>
<td>aerAxisIndex2</td>
<td>1</td>
</tr>
<tr>
<td>aerAxisIndex3</td>
<td>2</td>
</tr>
<tr>
<td>aerAxisIndex4</td>
<td>3</td>
</tr>
<tr>
<td>aerAxisIndex5</td>
<td>4</td>
</tr>
<tr>
<td>aerAxisIndex6</td>
<td>5</td>
</tr>
<tr>
<td>aerAxisIndex7</td>
<td>6</td>
</tr>
<tr>
<td>aerAxisIndex8</td>
<td>7</td>
</tr>
<tr>
<td>aerAxisIndex9</td>
<td>8</td>
</tr>
<tr>
<td>aerAxisIndex10</td>
<td>9</td>
</tr>
<tr>
<td>aerAxisIndex11</td>
<td>10</td>
</tr>
<tr>
<td>aerAxisIndex12</td>
<td>11</td>
</tr>
<tr>
<td>aerAxisIndex13</td>
<td>12</td>
</tr>
<tr>
<td>aerAxisIndex14</td>
<td>13</td>
</tr>
<tr>
<td>aerAxisIndex15</td>
<td>14</td>
</tr>
<tr>
<td>aerAxisIndex16</td>
<td>15</td>
</tr>
<tr>
<td>aerMaxAxes</td>
<td>16</td>
</tr>
</tbody>
</table>

Used By

AerParmAxisGetValue, …
B.7. aerAxisMask

Specifies multiple axes within one parameter passed to a function (i.e., aerAxisMask1 + aerAxisMask2).

**Constants**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>aerAxisMask1</td>
<td>&amp;H0001</td>
</tr>
<tr>
<td>aerAxisMask2</td>
<td>&amp;H0002</td>
</tr>
<tr>
<td>aerAxisMask3</td>
<td>&amp;H0004</td>
</tr>
<tr>
<td>aerAxisMask4</td>
<td>&amp;H0008</td>
</tr>
<tr>
<td>aerAxisMask5</td>
<td>&amp;H0010</td>
</tr>
<tr>
<td>aerAxisMask6</td>
<td>&amp;H0020</td>
</tr>
<tr>
<td>aerAxisMask7</td>
<td>&amp;H0040</td>
</tr>
<tr>
<td>aerAxisMask8</td>
<td>&amp;H0080</td>
</tr>
<tr>
<td>aerAxisMask9</td>
<td>&amp;H0100</td>
</tr>
<tr>
<td>aerAxisMask10</td>
<td>&amp;H0200</td>
</tr>
<tr>
<td>aerAxisMask11</td>
<td>&amp;H0400</td>
</tr>
<tr>
<td>aerAxisMask12</td>
<td>&amp;H0800</td>
</tr>
<tr>
<td>aerAxisMask13</td>
<td>&amp;H1000</td>
</tr>
<tr>
<td>aerAxisMask14</td>
<td>&amp;H2000</td>
</tr>
<tr>
<td>aerAxisMask15</td>
<td>&amp;H4000</td>
</tr>
<tr>
<td>aerAxisMask16</td>
<td>&amp;H8000</td>
</tr>
<tr>
<td>aerAxisMask</td>
<td>&amp;HFFFF</td>
</tr>
</tbody>
</table>

**Used by**

*AerMoveMulti*
B.8. aerAxisParm

Axis Parameter numbers

Constants

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>aerAxisParmStatus</td>
<td>1</td>
</tr>
<tr>
<td>aerAxisParmPos</td>
<td>2</td>
</tr>
<tr>
<td>aerAxisParmEcho</td>
<td>3</td>
</tr>
<tr>
<td>aerAxisParmClock</td>
<td>4</td>
</tr>
<tr>
<td>aerAxisParmResolver</td>
<td>5</td>
</tr>
<tr>
<td>aerAxisParmRawPos</td>
<td>6</td>
</tr>
<tr>
<td>aerAxisParmPosCmd</td>
<td>7</td>
</tr>
<tr>
<td>aerAxisParmPosErr</td>
<td>8</td>
</tr>
<tr>
<td>aerAxisParmPosToGo</td>
<td>9</td>
</tr>
<tr>
<td>aerAxisParmPosTarget</td>
<td>10</td>
</tr>
<tr>
<td>aerAxisParmIVel</td>
<td>11</td>
</tr>
<tr>
<td>aerAxisParmAvgVel</td>
<td>12</td>
</tr>
<tr>
<td>aerAxisParmAvgVelTime</td>
<td>13</td>
</tr>
<tr>
<td>aerAxisParmKI</td>
<td>16</td>
</tr>
<tr>
<td>aerAxisParmKP</td>
<td>17</td>
</tr>
<tr>
<td>aerAxisParmPGain</td>
<td>18</td>
</tr>
<tr>
<td>aerAxisParmVff</td>
<td>19</td>
</tr>
<tr>
<td>aerAxisParmDrive</td>
<td>20</td>
</tr>
<tr>
<td>aerAxisParmAux</td>
<td>21</td>
</tr>
<tr>
<td>aerAxisParmAffGain</td>
<td>24</td>
</tr>
<tr>
<td>aerAxisParmReversalMode</td>
<td>29</td>
</tr>
<tr>
<td>aerAxisParmReversalValue</td>
<td>30</td>
</tr>
<tr>
<td>aerAxisParmAvg</td>
<td>31</td>
</tr>
<tr>
<td>aerAxisParmIMax</td>
<td>32</td>
</tr>
<tr>
<td>aerAxisParmAvgLimit</td>
<td>33</td>
</tr>
<tr>
<td>aerAxisParmAvgTime</td>
<td>34</td>
</tr>
<tr>
<td>aerAxisParmICmd</td>
<td>35</td>
</tr>
<tr>
<td>aerAxisParmPosErrLimit</td>
<td>36</td>
</tr>
<tr>
<td>aerAxisParmInPosLimit</td>
<td>37</td>
</tr>
<tr>
<td>aerAxisParmCWEot</td>
<td>38</td>
</tr>
<tr>
<td>aerAxisParmCCWIEot</td>
<td>39</td>
</tr>
<tr>
<td>aerAxisParmVelCmdTrap</td>
<td>41</td>
</tr>
<tr>
<td>aerAxisParmVelPosition</td>
<td>42</td>
</tr>
<tr>
<td>aerAxisParmFBWindow</td>
<td>43</td>
</tr>
<tr>
<td>aerAxisParmSafeZoneCW</td>
<td>44</td>
</tr>
<tr>
<td>aerAxisParmSafeZoneCCW</td>
<td>45</td>
</tr>
<tr>
<td>aerAxisParmSafezoneMode</td>
<td>46</td>
</tr>
<tr>
<td>aerAxisParmSimulation</td>
<td>47</td>
</tr>
<tr>
<td>aerAxisParmAccel</td>
<td>48</td>
</tr>
<tr>
<td>aerAxisParmDecel</td>
<td>49</td>
</tr>
<tr>
<td>aerAxisParmAccelMode</td>
<td>50</td>
</tr>
<tr>
<td>aerAxisParmDecelMode</td>
<td>51</td>
</tr>
<tr>
<td>aerAxisParmFeedrateMode</td>
<td>52</td>
</tr>
<tr>
<td>aerAxisParmAccelRate</td>
<td>53</td>
</tr>
<tr>
<td>aerAxisParmDecelRate</td>
<td>54</td>
</tr>
<tr>
<td>aerAxisParmHomeSwitchPos</td>
<td>56</td>
</tr>
<tr>
<td>aerAxisParmHomeSwitchToL</td>
<td>57</td>
</tr>
<tr>
<td>aerAxisParmSystemClock</td>
<td>58</td>
</tr>
<tr>
<td>aerAxisParmFault</td>
<td>64</td>
</tr>
<tr>
<td>aerAxisParmFaultMask</td>
<td>65</td>
</tr>
<tr>
<td>aerAxisParmDisableMask</td>
<td>66</td>
</tr>
<tr>
<td>aerAxisParmIntMask</td>
<td>67</td>
</tr>
</tbody>
</table>
aerAxisParmAuxMask 68
aerAxisParmHaltMask 69
aerAxisParmIOLevel 70
aerAxisParmAuxOffset 71
aerAxisParmAbortMask 72
aerAxisParmCamPosition 78
aerAxisParmCamPoint 79
aerAxisParmMasterPos 80
aerAxisParmMasterRes 81
aerAxisParmMasterLen 82
aerAxisParmCamOffset 84
aerAxisParmSyncSpeed 85
aerAxisParmProfQDepth 87
aerAxisParmProfQSize 88
aerAxisParmMoveQDepth 89
aerAxisParmMoveQSize 90
aerAxisParmOffset 95
aerAxisParmMotionStatus 97
aerAxisParmServoStatus 98
aerAxisParmExtr2dscl 109
aerAxisParmBase_Speed 110
aerAxisParmMax_Phase 111
aerAxisParmPhase_Speed 112
aerAxisParmAlt_Status 113
aerAxisParmSoftLimitMode 115
aerAxisParmHomeVelMult 116
aerAxisParmCamAdvance 117
aerAxisParmVGain 118
aerAxisParmAlpha 119
aerAxisParmVelTimeConst 120
aerAxisParmMaxCamAccel 121
aerAxisParmHomeOffset 122
aerAxisParmCmdPolarity 123
aerAxisParmBrakeMask 124
aerAxisParmDacOffset 125
aerAxisParmB0 126
aerAxisParmB1 127
aerAxisParmB2 128
aerAxisParmA1 129
aerAxisParmA2 130
aerAxisParmPosToGoIRQ 131
aerAxisParmAuxDelay 132
aerAxisParmScalePGain 133
aerAxisParmPosModulo 134
aerAxisParmNegModulo 135
aerAxisParmModuloMode 136
aerAxisParmIMaxPlus 137
aerAxisParmIMaxMinus 138
aerAxisAuxLevel 139
aerAxisParmGearSlave 140
aerAxisParmGearMaster 141
aerAxisParmGearMode 142
aerAxisParmAuxVelCmd 143
aerAxisParmPositionTol 144
aerAxisParmPositionTolTime 145
aerAxisParmGantryMode 146
aerAxisParmGantryOffset 147
aerAxisParmGantryMaster 148
aerAxisParmPhaseAOffset 149
aerAxisParmPhaseBOffset 150
aerAxisParmDataPlotMode 151
aerAxisParmBB0 152
aerAxisParmBB1 153
aerAxisParmBB2 154
aerAxisParmAA1 155
aerAxisParmAA2 156
aerAxisParmMax 157

Used by

\textit{AerParmAxisxxxx}
B.9. aerAxisState

Specifies an axis state.

Constants

<table>
<thead>
<tr>
<th>aerAxisStateFree</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>aerAxisStateCaptured</td>
<td>1</td>
</tr>
<tr>
<td>aerAxisStateBound</td>
<td>2</td>
</tr>
</tbody>
</table>
B.10. aerAxisType

These constants specify the type of axis.

Constants

<table>
<thead>
<tr>
<th>aerAxisTypeLinear</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>aerAxisTypeRotaryModulo</td>
<td>1</td>
</tr>
<tr>
<td>aerAxisTypeRotaryNoModulo</td>
<td>2</td>
</tr>
<tr>
<td>aerAxisTypeMax</td>
<td>3</td>
</tr>
</tbody>
</table>

Used by

Machine parameter – Type
B.11. aerBit

These constants specify a bit in a bit-masked parameter, such as the FAULTMASK axis parameter, aerAxisParmFaultMask.

**Constants**

<table>
<thead>
<tr>
<th>aerBit</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>aerBit1</td>
<td>&amp;H00000001</td>
</tr>
<tr>
<td>aerBit2</td>
<td>&amp;H00000002</td>
</tr>
<tr>
<td>aerBit3</td>
<td>&amp;H00000004</td>
</tr>
<tr>
<td>aerBit4</td>
<td>&amp;H00000008</td>
</tr>
<tr>
<td>aerBit5</td>
<td>&amp;H00000010</td>
</tr>
<tr>
<td>aerBit6</td>
<td>&amp;H00000020</td>
</tr>
<tr>
<td>aerBit7</td>
<td>&amp;H00000040</td>
</tr>
<tr>
<td>aerBit8</td>
<td>&amp;H00000080</td>
</tr>
<tr>
<td>aerBit9</td>
<td>&amp;H00000100</td>
</tr>
<tr>
<td>aerBit10</td>
<td>&amp;H00000200</td>
</tr>
<tr>
<td>aerBit11</td>
<td>&amp;H00000400</td>
</tr>
<tr>
<td>aerBit12</td>
<td>&amp;H00000800</td>
</tr>
<tr>
<td>aerBit13</td>
<td>&amp;H00001000</td>
</tr>
<tr>
<td>aerBit14</td>
<td>&amp;H00002000</td>
</tr>
<tr>
<td>aerBit15</td>
<td>&amp;H00004000</td>
</tr>
<tr>
<td>aerBit16</td>
<td>&amp;H00008000</td>
</tr>
<tr>
<td>aerBit17</td>
<td>&amp;H00010000</td>
</tr>
<tr>
<td>aerBit18</td>
<td>&amp;H00020000</td>
</tr>
<tr>
<td>aerBit19</td>
<td>&amp;H00040000</td>
</tr>
<tr>
<td>aerBit20</td>
<td>&amp;H00080000</td>
</tr>
<tr>
<td>aerBit21</td>
<td>&amp;H00100000</td>
</tr>
<tr>
<td>aerBit22</td>
<td>&amp;H00200000</td>
</tr>
<tr>
<td>aerBit23</td>
<td>&amp;H00400000</td>
</tr>
<tr>
<td>aerBit24</td>
<td>&amp;H00800000</td>
</tr>
<tr>
<td>aerBit25</td>
<td>&amp;H01000000</td>
</tr>
<tr>
<td>aerBit26</td>
<td>&amp;H02000000</td>
</tr>
<tr>
<td>aerBit27</td>
<td>&amp;H04000000</td>
</tr>
<tr>
<td>aerBit28</td>
<td>&amp;H08000000</td>
</tr>
<tr>
<td>aerBit29</td>
<td>&amp;H10000000</td>
</tr>
<tr>
<td>aerBit30</td>
<td>&amp;H20000000</td>
</tr>
<tr>
<td>aerBit31</td>
<td>&amp;H40000000</td>
</tr>
<tr>
<td>aerBit32</td>
<td>&amp;H80000000</td>
</tr>
</tbody>
</table>
B.12. aerBrkPnt

These constants are used for defining breakpoints, on, off and toggling the break point.

Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>aerBrkPntToggle</td>
<td>-1</td>
</tr>
<tr>
<td>aerBrkPntOff</td>
<td>0</td>
</tr>
<tr>
<td>aerBrkPntOn</td>
<td>1</td>
</tr>
</tbody>
</table>

// long data type
B.13. aerCard

These constants are used to register a card with the system and to establish communications with the card during device initialization. The Aerotech device driver can communicate with up to four cards simultaneously.

**Constants**

- aerCardDefault 0
- aerCard1 1
- aerCard2 2
- aerCard3 3
- aerCard4 4

**Used by**

- AerRegxxxx
- AerSysOpen
B.14.  aerCompileError

This constant specifies a compiler error.

Constants

\begin{align*}
\text{aerCompileError} & \quad \& \text{HE0085150} \\
\end{align*}

Used by

\texttt{AerCompilerCompileFile}
B.15. aerCPUStatus

These constants are used for testing the status of the firmware on the controller, Boot Image executing, normal image executing or a fatal error has occurred.

**Constants**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AerCPUStatusBootExec</td>
<td>1</td>
</tr>
<tr>
<td>AerCPUStatusImgExec</td>
<td>2</td>
</tr>
<tr>
<td>AerCPUStatusFatal</td>
<td>3</td>
</tr>
</tbody>
</table>
**B.16. aerDeviceID**

These constants are used to specify to the device driver, the type of Aerotech controller hardware that it will be communicating with. They are used when registering the system in the Win32 registry.

**Constants**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>aerDeviceIDDefault</td>
<td>0</td>
</tr>
<tr>
<td>aerDeviceID600</td>
<td>600</td>
</tr>
<tr>
<td>aerDeviceID631</td>
<td>631</td>
</tr>
</tbody>
</table>

**Used by**

- AerRegxxxx
- AerSysOpen
B.17. aerDrv600Default

These constants are used for defining the default values of the device driver to communicate with the controller. They are AT Window, &hdc00 0000, I/O base address &h220, IRQ 5 and DRAM Size Code, 2 Megabytes.

**Constants**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>aerDrv600DefaultATWin</td>
<td>&amp;hDC00 0000</td>
<td>AT Window &amp;hDC00 0000</td>
</tr>
<tr>
<td>aerDrv600DefaultI0</td>
<td>&amp;h2200</td>
<td>I/O address &amp;h2200</td>
</tr>
<tr>
<td>aerDrv600DefaultIRQ</td>
<td>5</td>
<td>Interrupt 5</td>
</tr>
<tr>
<td>aerDrv600DefaultDSC</td>
<td>2</td>
<td>2 megabytes</td>
</tr>
</tbody>
</table>
B.18. aerErrType

These constants specify the type of warning message (Message Only, Warning Message, Error Message or No Message).

Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>aerErrTypeMsg</td>
<td>0</td>
</tr>
<tr>
<td>aerErrTypeWarn</td>
<td>1</td>
</tr>
<tr>
<td>aerErrTypeError</td>
<td>2</td>
</tr>
<tr>
<td>aerErrTypeNone</td>
<td>3</td>
</tr>
</tbody>
</table>
B.19. aerEvent

These constants specify an event type.

**Constants**

- aerEventUnknownEvent &H8000
- aerEventIRQ2Timer &H8001
- aerEventTaskFault &H8002
- aerEventAxisFault &H8003
- aerEventSerial &H8004
- aerEventVirtIOUpdate &H8005
- aerEventTaskCallback &H8006
- aerEventJoystick &H8007
- aerEventNone &HFFFFFFFF

**Used by**

*AerEventCreateEvent*
B.20. aerFBType

An axis feedback type definition.

Constants

| aerFBTypeNull   | 0 |
| aerFBTypeResolver  | 1 |
| aerFBTypeEncoder    | 2 |
| aerFBTypeEncoderHall | 3 |
| aerFBTypeResolverHall | 6 |

Used by

AerConfigXXXX
B.21. aerFLT

To determine which fault is active.

**Constants**

- `aerFltPositionError` &H00000001 // excess position error
- `aerFltIAverage` &H00000002 // excess RMS current error
- `aerFltCWHardLimit` &H00000004 // CW hardware limit error
- `aerFltCCWHardLimit` &H00000008 // CCW hardware limit error
- `aerFltCWSoftLimit` &H00000010 // CW software limit error
- `aerFltCCWSoftLimit` &H00000020 // CCW software limit error
- `aerFltDrive` &H00000040 // drive fault error
- `aerFltFeedback` &H00000080 // feedback failure error
- `aerFltProgramming` &H00000100 // programming error
- `aerFltMasterFeedback` &H00000200 // master feedback failure
- `aerFltHoming` &H00000400 // homing fault
- `aerFltUser` &H00000800 // user trigger
- `aerFltVelTrap` &H00001000 // velocity trap
- `aerFltVel_CommandTrap` &H00002000 // velocity command trap
- `aerFltHomeTolerance` &H00004000 // home fault tolerance
- `aerFltProbe` &H00008000 // probe input fault
- `aerFltTask` &H00010000 // Task fault
- `aerFltExternalFeedback` &H00020000 // external feedback failure
- `aerFltSafezone` &H00040000 // safe zone fault
- `aerFltConstant` &H00080000 // irq at end accel
- `aerFltDecel` &H01000000 // irq at beginning of decel
- `aerFltDone` &H02000000 //irq at move completion
- `aerFltPosToGo` &H04000000 //irq at position to go
- `aerFltEStop` &H08000000 // ESTOP

**Used by**

*Axis parameter FAULT*
B.22. aerGStripMode

These constants are used for defining the mode of the strip chart.

Constants

- aerGStripModeTime 0
- aerGStripModeTimeSpdOverRide 1
- aerGStripModePos 2
- aerGStripModeQueue 3
- aerGStripModeQueueHold 4
- aerGStripModeEventCount 5
- aerGStripModeEventQueue 6
- aerGStripModeIO 7
- aerGStripModeIOQueue 8
B.23. aerGStripStatus

These constants are used to determine the status of the strip chart.

**Constants**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>aerGStripStatusAllocated</td>
<td>&amp;H0001</td>
</tr>
<tr>
<td>aerGStripStatusArmed</td>
<td>&amp;H0002</td>
</tr>
<tr>
<td>aerGStripStatusTriggered</td>
<td>&amp;H0004</td>
</tr>
<tr>
<td>aerGStripStatusDone</td>
<td>&amp;H0008</td>
</tr>
<tr>
<td>aerGStripStatusOverflow</td>
<td>&amp;H0010</td>
</tr>
<tr>
<td>aerGStripStatusFRMode</td>
<td>&amp;H0020</td>
</tr>
<tr>
<td>aerGStripStatusHold</td>
<td>&amp;H0040</td>
</tr>
<tr>
<td>aerGStripStatusTableMode</td>
<td>&amp;H0080</td>
</tr>
<tr>
<td>aerGStripStatusTrigAllocated</td>
<td>&amp;H0100</td>
</tr>
<tr>
<td>aerGStripStatusQueueMode</td>
<td>&amp;H0200</td>
</tr>
<tr>
<td>aerGStripStatusAborted</td>
<td>&amp;H0400</td>
</tr>
<tr>
<td>aerGStripStatusEventMode</td>
<td>&amp;H0800</td>
</tr>
<tr>
<td>aerGStripStatusUploading</td>
<td>&amp;H1000</td>
</tr>
<tr>
<td>aerGStripStatusLogicTime</td>
<td>&amp;H2000</td>
</tr>
<tr>
<td>aerGStripStatusLogicLevel</td>
<td>&amp;H4000</td>
</tr>
<tr>
<td>aerGStripStatus4khz</td>
<td>&amp;H8000</td>
</tr>
</tbody>
</table>

**Used by**

*AerStripGetStatus*
B.24. aerGStripLogicMode

These constants define the mode of the strip chart.

**Constants**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>aerGStripLogicModeStop</td>
<td>0</td>
</tr>
<tr>
<td>aerGStripLogicModeTime</td>
<td>1</td>
</tr>
<tr>
<td>aerGStripLogicModeLevel</td>
<td>2</td>
</tr>
</tbody>
</table>
B.25. aerGStripLogic

These constants specify the type of virtual I/O data to be collected by the strip chart.

Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>aerGStripLogicBI</td>
<td>0</td>
</tr>
<tr>
<td>aerGStripLogicBO</td>
<td>1</td>
</tr>
<tr>
<td>aerGStripLogicRI</td>
<td>2</td>
</tr>
<tr>
<td>aerGStripLogicRO</td>
<td>3</td>
</tr>
</tbody>
</table>
B.26. aerGlobalParm

Global parameter names.

**Constants**

- aerGlobalParmAvgPollTimeSec 0
- aerGlobalParmVersion 1
- aerGlobalParmNumGlobalDoubles 2
- aerGlobalParmNumGlobalStrings 3
- aerGlobalParmNumGlobalAxisPts 4
- aerGlobalParmEStopEnabled 5
- aerGlobalParmCallBackTimeoutSec 6
- aerGlobalParmInterrupt2TimeSec 7
- aerGlobalParmEnable1KhzServo 8
- aerGlobalParmBuildNumber 9
- aerGlobalParmUserMode 10
- aerGlobalParmThrowTaskWarningsAsFaults 11
- aerGlobalParmSystemStatus 12
- aerGlobalParmEnable2Dcalibration 13
- aerGlobalParmNumCannedFunctions 14
- aerGlobalParmComptatabilityMode 15
- aerGlobalParmNumDecimalsCompare 16
- aerGlobalParmMeasurementMode 17
- aerGlobalParmMax 18

**Used by**

*AerParmGlobalxxxx*
B.27. aerImgExecuting

This constant specifies an error code indicating that the controller is currently running its firmware, most typically when you try to preload its firmware.

**Constants**

| aerImgExecuting | &HE0083009 |

**Used by**

* AerSysDownLoad
B.28. aerIOType

An axis IO type definition.

Constants

| aerIOTypeNull | 0 |
| aerIOTypeD2A | 1 |

Used by

AerConfigXXX
### B.29. aerMachParm

Machine Parameter names.

#### Constants

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>aerMachParmType</td>
<td>0</td>
</tr>
<tr>
<td>aerMachParmCntsPerInch</td>
<td>1</td>
</tr>
<tr>
<td>aerMachParmCntsPerDeg</td>
<td>2</td>
</tr>
<tr>
<td>aerMachParmMaxFeedRateIPM</td>
<td>3</td>
</tr>
<tr>
<td>aerMachParmMaxFeedRateRPM</td>
<td>4</td>
</tr>
<tr>
<td>aerMachParmRapidFeedRateIPM</td>
<td>5</td>
</tr>
<tr>
<td>aerMachParmRapidFeedRateRPM</td>
<td>6</td>
</tr>
<tr>
<td>aerMachParmHomeType</td>
<td>7</td>
</tr>
<tr>
<td>aerMachParmHomeDirection</td>
<td>8</td>
</tr>
<tr>
<td>aerMachParmHomeFeedRateIPM</td>
<td>9</td>
</tr>
<tr>
<td>aerMachParmHomeFeedRateRPM</td>
<td>10</td>
</tr>
<tr>
<td>aerMachParmHomeOffsetInch</td>
<td>11</td>
</tr>
<tr>
<td>aerMachParmHomeOffsetDeg</td>
<td>12</td>
</tr>
<tr>
<td>aerMachParmNumDecimalsEnglish</td>
<td>13</td>
</tr>
<tr>
<td>aerMachParmNumDecimalsMetric</td>
<td>14</td>
</tr>
<tr>
<td>aerMachParmAxisState</td>
<td>15</td>
</tr>
<tr>
<td>aerMachParmControllingTask</td>
<td>16</td>
</tr>
<tr>
<td>aerMachParmPositionUnits</td>
<td>17</td>
</tr>
<tr>
<td>aerMachParmPositionCmdUnits</td>
<td>18</td>
</tr>
<tr>
<td>aerMachParmPresetCmdUnits</td>
<td>19</td>
</tr>
<tr>
<td>aerMachParmAvgVelUnits</td>
<td>20</td>
</tr>
<tr>
<td>aerMachParmFixtureOffset</td>
<td>21</td>
</tr>
<tr>
<td>aerMachParmFixtureOffset1</td>
<td>21</td>
</tr>
<tr>
<td>aerMachParmScaleFactor</td>
<td>22</td>
</tr>
<tr>
<td>aerMachParmPresetUnits</td>
<td>23</td>
</tr>
<tr>
<td>aerMachParmFixtureOffset2</td>
<td>24</td>
</tr>
<tr>
<td>aerMachParmFixtureOffset3</td>
<td>25</td>
</tr>
<tr>
<td>aerMachParmFixtureOffset4</td>
<td>26</td>
</tr>
<tr>
<td>aerMachParmFixtureOffset5</td>
<td>27</td>
</tr>
<tr>
<td>aerMachParmFixtureOffset6</td>
<td>28</td>
</tr>
<tr>
<td>aerMachParmJogDistanceInch</td>
<td>29</td>
</tr>
<tr>
<td>aerMachParmJogDistanceDeg</td>
<td>30</td>
</tr>
<tr>
<td>aerMachParmJogVelocityIPM</td>
<td>31</td>
</tr>
<tr>
<td>aerMachParmJogVelocityRPM</td>
<td>32</td>
</tr>
<tr>
<td>aerMachParmUnusedAxis</td>
<td>33</td>
</tr>
<tr>
<td>aerMachParmReverseSlewDir</td>
<td>34</td>
</tr>
</tbody>
</table>

#### Used by

*AerParmMachine****

// same as aerMachParmFixtureOffset
B.30. aerMax

These constants specify various maximum lengths of data used by the functions.

**Constants**

- aerMaxParmNameLength 32
- aerMaxPath 261
- aerMaxTextLength 255
B.31. aerMaxVirtIO

These constants specify the maximum number of virtual I/O registers and virtual I/O binary bits (as bits, bytes, words and dwords (32 bits)).

**Constants**

- aerMaxVirtIOBits: 512
- aerMaxVirtIOBytes: (aerMaxVirtIOBits/8)
- aerMaxVirtIOWords: (aerMaxVirtIOBits/16)
- aerMaxVirtIODWords: (aerMaxVirtIOBits/32)
- aerMaxVirtIORegisters: 128
B.32. aerMFBType

An axis master feedback type definition.

Constants

- aerMFBTypeNull 0
- aerMFBTypeResolver 1
- aerMFBTypeEncoder 2
- aerMFBTypeVirtual 3

Used by

AerConfigXXXX
B.33. aerMove

These constants specify the type of motion to be generated by the AerMoveAxis function.

**Constants**

- `aerMoveAbsolute` &H80000000
- `aerMoveHome` &H80000001
- `aerMoveIncremental` &H80000002
- `aerMoveFreerun` &H80000003
- `aerMoveInfeedSlave` &H80000004
- `aerMoveQIncremental` &H80000005
- `aerMoveQAbsolute` &H80000006
- `aerMoveHomeQuick` &H80000007
- `aerMoveHomeAltRev` &H80000008
- `aerMoveHomeNoLimit` &H80000009
- `aerMoveOscillate` &H8000000A
- `aerMoveAlignSlave` &H8000000B
- `aerMoveHalt` &H40000010
- `aerMoveAbort` &H40000011
- `aerMoveFeedhold` &H40000012
- `aerMoveRelease` &H40000013
- `aerMoveQFlush` &H40000014
- `aerMoveQHold` &H40000015
- `aerMoveQRelease` &H40000016
- `aerMoveLinear` &H20000000

**Used by**

_AerMoveAxis_
B.34. aerNoErr
A constant specifying no error has occurred.

Constants
aerNoErr 0

Used by
AerCompilerCompileFile
B.35. aerOS

These constants specify the Operating System the application is running under.

Constants

- aerOSWin95 (Win98, also) 1
- aerOSWinNT 2

Used by

AerVerGetOS
## B.36. aerParm

These constants specify the parameter type (Axis, Machine, Global, or Task) and the subgroup type.

### Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>aerParmSubGroupNone</td>
<td>&amp;H0000</td>
</tr>
<tr>
<td>aerParmAxisSubGroupCamming</td>
<td>&amp;H0001</td>
</tr>
<tr>
<td>aerParmAxisSubGroupFaultLevel</td>
<td>&amp;H0002</td>
</tr>
<tr>
<td>aerParmAxisSubGroupFaultMask</td>
<td>&amp;H0003</td>
</tr>
<tr>
<td>aerParmAxisSubGroupMotion</td>
<td>&amp;H0004</td>
</tr>
<tr>
<td>aerParmAxisSubGroupServo</td>
<td>&amp;H0005</td>
</tr>
<tr>
<td>aerParmAxisSubGroupFilter</td>
<td>&amp;H0006</td>
</tr>
<tr>
<td>aerParmAxisSubGroupAdvanced</td>
<td>&amp;H0007</td>
</tr>
<tr>
<td>aerParmAxisSubGroupConfig</td>
<td>&amp;H0008</td>
</tr>
<tr>
<td>aerParmAxisSubGroupMisc</td>
<td>&amp;H0009</td>
</tr>
<tr>
<td>aerParmMachineSubGroupConfig</td>
<td>&amp;H0011</td>
</tr>
<tr>
<td>aerParmMachineSubGroupBasic</td>
<td>&amp;H0012</td>
</tr>
<tr>
<td>aerParmMachineSubGroupDisplay</td>
<td>&amp;H0013</td>
</tr>
<tr>
<td>aerParmMachineSubGroupHoming</td>
<td>&amp;H0014</td>
</tr>
<tr>
<td>aerParmMachineSubGroupOffset</td>
<td>&amp;H0015</td>
</tr>
<tr>
<td>aerParmTaskSubGroupFeedrate</td>
<td>&amp;H0021</td>
</tr>
<tr>
<td>aerParmTaskSubGroupJogging</td>
<td>&amp;H0022</td>
</tr>
<tr>
<td>aerParmTaskSubGroupBasic</td>
<td>&amp;H0023</td>
</tr>
<tr>
<td>aerParmTaskSubGroupCircular</td>
<td>&amp;H0024</td>
</tr>
<tr>
<td>aerParmTaskSubGroupTransform</td>
<td>&amp;H0025</td>
</tr>
<tr>
<td>aerParmTaskSubGroupProgram</td>
<td>&amp;H0026</td>
</tr>
<tr>
<td>aerParmTaskSubGroupSpindle</td>
<td>&amp;H0027</td>
</tr>
<tr>
<td>aerParmTaskSubGroupMisc</td>
<td>&amp;H0028</td>
</tr>
</tbody>
</table>
B.37. aerParmAttr

These values can be binary anded with the attribute mask returned by the AerParmxxxxGetInfo functions, to determine if the conditions listed in the comments below are true or not.

Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>aerParmAttrValid</td>
<td>&amp;H0001</td>
</tr>
<tr>
<td>aerParmAttrRead</td>
<td>&amp;H0002</td>
</tr>
<tr>
<td>aerParmAttrWrite</td>
<td>&amp;H0004</td>
</tr>
<tr>
<td>aerParmAttrUpdate</td>
<td>&amp;H0008</td>
</tr>
<tr>
<td>aerParmAttrUnsigned</td>
<td>&amp;H0010</td>
</tr>
<tr>
<td>aerParmAttrTest</td>
<td>&amp;H0020</td>
</tr>
<tr>
<td>aerParmAttrInteger</td>
<td>&amp;H0040</td>
</tr>
<tr>
<td>aerParmAttrEnglish</td>
<td>&amp;H0100</td>
</tr>
<tr>
<td>aerParmAttrNoLimit</td>
<td>&amp;H0200</td>
</tr>
</tbody>
</table>

Used by

AerParmAxisGetInfo
AerParmMachineGetInfo
AerParmGlobalGetInfo
AerParmTaskGetInfo
B.38. aerParmDisplay

These constants specify the display attributes of the parameters (Axis, Global, Machine, and Task).

**Constants**

- aerParmDisplayAttrValue &H0000
- aerParmDisplayAttrBitmask &H0001
- aerParmDisplayAttrChoice &H0002
- aerParmDisplayAttrHex &H0004
- aerParmDisplayAttrDSPLong &H0010
- aerParmDisplayAttrDSPFixInt &H0020
- aerParmDisplayAttrDSPFixLong &H0040
B.39. aerParmType

These constants specify the type of parameter.

**Constants**

<table>
<thead>
<tr>
<th>aerParmTypeAxis</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>aerParmTypeMachine</td>
<td>1</td>
</tr>
<tr>
<td>aerParmTypeTask</td>
<td>2</td>
</tr>
<tr>
<td>aerParmTypeGlobal</td>
<td>3</td>
</tr>
</tbody>
</table>
B.40. aerPhysAxisIndex

These constants specify physical axis indexes.

**Constants**

aerPhysAxisIndex1 0
aerPhysAxisIndex2 1
aerPhysAxisIndex3 2
aerPhysAxisIndex4 3
aerPhysAxisIndex5 4
aerPhysAxisIndex6 5
aerPhysAxisIndex7 6
aerPhysAxisIndex8 7
aerPhysAxisIndex9 8
aerPhysAxisIndex10 9
aerPhysAxisIndex11 10
aerPhysAxisIndex12 11
aerPhysAxisIndex13 12
aerPhysAxisIndex14 13
aerPhysAxisIndex15 14
aerPhysAxisIndex16 15
B.41. aerProbeStatus

These constants are used for testing the status of the probe.

Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>aerProbeStatusArmed</td>
<td>1</td>
<td>// probe is armed</td>
</tr>
<tr>
<td>aerProbeStatusValidPos</td>
<td>2</td>
<td>// position is valid</td>
</tr>
<tr>
<td>aerProbeStatusInput</td>
<td>4</td>
<td>// has been configured</td>
</tr>
<tr>
<td>aerProbeStatusHighSpeed</td>
<td>8</td>
<td>// using high speed position latch</td>
</tr>
</tbody>
</table>
B.42. aerProgramMsg

These constants specify the status of the CNC program compilation and download process.

**Constants**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>aerProgramMsgNone</td>
<td>-1</td>
</tr>
<tr>
<td>aerProgramMsgCompileStart</td>
<td>0</td>
</tr>
<tr>
<td>aerProgramMsgCompileEnd</td>
<td>1</td>
</tr>
<tr>
<td>aerProgramMsgCompileWarn</td>
<td>2</td>
</tr>
<tr>
<td>aerProgramMsgCompileError</td>
<td>3</td>
</tr>
<tr>
<td>aerProgramMsgCompileInfo</td>
<td>4</td>
</tr>
<tr>
<td>aerProgramMsgDownloadStart</td>
<td>5</td>
</tr>
<tr>
<td>aerProgramMsgDownloadEnd</td>
<td>6</td>
</tr>
<tr>
<td>aerProgramMsgDownloadError</td>
<td>7</td>
</tr>
</tbody>
</table>
B.43. aerPSOChannelMask

These constants are used for defining the channels that the PSO-PC card will track for generating the firing pulse, etc.

**Constants**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>aerPsoChannelMask1</td>
<td>0x01</td>
<td>PSO channel 1</td>
</tr>
<tr>
<td>aerPsoChannelMask2</td>
<td>0x02</td>
<td>PSO channel 2</td>
</tr>
<tr>
<td>aerPsoChannelMask3</td>
<td>0x04</td>
<td>PSO channel 3</td>
</tr>
<tr>
<td>aerPsoChannelMask4</td>
<td>0x08</td>
<td>PSO channel 4</td>
</tr>
</tbody>
</table>
B.44. aerPSOMode

These constants are used for defining the mode of the PSO-PC card.

**Constants**

- aerPsoModeImmediate 0x00 // write output value NOW
- aerPsoModeVelTracking 0x01 // track velocity
- aerPsoModePosTracking 0x02 // track position
B.45. aerPSOTable

These constants are used for defining the mode of the PSO-PC table based commands.

Constants

- aerPSOTableIncDist 0 // incremental distances
- aerPSOTableAbsDist 1 // absolute distances
- aerPSOTableInvalid &HFFFF // invalid firing table type
B.46. aerPtrType

Constants

- aerPtrTypeNull &H00
- aerPtrTypeCSPParmVar &H01
- aerPtrTypeAptGlobalVar &H02
- aerPtrTypeAptTaskVar &H03
- aerPtrTypeAptCSParmVar &H04
- aerPtrTypeStrGlobalVar &H05
- aerPtrTypeStrTaskVar &H06
- aerPtrTypeStrProgramVar &H07
- aerPtrTypeDblGlobalVar &H08
- aerPtrTypeDblTaskVar &H09
- aerPtrTypeDblProgramVar &H0A
- aerPtrTypeDblCSParmVar &H0B
- aerPtrTypeMaskCSParmVar &H0C
- aerPtrTypeDblAptGlobalVar &H0D
- aerPtrTypeMaskAptGlobalVar &H0E
- aerPtrTypeDblAptTaskVar &H0F
- aerPtrTypeMaskAptTaskVar &H10
- aerPtrTypeDblGlobalParm &H11
- aerPtrTypeDblTaskParm &H12
- aerPtrTypeDblMachParm &H13
- aerPtrTypeDWordAxisParm &H14

// Virtual I/O
- aerPtrTypeByteBIVIO &H15 : Binary Input
- aerPtrTypeByteBOVIO &H16 : Binary Input
- aerPtrTypeWordRIVIO &H17 : Register Input
- aerPtrTypeWordROVIO &H18 : Register Input
- aerPtrTypeDblInputAnalog &H19 : Analog Input

Used by

AerVarXXXX
B.47. aerPulseWidth

These constants are used for defining the type of pulse for PSOP CNC command.

Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>aerPulseWidthMsec</td>
<td>0x00</td>
<td>define pulse width in mSec (PSOP 0)</td>
</tr>
<tr>
<td>aerPulseWidthUsec</td>
<td>0x01</td>
<td>define pulse width in uSec (PSOP 4)</td>
</tr>
<tr>
<td>aerPulseDefine</td>
<td>0x02</td>
<td>define 3 phase pulse (PSOP 1)</td>
</tr>
<tr>
<td>aerPulseDefineRamp</td>
<td>0x03</td>
<td>define 3 phase pulse with ramp (PSOP 1)</td>
</tr>
<tr>
<td>aerPulseToggle</td>
<td>0x04</td>
<td>(PSOP 5)</td>
</tr>
</tbody>
</table>
B.48. aerRegID

Constants

- aerRegIDConverted: -1
- aerRegIDAxisCfgFile: 0
- aerRegIDAxisWizFile: 1
- aerRegIDAxisParmFile: 2
- aerRegIDMachParmFile: 3
- aerRegIDTaskParmFile: 4
- aerRegIDGlobParmFile: 5
- aerRegIDManualPageFile: 6
- aerRegIDInstallDir: 7
- aerRegIDProgramDir: 8
- aerRegIDDefaultTeachFile: 9
- aerRegIDPSOImageFile: 10
- aerRegIDImageFile: 11
- aerRegIDBootImageFile: 12
- aerRegIDSymbolicName: 13
- aerRegIDCal2DFile: 14
- aerRegIDAutomationFile: 15
- aerRegIDToolFile: 16
- aerRegIDVirtIOFile: 17
- aerRegIDU600MMIniFile: 18
- aerRegIDU600MMIPosFile: 19
- aerRegIDU600IniFile: 20
- aerRegIDUserErrFile: 21
- aerRegIDManualIO1File: 22
- aerRegIDManualIO2File: 23
- aerRegIDManualIO3File: 24
- aerRegIDManualIO4File: 25
- aerRegIDTaskIni1File: 26
- aerRegIDTaskIni2File: 27
- aerRegIDTaskIni3File: 28
- aerRegIDTaskIni4File: 29
- aerRegIDMax: 30

Used By

- AerRegGetFilename()
- AerRegSetFilename()
B.49. aerReturnType

These constants specify the return type from the ONGOSUB CNC command.

Constants

- aerReturnTypeNull 0
- aerReturnTypeStart 1
- aerReturnTypeInterrupt 2
- aerReturnTypeEnd 3
- aerReturnTypeOffset 4
B.50. aerRIO

These constants specify the Register I/O action.

**Constants**

- aerRIOCycleStart &H0001
- aerRIOCycleStep &H0002
- aerRIOCycleRetraceOn &H0004
- aerRIOCycleRetraceOff &H0008
- aerRIOCycleStop &H0010
- aerRIOCycleReset &H0020
- aerRIOCycleAbort &H0040
- aerRIOAsyncMove &H0080
- aerRIOslewStart &H0100
- aerRIOslewStop &H0200
- aerRIOReserved2 &H0400
- aerRIOReserved3 &H0800
- aerRIOAutoModeOn &H1000
- aerRIOAutoModeOff &H2000
B.51. aerServoStat

To determine the state of an axis.

Constants

- aerServoStatDrive &H00000001 // drive on,off
- aerServoStatAux &H00000002 // auxiliary output on,off
- aerServoStatCWLimit &H00000004 // cw limit switch on,off
- aerServoStatCCWLimit &H00000008 // ccw limit switch on,off
- aerServoStatHome &H00000010 // home switch on,off
- aerServoStatDriveFlt &H00000020 // drive fault status
- aerServoStatAtHome &H00000040 // axis at home position
- aerServoStatDone &H00000080 // motion done
- aerServoStatInPos &H00000100 // axis in position
- aerServoStatFaulted &H00000200 // axis is faulted
- aerServoStatProbeInput &H00000400 // probe input active
- aerServoStatMarker &H00000800 // marker
- aerServoStatHall1 &H00001000 // hallinput 1
- aerServoStatHall2 &H00002000 // hallinput 2
- aerServoStatHall3 &H00004000 // hallinput 3
- aerServoStatHall4 &H00008000 // hallinput 4
- aerServoStatINegLimit &H00010000 // integral negative clamped
- aerServoStatIPosLimit &H00020000 // integral positive clamped
- aerServoStatVFF &H00040000 // VFF Enabled
- aerServoStatBrake &H00080000 // Brake output active
- aerServoStatAlive &H00100000 // axis has been configured
- aerServoStatVVF_0ATC &H00200000 // VFF or position loop zero
- aerServoStatFeedbackIn &H00400000 // Feedback fault input
- aerServoStatMFeedbackIn &H00800000 // Mst Feedback fault input
- aerServoStatHPVMELaser &H01000000 // HPVME Laser
- aerServoStatScalePGain &H02000000 // SCALEPGAIN axis parameter
- aerServoStatAC &H04000000 // AC motor selected
- aerServoStatMSET &H08000000 // Axis in MSET mode
- aerServoStatHomed &H10000000 // Axis has been homed
- aerServoStatEncoder &H20000000 // Axis has encoder feedback
- aerServoStatErrorMap &H40000000 // Error mapping enabled
- aerServoStatPLoopOnly &H80000000 // position loop only

Used by

Axis parameter SERVOSTATUS
### B.52. aerStat

To determine the state of the axis processor.

**Constants**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>aerStatDrive</td>
<td>&amp;H00000001</td>
<td>drive on, off</td>
</tr>
<tr>
<td>aerStatAux</td>
<td>&amp;H00000002</td>
<td>auxiliary output on, off</td>
</tr>
<tr>
<td>aerStatCWLimit</td>
<td>&amp;H00000004</td>
<td>cw limit switch on, off</td>
</tr>
<tr>
<td>aerStatCCWLimit</td>
<td>&amp;H00000008</td>
<td>ccw limit switch on, off</td>
</tr>
<tr>
<td>aerStatHome</td>
<td>&amp;H00000010</td>
<td>home switch on, off</td>
</tr>
<tr>
<td>aerStatDriveFlt</td>
<td>&amp;H00000020</td>
<td>drive fault status</td>
</tr>
<tr>
<td>aerStatAtHome</td>
<td>&amp;H00000040</td>
<td>axis at home position</td>
</tr>
<tr>
<td>aerStatDone</td>
<td>&amp;H00000080</td>
<td>motion done</td>
</tr>
<tr>
<td>aerStatInPos</td>
<td>&amp;H00000100</td>
<td>axis in position</td>
</tr>
<tr>
<td>aerStatFaulted</td>
<td>&amp;H00000200</td>
<td>axis is faulted</td>
</tr>
<tr>
<td>aerStatProbeInput</td>
<td>&amp;H00000400</td>
<td>probe input active</td>
</tr>
<tr>
<td>aerStatMarker</td>
<td>&amp;H00000800</td>
<td>marker</td>
</tr>
<tr>
<td>aerStatHall1</td>
<td>&amp;H00001000</td>
<td>hall input 1</td>
</tr>
<tr>
<td>aerStatHall2</td>
<td>&amp;H00002000</td>
<td>hall input 2</td>
</tr>
<tr>
<td>aerStatHall3</td>
<td>&amp;H00004000</td>
<td>hall input 3</td>
</tr>
<tr>
<td>aerStatHall4</td>
<td>&amp;H00008000</td>
<td>hall input 4</td>
</tr>
<tr>
<td>aerStatMoveDir</td>
<td>&amp;H00010000</td>
<td>move direction</td>
</tr>
<tr>
<td>aerStatMoving</td>
<td>&amp;H00020000</td>
<td>moving</td>
</tr>
<tr>
<td>aerStatAccel</td>
<td>&amp;H00040000</td>
<td>axis in accel phase</td>
</tr>
<tr>
<td>aerStatDecel</td>
<td>&amp;H00080000</td>
<td>axis in decel phase</td>
</tr>
<tr>
<td>aerStatHoming</td>
<td>&amp;H00100000</td>
<td>axis homing</td>
</tr>
<tr>
<td>aerStatFeedOver</td>
<td>&amp;H00200000</td>
<td>feed rate override</td>
</tr>
<tr>
<td>aerStatProfile</td>
<td>&amp;H00400000</td>
<td>axis in profile mode</td>
</tr>
<tr>
<td>aerStatSync</td>
<td>&amp;H00800000</td>
<td>axis in sync mode</td>
</tr>
<tr>
<td>aerStatCamTable</td>
<td>&amp;H01000000</td>
<td>cam table enabled</td>
</tr>
<tr>
<td>aerStatHomeDir</td>
<td>&amp;H02000000</td>
<td>home direction</td>
</tr>
<tr>
<td>aerStatContMove</td>
<td>&amp;H04000000</td>
<td>continuous move</td>
</tr>
<tr>
<td>aerStatQueue</td>
<td>&amp;H08000000</td>
<td>motion queue active</td>
</tr>
<tr>
<td>aerStatHold</td>
<td>&amp;H10000000</td>
<td>hold active</td>
</tr>
<tr>
<td>aerStatAuxMode</td>
<td>&amp;H20000000</td>
<td>aux mode</td>
</tr>
<tr>
<td>aerStatBlockMotion</td>
<td>&amp;H40000000</td>
<td>block motion</td>
</tr>
<tr>
<td>aerStatHoldQueue</td>
<td>&amp;H80000000</td>
<td>hold queue</td>
</tr>
</tbody>
</table>

**Used by**

*Axis parameter STATUS*
B.53. aerStripIOPosLatch

These constants are used to test the state of the I/O position latch.

**Constants**

- aerStripIOPosLatchDisable 0
- aerStripIOPosLatchOneShot 1
- aerStripIOPosLatchCont 2
- aerStripIOPosLatchAxisIO 0
- aerStripIOPosLatchVirtIO 1
- aerStripIOPosLatchBitCCW 0
- aerStripIOPosLatchBitCW 1
- aerStripIOPosLatchHome 2
- aerStripIOPosLatchEncFlt 3
- aerStripIOPosLatchFault 4
- aerStripIOPosLatchHallA 5
- aerStripIOPosLatchHallB 6
- aerStripIOPosLatchHallC 7
B.54. aerStatus

These constants specify status words.

**Constants**

- aerStatusFault &H0001
- aerStatusAxisWord1 &H0002
- aerStatusAxisWord2 &H0003
B.55. aerSP1Type

A spare axis feedback type definition.

**Constants**

- aerSp1TypeNull: 0
- aerSp1TypeEncoder: 3
- aerSp1TypeEncoderSlave: 4
- aerSp1TypeResolver: 5

**Used by**

AerConfigXXXX
B.56. aerSP2Type

A spare feedback type definition, defined as 0.

Constants

| aerSp2TypeNull         | 0 |

Used by

AerConfigXXXX
B.57. aerTaskAxisIndex

These constants are used to specify a task axis index.

**Constants**

- aerTaskAxisIndex1: 0
- aerTaskAxisIndex2: 1
- aerTaskAxisIndex3: 2
- aerTaskAxisIndex4: 3
- aerTaskAxisIndex5: 4
- aerTaskAxisIndex6: 5
- aerTaskAxisIndex7: 6
- aerTaskAxisIndex8: 7
- aerTaskAxisIndex9: 8
- aerTaskAxisIndex10: 9
- aerTaskAxisIndex11: 10
- aerTaskAxisIndex12: 11
- aerTaskAxisIndex13: 12
- aerTaskAxisIndex14: 13
- aerTaskAxisIndex15: 14
- aerTaskAxisIndex16: 15
B.58. aerTaskExec

Specifies the type of program execution to begin.

Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>aerTaskExecDefault</td>
<td>-1</td>
</tr>
<tr>
<td>aerTaskExecRun</td>
<td>0</td>
</tr>
<tr>
<td>aerTaskExecRunInto</td>
<td>0</td>
</tr>
<tr>
<td>aerTaskExecStepInto</td>
<td>1</td>
</tr>
<tr>
<td>aerTaskExecStepOver</td>
<td>2</td>
</tr>
<tr>
<td>aerTaskExecRunOver</td>
<td>3</td>
</tr>
</tbody>
</table>

Used by

-AerTaskProgramExecute-
B.59. aerTaskIndex

Task index numbers.

Constants

- aerTaskIndex1 0
- aerTaskIndex2 1
- aerTaskIndex3 2
- aerTaskIndex4 3
- aerMaxTasks 4

Used by

TASKINDEX variables
B.60. `aerTaskMask`

These constants allow multiple tasks to be specified as one parameter to a function (i.e., `aerTaskMask1 + aerTaskMask2`).

**Constants**

- `aerTaskMask1` &H0001
- `aerTaskMask2` &H0002
- `aerTaskMask3` &H0004
- `aerTaskMask4` &H0008
- `aerTaskMask` &H000F

**Used by**

- `AerSysInitSystem`
- `AerAutoProgXXXX`
- `AerCompilerAutoIncludeEx`
- `AerDCGetTaskDirect`
- `AerParmTaskDownloadFile`
- `AerSysFaultAck`
B.61. aerTaskMode

To determine which task mode is active. Parentheses indicate the interpretation of the mode when it is not active.

**Constants**

```c
aerTaskModeWord1 = 0
aerTaskMode1BitEnglish = &H00000001 // G70/G71 !Metric
aerTaskMode1BitAbsolute = &H00000002 // !Incremental
aerTaskMode1BitAccelModeLinear = &H00000004 // !1-cosine
aerTaskMode1BitAccelModeRate = &H00000008 // !Time based
aerTaskMode1BitRotaryDominant = &H00000010 // !Linear
aerTaskMode1BitMotionContinuous = &H00000020 // !Decel To Zero
aerTaskMode1BitInverseCircular = &H00000040
aerTaskMode1BitSpindleShutDown = &H00000080
aerTaskMode1BitBlockDelete = &H00000100
aerTaskMode1BitOptionalStop = &H00000200
aerTaskMode1BitBreakPoint = &H00000400
aerTaskMode1BitMFOLock = &H00000800
aerTaskMode1BitMSOLock = &H00001000
aerTaskMode1BitDryRunFeedRate = &H00002000
aerTaskMode1BitRetrace = &H00004000
aerTaskMode1BitAutoMode = &H00008000
aerTaskMode1BitProgramFeedRateMPU = &H00010000
aerTaskMode1BitProgramFeedRateUPR = &H00020000
aerTaskMode1BitProgramSFeedRateSurf1 = &H00040000
aerTaskMode1BitProgramSFeedRateSurf2 = &H00080000
aerTaskMode1BitProgramSFeedRateSurf3 = &H00100000
aerTaskMode1BitProgramSFeedRateSurf4 = &H00200000
aerTaskMode1BitBlockDelete2 = &H00400000
aerTaskMode1BitRunOverMode = &H00800000
aerTaskMode1BitMultiBlockLookAhead = &H01000000
aerTaskMode1BitMachineLock = &H02000000
aerTaskMode1BitHighSpeedBuffering = &H04000000
```

**Used by**

- AerTaskGetModeName
- AER_TASK_MODE
B.62. aerTaskParm

Task Parameter names.

Constants

aerTaskParmNumber 0
aerTaskParmTaskFault 1
aerTaskParmTaskWarning 2
aerTaskParmMaxCallStack 3
aerTaskParmMaxModeStack 4
aerTaskParmNumTaskDoubles 5
aerTaskParmNumTaskStrings 6
aerTaskParmNumTaskAxisPts 7
aerTaskParmESTopInput 8
aerTaskParmFeedHoldInput 9
aerTaskParmFeedHoldEdgeInput 10
aerTaskParmS1_Index 11
aerTaskParmS1_RPM 12
aerTaskParmS2_Index 13
aerTaskParmS2_RPM 14
aerTaskParmS3_Index 15
aerTaskParmS3_RPM 16
aerTaskParmS4_Index 17
aerTaskParmS4_RPM 18
aerTaskParmRotateX 19
aerTaskParmRotateY 20
aerTaskParmRotateAngleDeg 21
aerTaskParmRThetaX 22
aerTaskParmRThetaY 23
aerTaskParmRThetaR 24
aerTaskParmRThetaT 25
aerTaskParmRThetaRadiusInch 26
aerTaskParmRThetaEnabled 27
aerTaskParmUpdateTimeSec 28
aerTaskParmAccelTimeSec 29
aerTaskParmDecelTimeSec 30
aerTaskParmAccelRateIPS2 31
aerTaskParmDecelRateIPS2 32
aerTaskParmAccelRateDPS2 33
aerTaskParmDecelRateDPS2 34
aerTaskParmLinearFeedRate 35
aerTaskParmRotaryFeedRate 36
aerTaskParmMFO 37
aerTaskParmMSO 38
aerTaskParmCoord1I 39
aerTaskParmCoord1J 40
aerTaskParmCoord1K 41
aerTaskParmCoord2I 42
aerTaskParmCoord2J 43
aerTaskParmCoord2K 44
aerTaskParmCoord2Plane 45
aerTaskParmCutterX 46
aerTaskParmCutterY 47
aerTaskParmCutterRadiusInch 48
aerTaskParmNormalcyX 49
aerTaskParmNormalcyY
aerTaskParmNormalcyAxis
aerTaskParmUserFeedRateMode
aerTaskParmMaxMonitorData
aerTaskParmMaxOnGosubData
aerTaskParmAnalogMFOInput
aerTaskParmFeedHold
aerTaskParmMaxRadiusError
aerTaskParmStatus1
aerTaskParmStatus2
aerTaskParmStatus3
aerTaskParmMode1
aerTaskParmAnalogMSOInput
aerTaskParmErrCode
aerTaskParmGlobalEStopDisable
aerTaskParmLinearFeedRateActual
aerTaskParmRotaryFeedRateActual
HaltTaskOnAxisFault
aerTaskParmInterrupt
aerTaskParmInterruptReturnType
aerTaskParmS2_AnalogMSOInput
aerTaskParmS3_AnalogMSOInput
aerTaskParmS4_AnalogMSOInput
aerTaskParmSlewPair1
aerTaskParmSlewPair2
aerTaskParmSlewPair3
aerTaskParmSlewPair4
aerTaskParmSlewPair5
aerTaskParmSlewPair6
aerTaskParmSlewPair7
aerTaskParmSlewPair8
aerTaskParmRIActionOpCode
aerTaskParmRIActionAxis
aerTaskParmRIActionParm1
aerTaskParmRIActionParm2
aerTaskParmSlewPair1
aerTaskParmSlewPair2
aerTaskParmSlewPair3
aerTaskParmSlewPair7
aerTaskParmSlewPair8
aerTaskParmSlewPair9
aerTaskParmSlewPair10
aerTaskParmSlewPair11
aerTaskParmSlewPair12
aerTaskParmSlewPair13
aerTaskParmSlewPair14
aerTaskParmSlewPair15
aerTaskParmSlewPair16
aerTaskParmSlewPair17
aerTaskParmSlewPair18
aerTaskParmSlewPair19
aerTaskParmSlewPair20
aerTaskParmSlewPair21
aerTaskParmSlewPair22
aerTaskParmSlewPair23
aerTaskParmSlewPair24
aerTaskParmSlewPair25
aerTaskParmSlewPair26
aerTaskParmSlewPair27
aerTaskParmSlewPair28
aerTaskParmSlewPair29
aerTaskParmSlewPair30
aerTaskParmSlewPair31
aerTaskParmSlewPair32
aerTaskParmSlewPair33
aerTaskParmSlewPair34
aerTaskParmSlewPair35
aerTaskParmSlewPair36
aerTaskParmSlewPair37
aerTaskParmSlewPair38
aerTaskParmSlewPair39
aerTaskParmSlewPair40
aerTaskParmSlewPair41
aerTaskParmSlewPair42
aerTaskParmSlewPair43
aerTaskParmSlewPair44
aerTaskParmSlewPair45
aerTaskParmSlewPair46
aerTaskParmSlewPair47
aerTaskParmSlewPair48
aerTaskParmSlewPair49
aerTaskParmSlewPair50
aerTaskParmSlewPair51
aerTaskParmSlewPair52
aerTaskParmSlewPair53
aerTaskParmSlewPair54
aerTaskParmSlewPair55
aerTaskParmSlewPair56
aerTaskParmSlewPair57
aerTaskParmSlewPair58
aerTaskParmSlewPair59
aerTaskParmSlewPair60
aerTaskParmSlewPair61
aerTaskParmSlewPair62
aerTaskParmSlewPair63
aerTaskParmSlewPair64
aerTaskParmSlewPair65
aerTaskParmSlewPair66
aerTaskParmSlewPair67
aerTaskParmSlewPair68
aerTaskParmSlewPair69
aerTaskParmSlewPair70
aerTaskParmSlewPair71
aerTaskParmSlewPair72
aerTaskParmSlewPair73
aerTaskParmSlewPair74
aerTaskParmSlewPair75
aerTaskParmSlewPair76
aerTaskParmSlewPair77
aerTaskParmSlewPair78
aerTaskParmSlewPair79
aerTaskParmSlewPair80
aerTaskParmSlewPair81
aerTaskParmSlewPair82
aerTaskParmSlewPair83
aerTaskParmSlewPair84
aerTaskParmSlewPair85
aerTaskParmSlewPair86
aerTaskParmSlewPair87
aerTaskParmSlewPair88
aerTaskParmSlewPair89
aerTaskParmSlewPair90
aerTaskParmSlewPair91
aerTaskParmSlewPair92
aerTaskParmSlewPair93
aerTaskParmSlewPair94
aerTaskParmSlewPair95
aerTaskParmSlewPair96
aerTaskParmSlewPair97
aerTaskParmSlewPair98
aerTaskParmSlewPair99
aerTaskParmReserved
aerTaskParmActiveFixtureOffset
aerTaskParmExecuteNumLines
aerTaskParmJogPair1EnableIn
aerTaskParmJogPair1Mode
aerTaskParmJogPair1Axis1
aerTaskParmJogPair1Axis1PlusIn
aerTaskParmJogPair1Axis1MinusIn
aerTaskParmJogPair1Axis2 108
aerTaskParmJogPair1Axis2PlusIn 109
aerTaskParmJogPair1Axis2MinusIn 110
aerTaskParmJogPair2EnableIn 111
aerTaskParmJogPair2Mode 112
aerTaskParmJogPair2Axis1 113
aerTaskParmJogPair2Axis1PlusIn 114
aerTaskParmJogPair2Axis1MinusIn 115
aerTaskParmJogPair2Axis2 116
aerTaskParmJogPair2Axis2PlusIn 117
aerTaskParmJogPair2Axis2MinusIn 118
aerTaskParmDryRunLinearFeedRateIPM 119
aerTaskParmDryRunRotaryFeedRateRPM 120
aerTaskParmMaxLookAheadMoves 121
aerTaskParmLineNumberUser 122
aerTaskParmLineNumber960 123
aerTaskParmCannedFunctionID 124
aerTaskParmDecelOnProgramAbortMask 125
aerTaskParmIgnoreAxesMask 126
aerTaskParmChordicalToleranceInch 127
aerTaskParmROReq1Mask 128
aerTaskParmNormalcyToleranceDeg 129
aerTaskParmChordicalSlowdownMsec 130
aerTaskParmCommandVelocityVariance 131
aerTaskParmCutterToleranceDeg 132
aerTaskParmCutterZ 133
aerTaskParmCutterLength 134
aerTaskParmCutterWear 135
aerTaskParmCutterOffsetX 136
aerTaskParmCutterOffsetY 137
aerTaskParmCutterRadius 138
aerTaskParmCutterActive 139
aerTaskParmRThetaRadius 140
aerTaskParmAccelRate 141
aerTaskParmDecelRate 142
aerTaskParmMaxRadiusAdjust 143
aerTaskParmUpdateNumEntries 144
aerTaskParmMaxCornerRoundErr 145
aerTaskParmGroup1GCodeMode 146
aerTaskParmExecuteNumMonitors 147
aerTaskParmExecuteNumSpindles 148
aerTaskParmProfileAxesZeroVel 149
aerTaskParmProfileAxesInpos 150
aerTaskParmMax 151

Used by

AerParmTaskxxxx
B.63. aerTaskStatus

To determine which task status is active. TaskStatus, which is a result of a G-code, is identified with the appropriate status bit.

Constants

<table>
<thead>
<tr>
<th>Status Word</th>
<th>Status Bit Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>aerTaskStatusWord1</td>
<td>0 aerTaskStatus1BitProgramAssociated</td>
<td>&amp;H00000001</td>
</tr>
<tr>
<td>aerTaskStatusWord2</td>
<td>1 aerTaskStatus1BitProgramActive</td>
<td>&amp;H00000002</td>
</tr>
<tr>
<td>aerTaskStatusWord3</td>
<td>2 aerTaskStatus1BitImmediateCodeExecuting</td>
<td>&amp;H00000008</td>
</tr>
<tr>
<td>aerTaskStatus1BitAborted</td>
<td>&amp;H00000020</td>
<td>aerTaskStatus1BitProgramCleanup</td>
</tr>
<tr>
<td>aerTaskStatus1BitInterruptFaultPending</td>
<td>&amp;H00000040</td>
<td>aerTaskStatus1BitMotionTypeCW</td>
</tr>
<tr>
<td>aerTaskStatus1BitMotionTypeCCW</td>
<td>&amp;H00100000</td>
<td>aerTaskStatus1BitNormalcyActiveLeft</td>
</tr>
<tr>
<td>aerTaskStatus1BitNormalcyActiveRight</td>
<td>&amp;H00400000</td>
<td>aerTaskStatus1BitMotionType1Plane1Active</td>
</tr>
<tr>
<td>aerTaskStatus1BitMotionType2Plane1Active</td>
<td>&amp;H01000000</td>
<td>aerTaskStatus1BitMotionType3Plane1Active</td>
</tr>
<tr>
<td>aerTaskStatus1BitMotionType4Plane1Active</td>
<td>&amp;H04000000</td>
<td>aerTaskStatus1BitMotionNoAccel</td>
</tr>
</tbody>
</table>

Used by

AerTaskGetStatusName
AER_TASK_STATUS
B.64. aerTool

These constants are used for Tool Tables.

Constants

- aerToolEnglish \&H00000001
- aerToolInUse \&H00000002
- aerToolForceUnits \&H00000004
- aerToolUserDiameter \&H00008000
- aerToolValid \&H80000000

Used by

_AerToolXXX_
B.65. aerVersion

These constants specify the controllers software version.

Constants

aerVersionMajor  |
- Indicates current version number

aerVersionMinor  |
| i.e., 6.0.132

aerVersionBuild  |
### B.66. Q

These are miscellaneous constants.

<table>
<thead>
<tr>
<th>Constants</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>QConstant</td>
<td>1</td>
</tr>
<tr>
<td>QBinaryIn</td>
<td>2</td>
</tr>
<tr>
<td>QBinaryOut</td>
<td>3</td>
</tr>
<tr>
<td>QRegisterIn</td>
<td>4</td>
</tr>
<tr>
<td>QRegisterOut</td>
<td>5</td>
</tr>
<tr>
<td>QLongIn</td>
<td>5</td>
</tr>
<tr>
<td>QLongOut</td>
<td>6</td>
</tr>
<tr>
<td>QServoStatus</td>
<td>98</td>
</tr>
<tr>
<td>QMotionStatus</td>
<td>97</td>
</tr>
<tr>
<td>QEQ</td>
<td>0</td>
</tr>
<tr>
<td>QLT</td>
<td>1</td>
</tr>
<tr>
<td>QLE</td>
<td>2</td>
</tr>
<tr>
<td>QGT</td>
<td>3</td>
</tr>
<tr>
<td>QGE</td>
<td>4</td>
</tr>
<tr>
<td>QNE</td>
<td>5</td>
</tr>
<tr>
<td>QOR</td>
<td>6</td>
</tr>
<tr>
<td>QAND</td>
<td>7</td>
</tr>
<tr>
<td>QANDNOT</td>
<td>8</td>
</tr>
<tr>
<td>QADD</td>
<td>9</td>
</tr>
<tr>
<td>QSUB</td>
<td>10</td>
</tr>
<tr>
<td>QMULT</td>
<td>11</td>
</tr>
<tr>
<td>QDIV</td>
<td>12</td>
</tr>
<tr>
<td>QASSIGN</td>
<td>13</td>
</tr>
<tr>
<td>QXOR</td>
<td>14</td>
</tr>
<tr>
<td>QXNOR</td>
<td>15</td>
</tr>
<tr>
<td>QMoveAbs</td>
<td>0</td>
</tr>
<tr>
<td>QMoveIndex</td>
<td>1</td>
</tr>
<tr>
<td>QMoveStart</td>
<td>2</td>
</tr>
<tr>
<td>QMoveHalt</td>
<td>3</td>
</tr>
<tr>
<td>QStart</td>
<td>1</td>
</tr>
<tr>
<td>QHold</td>
<td>0</td>
</tr>
<tr>
<td>QReset</td>
<td>2</td>
</tr>
<tr>
<td>QStep</td>
<td>3</td>
</tr>
<tr>
<td>QAuto</td>
<td>4</td>
</tr>
</tbody>
</table>
APPENDIX C: STRUCTURES AND DATA TYPES (C LANGUAGE)

C.1. Description

The following section provides a definition of all the structures used within the library (AerSys.DLL).

Any of the following structures can be proceeded by a “P” to refer to a pointer to the structure (i.e., SPINDLEMASK becomes a pointer to the structure as “PSPINDLEMASK”)
C.2. AER_AUX_POINT

Structure
typedef struct tagAER_AUX_POINT
{
  LONG lMaster;   // Master coordinate (in counts)
  WORD wLevel;    // Aux level to set (from lMaster to next
                  // point’s lMaster)
} AER_AUX_POINT;
typedef AER_AUX_POINT *PAER_AUX_POINT;

Members
See inline comments in the structure listed above.

See
AerAuxTablexxx
C.3. **AER_AXISCAL_PACKET**

**Structure**
```c
typedef struct tagAER_AXISCAL_PACKET
{
    WORD    wTable; // table number.
    DWORD   dwInput; // axis number whose positions produce the correction.
    DWORD   dwScale; // scale for the correction value.
} AER_AXISCAL_PACKET;
```

typedef AER_AXISCAL_PACKET *PAER_AXISCAL_PACKET;

**Members**
See inline comments in the structure listed above.

**See**

*AerAxisCalGetPacket*
C.4. AER_AXISCAL_STATUS_PACKET

Structure

typedef struct tagAER_AXISCAL_STATUS_PACKET
{
    WORD wSize; // size of axis calibration table
    WORD wStatus; // 1/0 = allocated/not allocated
} AER_AXISCAL_STATUS_PACKET;
typedef AER_AXISCAL_STATUS_PACKET *PAER_AXISCAL_STATUS_PACKET;

Members

See inline comments in the structure listed above.

See

AerAxisCalGetStatusPacket
C.5. **AER_AXIS_DATA_EX**

**Structure**

```c
typedef struct tagAER_AXIS_DATA_EX
{
    LONG lPos;     // counts - Reflects Axis Parm - POS
    LONG lPosCmd;  // counts - Reflects Axis Parm - POSCMD
    LONG lAvgVel;  // cnts/sec - Reflects Axis Parm - AVGVEL
    LONG lRawPos;  // counts - Reflects Axis Parm - RAWPOS
    LONG lTarget;  // counts - Reflects Axis Parm - TARGET
    DOUBLE dPreset; // units,deg
    DOUBLE dFixtureOffset; // units,deg
    DOUBLE dPosFactor; // Position conversion factor -> counts To User Units
    DOUBLE dAvgVel;    // AvgVel User Units
    DWORD dwFaultStatus;  // Reflects Axis parameter - FAULT
    DWORD dwServoStatus;  // Reflects Axis parameter - SERVOSTATUS
    DWORD dwMotionStatus; // Reflects Axis parameter - MOTIONSTATUS
    DWORD dwType;         // AXISTYPE_
    WORD wNumDecimals;   // Number of Decimals
    WORD wEnglish;       // Whether your in English or Metric
} AER_AXIS_DATA_EX;
typedef AER_AXIS_DATA_EX *PAER_AXIS_DATA_EX;
```

The position information is in counts. To convert to user units - mm or inch:

\[
dPosUserUnits = lPos \times dPosFactor;
\]

The "PositionFactor" takes into account the current mode (English/Metric) and the CntsPerInch machine parameter.

**Members**

See inline comments in the structure listed above.

**See**

Data Center Functions
C.6. AER_CAM_GETPOINT

Structure

typedef struct tagAER_CAM_GETPOINT
{
    LONG lMaster;  // Master position (in counts)
    LONG lSlave;   // Slave  position (in counts)
    LONG lCoeffA; // A coefficient
    LONG lCoeffB; // B coefficient
    LONG lCoeffC; // C coefficient
    WORD wType;   // Type of interpolation (AERCAM_POINT_XXX
    // constant)
} AER_CAM_GETPOINT;
typedef AER_CAM_GETPOINT *PAER_CAM_GETPOINT;

Members

See inline comments in the structure listed above.

See

AerCamTableGetPoint
C.7. AER_CAM_SETPOINT

Structure

typedef struct tagAER_CAM_SETPOINT
{
    LONG lMaster;   // Master position (in counts)
    LONG lSlave;    // Slave position (in counts)
    WORD wType;     // Type of interpolation (AERCAM_POINT_XXX
                   // constant)
} AER_CAM_SETPOINT;

typedef AER_CAM_SETPOINT   *PAER_CAM_SETPOINT;

Members

See inline comments in the structure listed above.

See

AerCamTableSetPoint
C.8. AER_CFG_FBPACKET_ENCODER

Structure

// Use if wFBType == AER_FBTYPE_ENCODER
typedef struct tagAER_CFG_FBPACKET_ENCODER
{
    WORD wChannel;  // Position feedback channel 1 to 16
    DWORD dwLines;  // Lines per motor revolution/after x4
    // multiplication
    WORD wBounded;  // 1 to activate software limits, 0 to
    // disable them
} AER_CFG_FBPACKET_ENCODER;

typedef AER_CFG_FBPACKET_ENCODER *PAER_CFG_FBPACKET_ENCODER;

Members

See inline comments in the structure listed above.

See

AerConfig
C.9. AER_CFG_FBPACKET_ENCODER_HALL

Structure

// Use if wFBType == AER_FBTYPETER_ENCODER_HALL  3
typedef struct tagAER_CFG_FBPACKET_ENCODER_HALL
{
    WORD  wChannel;  // Position feedback channel 1 to 16
    DWORD dwLines;   // Lines per motor revolution/after x4
                     // multiplication
    DWORD dwHallLines; // Lines per elec. cycle after x4
                       // multiplication
    WORD  wCommOffset;  // Commutation offset (Hall effect)
    WORD  wCommChannel; // Feedback channel, 1 to 16 for commutation
    WORD  wBounded;    // 1 to activate software limits, 0 to
                       // disable them
} AER_CFG_FBPACKET_ENCODER_HALL;
typedef AER_CFG_FBPACKET_ENCODER_HALL
*PAER_CFG_FBPACKET_ENCODER_HALL;

Members

See inline comments in the structure listed above.

See

AerConfigHallEffect
C.10. AER_CFG_FBPACKET_HPVME

Structure

```c
// Use if wFBType == AER_FBTYPE_HPVME
typedef struct tagAER_CFG_FBPACKET_HPVME
{
    WORD  wAddress;    // Upper 16 bits of address
    WORD  wLSCR;       // Laser source control register
    DWORD dwActualLines;    // Number of actual lines
    DWORD dwEffectiveLines;  // Number of effective lines
} AER_CFG_FBPACKET_HPVME;
```

typedef AER_CFG_FBPACKET_HPVME *PAER_CFG_FBPACKET_HPVME;

Members

See inline comments in the structure listed above.

See

AerConfig
C.11.  AER_CFG_FBPACKET_NULL

Structure
   // Use if wFBType == AER_FBTYPE_NULL
   typedef struct tagAER_CFG_FBPACKET_NULL
   {
      DWORD dwDummyLines;
   } AER_CFG_FBPACKET_NULL;
   typedef AER_CFG_FBPACKET_NULL *PAER_CFG_FBPACKET_NULL;

Members
   See inline comments in the structure listed above.

See
   AerConfig
C.12. AER_CFG_FBPACKET_RESOLVER

Structure

// Use if wFBType == AER_FBTYPE_RESOLVER
typedef struct tagAER_CFG_FBPACKET_RESOLVER
{
    WORD  wChannel;   // Position feedback channel 1 to 16
    WORD  wResolution // Resolution of resolver (number of bits
        // in R/D
        // converter, must be 10, 12, 14, 16,
        // based on R/D
        // converter hardware used)
    WORD  wPoles;      // Number of poles (must be even, use 0
        // for a DC motor)
    WORD  wCommOffset // Commutation offset (Hall effect)
    WORD  wBounded;    // 1 to activate software limits, 0 to
        // disable them
} AER_CFG_FBPACKET_RESOLVER;
typedef AER_CFG_FBPACKET_RESOLVER *PAER_CFG_FBPACKET_RESOLVER;

Members

See inline comments in the structure listed above.

See

AerConfig
C.13. **AER_CFG_FBPACKET_RESOLVER_HALL**

Structure

```c
// Use if wFBType == AER_FBTYPE_RESOLVER_HALL
typedef struct tagAER_CFG_FBPACKET_RESOLVER_HALL
{
    WORD  wChannel;    // Position feedback channel 1 to 16
    WORD  wResolution; // Resolution of resolver (number of bits
    // in R/D
    // converter, must be 10, 12, 14, 16,
    // based on
    // R/D converter hardware used)
    DWORD dwHallLines; // Lines per electrical cycle after x4
    // multiplication
    WORD  wCommOffset; // Commutation offset (Hall effect)
    WORD  wCommChannel; // Feedback channel for commutation
    WORD  wBounded;    // 1 to activate software limits, 0 to
    // disable them
} AER_CFG_FBPACKET_RESOLVER_HALL;
typedef AER_CFG_FBPACKET_RESOLVER_HALL
*PAER_CFG_FBPACKET_RESOLVER_HALL;
```

Members

See inline comments in the structure listed above.

See

*AerConfigHallEffect*
C.14. AER_CFG_FBPACKET_STEPPER

Structure

```c
typedef struct tagAER_CFG_FBPACKET_STEPPER
{
    WORD wChannel;   // Position feedback channel 1 to 16
    DWORD dwLines;   // Lines per motor revolution
    DWORD dwCommLines; // Number of lines per electrical cycle
    WORD wBounded;   // 1 to activate software limits, 0 to disable them
} AER_CFG_FBPACKET_STEPPER;
```

typedef AER_CFG_FBPACKET_STEPPER *PAER_CFG_FBPACKET_STEPPER;

Members

See inline comments in the structure listed above.

See

`AerConfig`
C.15. AER_CFG_IOPACKET_D2A

Structure

```c
// Use if wIOType == AER_IOTYPE_D2A
typedef struct tagAER_CFG_IOPACKET_D2A
{
    WORD wChannel;     // current/velocity output channel
                       // 1 to 16
} AER_CFG_IOPACKET_D2A;

typedef AER_CFG_IOPACKET_D2A *PAER_CFG_IOPACKET_D2A;
```

Members

See inline comments in the structure listed above.

See

AerConfig
C.16. AER_CFG_MASTER_PACKET

Structure

typedef struct tagAER_CFG_MASTER_PACKET
{
    WORD  wFBType;    // Type of master feedback. Use
                    // AER_MFBTYPE_xxx constant.
    union
    {
        char  szFBData[16]; // Padding - do not use
        struct
        {
            WORD  wChannel;   // position feedback channel 1 to 16
            WORD  wResolution; // resolution of resolver
        } Resolver;
        struct
        {
            WORD  wChannel;     // position feedback channel 1 to 16
            DWORD dwLines;      // lines per rev
        } Encoder;
        struct
        {
            WORD  wChannel;     // position feedback channel 1 to 16
        } Virtual;
    } FB;
} AER_CFG_MASTER_PACKET;

typedef AER_CFG_MASTER_PACKET *PAER_CFG_MASTER_PACKET;

Members

See inline comments in the structure listed above.

See

AerrRegGetDeviceInfo
C.17. AER_CFG_PACKET

The AER_CFG_PACKET structure defines configuration information used by the AerConfig functions. This large packet establishes how an axis is configured. It consists of four unions: FB, IO, SP1, and SP2; and four types associated with each union. The structures associated with the members of this structure, are detailed in the structure definitions following this one.

Structure

typedef struct tagAER_CFG_PACKET
{
    WORD wFBType; // Set = AER_FBTYPE_xxxx
    WORD wIOType; // Set = AER_IOTYPE_xxxx
    WORD wSpare1Type; // Set = AER_SPARETYPE1_xxxx
    WORD wSpare2Type; // Set = AER_SPARETYPE2_xxxx
    union
    {
        BYTE szFBData[16]; // Padding - not used
        AER_CFG_FBPACKET_NULL Null; // No Feedback
        AER_CFG_FBPACKET_RESOLVER Resolver; // Resolver
        AER_CFG_FBPACKET_ENCODER Encoder; // Encoder
        AER_CFG_FBPACKET_ENCODER_HALL EncoderHall; // Encoder with Hall
        AER_CFG_FBPACKET_HPVME Hp; // HP VME
        AER_CFG_FBPACKET_RESOLVER_HALL ResolverHall; // Hall Resolver
        AER_CFG_FBPACKET_STEPPER Stepper; // Stepper
    } FB;
    union
    {
        BYTE szIOData[16]; // Null
        AER_CFG_IOPACKET_D2A D2ACard; // D2A card
    } IO;
    union
    {
        BYTE szSpare1Data[16]; // Null
        AER_CFG_SP1PACKET_RESOLVER Resolver; // Resolver
        AER_CFG_SP1PACKET_ENCODER Encoder; // Encoder
        AER_CFG_SP1PACKET_STEPPER Stepper; // Stepper
    } SP1;
    union
    {
        BYTE szSpare2Data[16]; // Not used
    } SP2;
} AER_CFG_PACKET;
typedef AER_CFG_PACKET *PAER_CFG_PACKET;

Members

See inline comments in the structure listed above.

See

AerConfig
C.18. AER_CFG_SP1PACKET_ENCODER

Structure

// Use if wSpare1Type == AER_SPARETYPE1_ENCODER 
// Use if wSpare1Type == AER_SPARETYPE1_NEW_ENCODER 
// Use if wSpare1Type == AER_SPARETYPE1_SLAVE_ENCODER 
typedef struct tagAER_CFG_SP1PACKET_ENCODER
{
    WORD  wChannel; // Position feedback channel 1 to 16
    DWORD dwLines; // Lines per motor revolution
    WORD  wVelHomeFlag; // Set to 1 to use HOME marker on
    // velocity transducer. For
    // AER_SPARETYPE1_NEW_ENCODER and
    // AER_SPARETYPE1_SLAVE_ENCODER only
}
AER_CFG_SP1PACKET_ENCODER;

typedef AER_CFG_SP1PACKET_ENCODER *PAER_CFG_SP1PACKET_ENCODER;

Members

See inline comments in the structure listed above.

See

AerConfigHallResolver
C.19  AER_CFG_SP1PACKET_RESOLVER

Structure

// Use if wSpare1Type == AER_SPARETYPE1_RESOLVER          1
// Use if wSpare1Type == AER_SPARETYPE1_NEW_RESOLVER      5
typedef struct tagAER_CFG_SP1PACKET_RESOLVER
{
  WORD  wChannel; // Position feedback channel 1 to
                   // 16
  WORD  wResolution; // Resolution of resolver (number
                     // of bits in R/D converter, must
                     // be 10, 12, 14, 16, based on R/D
                     // converter hardware used)
  WORD  wPoles;  // For AER_SPARETYPE1_NEW_RESOLVER
                  // only
  WORD  wCommOffset; // For AER_SPARETYPE1_NEW_RESOLVER
                     // only
  WORD  wCommutationOnlyFlag; // For AER_SPARETYPE1_NEW_RESOLVER
                             // only
} AER_CFG_SP1PACKET_RESOLVER;
typedef AER_CFG_SP1PACKET_RESOLVER *PAER_CFG_SP1PACKET_RESOLVER;

Members
See inline comments in the structure listed above.

See

AerConfigHall
C.20. AER_COMPILE_ERROR_DATA

Structure

typedef struct        // Type returned by
{                      // AerCompilerGetErrData()
    AERERR_CODE dwCode; // Code of error (see aercode.h +
    // AerErr.rc for code meanings)
    // (AERERR_NOERR if no error)
    LONG dwProgLine;  // PROGRAM Line that error occurred. (0-
    // based) (-1 if not relevant)
    LONG dwFileLine;  // FILE Line that error occurred. (0-
    // based) (-1 if not relevant)
    LONG dwCharStart; // First Character of offending token (0-
    // based) (-1 if not applicable)
    LONG dwCharEnd;   // Last Character of offending token (0-
    // based) (-1 if not applicable)
    TCHAR pszFile[MAX_PATH]; // File error occurred in (or "" if
    // not applicable)
} AER_COMPILE_ERROR_DATA;

Members

See inline comments in the structure listed above.

See

AerCompilerGetErrData
C.21. AER_COMPILE_STATUS_DATA

Structure

typedef struct  // Type returned by  
AerCompilerGetStatus()
{
    DWORD dwState;  // (see states below)
    TCHAR pszState[MAX_STATE_CHARS]; // Verbal equivalent of  
    // dwState
    DWORD dwProgLine;  // Current program line being  
    // compiled (is total num at  
    // end)
    DWORD dwFileLine; // Current file line being  
    // compiled
    DWORD dwNErrs;  // Number of errors+warnings  
    // so far.
    DWORD dwNWarns; // Number of warnings so far.
    TCHAR pszFile[MAX_PATH];  // Current file being scanned    
    // (only pass 0 relevant,  
    // else blank.)
    LONG ctime;                // time (milliseconds) spent  
    // in compile
    LONG dtime;                // time (milliseconds) spent  
    // in download
} AER_COMPILE_STATUS_DATA;

Members

Compile states are as follows:

COMPILER_STATE_NULL   // Created, but nothing else  
// done.
COMPILER_STATE_PREPROC // Now preprocessing
COMPILER_STATE_PASS1  // Now in pass1
COMPILER_STATE_PASS2  // Now in pass2
COMPILER_STATE_DOWNLOADING // Now downloading
COMPILER_STATE_DONE_OK 5 // Compiled successfully, no  
// warnings
COMPILER_STATE_DONE_WITH_WARNINGS // Compiled successfully, with  
// warnings
COMPILER_STATE_DONE_WITH_ERRORS // Failed compile due to errors  
COMPILER_STATE_DOWNLOADED // Compiled and downloaded  
// successfully (may or may not  
// have warnings)
COMPILER_STATE_DOWNLOADERERROR // Compiled successfully, but  
// failure in download (may or  
// may not have compile  
// warnings)
COMPILER_STATE_DOWNLOADING // currently downloading
C.22. AER_GSTRIP_AXIS_DATA

Structure

typedef struct tagAER_GSTRIP_AXIS_DATA
{
    LONG lPos; // Position (in counts)
    LONG lPosCommand; // Position command (in counts)
    LONG lRawPos; // Raw position (in counts)
    LONG lMasterPos; // Master position (in counts)
    SHORT sVelocity; // Filtered velocity command, in
                      // counts/millisecond
    SHORT sVelocityCommand; // in counts/millisecond
    SHORT sAcceleration; // in counts/millisecond^2
    SHORT sTorque; // in units of -32K to +32K, times
                    // the maximum torque for that motor.
} AER_GSTRIP_AXIS_DATA;

typedef AER_GSTRIP_AXIS_DATA *PAER_GSTRIP_AXIS_DATA;

Members

Raw position is normally equal to position, only when axis calibration is active do they differ: raw position is before axis calibration, position is after axis calibration. The sVelocity member is not velocity directly from the feedback device, but it is the derivative of position from the feedback device computed once every millisecond (see axis parameter VELTIMECONST).

See

AerStripGlobalGetSample, AerAxisCalxxxx
**C.23. AER_GSTRIP_SAMPLE**

**Structure**

typedef struct tagAER_GSTRIP_SAMPLE
{
    PAER_GSTRIP_SYSTEM_DATA pSystem;
    PAER_GSTRIP_AXIS_DATA pAxis[MAX_AXES];
} AER_GSTRIP_SAMPLE;

typedef AER_GSTRIP_SAMPLE *PAER_GSTRIP_SAMPLE;

**Members**

This structure contains pointers to other structures. One pointer is for the global strip data (not related to a specific axis) the other pointers are for the axis data, one pointer per axis.

See the `GSTRIP_SYSTEM_DATA` and `GSTRIP_AXIS_DATA` structure definitions for more details.

See `AerStripGlobalGetSample`
C.24. AER_GSTRIP_SYSTEM_DATA

Structure

typedef struct tagAER_GSTRIP_SYSTEM_DATA
{
    WORD  wAnalogInput[8];  // Analog inputs
    DWORD dwIO;              // Debug, for internal use only
    LONG  lClock;            // Clock value (milliseconds)
} AER_GSTRIP_SYSTEM_DATA;

typedef AER_GSTRIP_SYSTEM_DATA   *PAER_GSTRIP_SYSTEM_DATA;

Members

The specified analog inputs are either onboard the UNIDEX 600/620 controller or are
the analog inputs on the Matrix analog input card in the UNIDEX 631 VME chassis,
unless the analog probe is enabled (see AerProbxxx routines), in which case these are
the probe inputs. The clock value is the number of milliseconds after the last axis
processor card reset.

See

AerStripGlobalGetSample
C.25. AER_MOTN_LINEAR_PACKET

Structure

```
typedef struct tagAER_MOTN_LINEAR_PACKET
{
    AXISMASK mAxis;   // Bitmask of axis to move.
    DWORD dwSpeed;    // Vectorial speed in machine
                      // steps/second.
    DWORD dwTarget[MAX_AXES]; // Array of distances for each
                      // axis to move.
} AER_MOTN_LINEAR_PACKET;
```

typedef AER_MOTN_LINEAR_PACKET *PAER_MOTN_LINEAR_PACKET;

Members

See inline comments in the structure listed above.

See

AerMotionLinear
C.26. AER_MOTN_PACKET

Structure

typedef struct tagAER_MOTN_PACKET
{
    DWORD dwMove;   // Distance in machine steps
    DWORD dwSpeed;  // Speed in machine steps/second
} AER_MOTN_PACKET;
typedef AER_MOTN_PACKET  *PAER_MOTN_PACKET;

Members
See inline comments in the structure listed above.

See
AerMotionPacket
C.27. AER_PARM_INFO

Structure

typedef struct tagAER_PARM_INFO
{
    CHAR   szName[MAX_PARM_NAME_LEN]; // Parameter name (see AerParmxxxx)
    DOUBLE fdMin; // Minimum value
    DOUBLE fdMax; // Maximum value
    DWORD  dwAttr; // Attribute mask (see Constants chapter under PARM_ATTR)
    DOUBLE fdDefault; // Default value
    DWORD  dwDisplaySubGroup; // Which subgroup to display the parameter in
} AER_PARM_INFO;

Members
See inline comments in the structure listed above.

See
AerParmxxxx
AerProbePosPacket
AerProbeStatusPacket
C.28. AER_PROBE_POS_PACKET

Structure

```c
typedef struct tagAER_PROBE_POS_PACKET
{
    WORD wStatus;    // AER_PROBESTATUS_xxx constant
    DWORD dwPos;     // Position probe was triggered
} AER_PROBE_POS_PACKET;

typedef AER_PROBE_POS_PACKET *PAER_PROBE_POS_PACKET;
```

Members

See inline comments in the structure listed above.

See

Probe Functions
C.29. AER_PROBE_STATUS_PACKET

Structure
typedef struct tagAER_PROBE_STATUS_PACKET
{
    WORD wStatus; // AER_PROBESTATUS_xxx constant
    WORD wInput;  // Probe Input
    WORD wLevel;  // Active level
} AER_PROBE_STATUS_PACKET;
typedef AER_PROBE_STATUS_PACKET *PAER_PROBE_STATUS_PACKET;

Members
    See inline comments in the structure listed above.

See
    Probe Functions
C.30. AER_PROFILE

Structure
typedef struct tagAER_PROFILE
{
    LONG lPoint; // Target Position in counts
    DWORD dwTime; // Motion time in msec
    LONG lVelStart; // Starting Velocity in msec/sec
    LONG lVelEnd; // Ending Velocity in msec/sec
} AER_PROFILE;
typedef AER_PROFILE *PAER_PROFILE;

Members
See inline comments in the structure listed above.

See
AerProfileLoadQueue
AerProfileLoadQueueMult
C.31. AER_PROGFAULT

Structure
typedef struct tagAER_PROG_FAULT
{
    DWORD wErrCode;  // error code number
    char szErrMsg[80];  // ASCII error message
    DWORD wErrLen;  // message length in bytes
    char szErrData[80];  // error data in hexadecimal format
} AER_PROG_FAULT;
typedef AER_PROG_FAULT *PAER_PROG_FAULT;

Members
See inline comments in the structure listed above.

See
AerProgGetFault
C.32. AER_PROG_HANDLE

Structure
typedef struct tagAER_PROG_HANDLE
{
    CHAR szName[MAX_PROG_NAME_LEN]; // Unique name for
    // referencing programs on
    // the controller.
} AER_PROG_HANDLE;
typedef AER_PROG_HANDLE *PAER_PROG_HANDLE;

Members
See inline comments in the structure listed above.

See
AerProgramAllocate
AerTaskProgramxxxx
C.33. AER_PROG_HEADER

Structure
typedef struct tagAER_PROG_HEADER
{
    DWORD    dwNumLines960;  // Number of program code lines
    DWORD    dwSizeBytes;    // Size in bytes for entire program
    DWORD    dwNumLabels;    // Number of program labels
    DWORD    dwNumDoubles;   // Number of double variables
    DWORD    dwNumStrings;   // Number of string variables
} AER_PROG_HEADER;
typedef AER_PROG_HEADER  *PAER_PROG_HEADER;

Members
See inline comments in the structure listed above.

See
AerProgramxxxx
C.34. AER_PROG_INFO

Structure

typedef struct tagAER_PROG_INFO
{
    AER_PROG_STATUS Status;    // Program status information
    DWORD dwNumLines960;      // Number of program code lines
    DWORD dwNumDoubles;       // Number of double variables in
                              // program
    DWORD dwNumStrings;       // Number of string variables in
                              // program
    DWORD dwNumLabels;        // Number of program labels in
                              // program
} AER_PROG_INFO;
typedef AER_PROG_INFO *PAER_PROG_INFO;

Members

See inline comments in the structure listed above.

See

AerProgramGetInfo
C.35. AER_PROG_LABEL

Structure
typedef struct tagAER_PROG_LABEL
{
    CHAR szName[MAX_PROG_LABEL_LEN]; // Label name
} AER_PROG_LABEL;
typedef AER_PROG_LABEL *PAER_PROG_LABEL;

Members
See inline comments in the structure listed above.

See
AER_PROG_LABEL_INFO
C.36. AER_PROG_LABEL_INFO

Structure
typedef struct tagAER_PROG_LABEL_INFO
{
   AER_PROG_LABEL Label;         // Label
   DWORD          dwLine960;     // Label line number
} AER_PROG_LABEL_INFO;
typedef AER_PROG_LABEL_INFO   *PAER_PROG_LABEL_INFO;

Members
   See inline comments in the structure listed above.

See
   AerProgramLoadLabel
   AerProgramGetLabel
C.37. AER_PROG_STATUS

Structure
typedef union tagAER_PROG_STATUS
{
    DWORD    dwStatus[1]; // Program Status, see
    // the bit breakdown below
struct
    {
        unsigned TaskAssociated:MAX_TASKS; // Mask of tasks that
        // are associated with
        // this program
        unsigned MemoryAllocated:1; // AerProgramAllocate has
        // been called for this
        // program
        unsigned LabelFilled:1; // All labels that have
        // been allocated, have
        // been downloaded
        unsigned Queue:1; // 1 if a queued program
        unsigned nCodeFilled:1; // All code lines that
        // has been allocated,
        // has been downloaded
        unsigned nLoadComplete:1; // Everything that has
        // been allocated, is now
        // downloaded
        unsigned qBufferFull:1; // Queue is currently full
        unsigned qBufferEmpty:1; // Queue is currently empty
    } Bit;
} AER_PROG_STATUS;
typedef AER_PROG_STATUS *PAER_PROG_STATUS;

Members
See inline comments in the structure listed above. The dwStatus member is broken into a series of bits, each of which can be true or false. The nCodeFilled and nLoadComplete members are not relevant for queued programs (if Queue is true), and the qBufferFull and qBufferEmpty bits are not relevant for unqueued programs (if Queue is false). See AerCompilerDownloadQueue for details on queues. nLoadComplete is true if and only if nCodeFilled and LabelFilled are true. LabelFilled is true only if MemoryAllocated is true, and AerProgramLoadLabel has been called for each label specified in the program header passed in through AerProgramAllocate. nCodeFilled is true only if MemoryAllocated is true, and AerProgramLoadLine has been called for each line specified in the program header passed in through AerProgramAllocate.

See
AerProgramGetInfo
C.38. AER_PROG_STATUS

Structure
typedef union tagAER_PROG_STATUS
{
    DWORD   dwStatus[1];  // Program Status, see
    struct
    {
        unsigned TaskAssociated:MAX_TASKS;  // Mask of tasks that
        // are associated with
        // this program
        unsigned MemoryAllocated:1;  // AerProgramAllocate
        // has been called for
        // this program
        unsigned LabelFilled:1;  // All labels that have
        // been allocated, have
        // been downloaded
        unsigned Queue:1;  // True if a queued
        // program
        unsigned nCodeFilled:1;  // All code lines that
        // have been allocated,
        // have been downloaded
        unsigned nLoadComplete:1;  // Everything that has
        // been allocated, is
        // now downloaded
        unsigned qBufferFull:1;  // Queue is currently
        // full
        unsigned qBufferEmpty:1;  // Queue is currently
        // empty
    } Bit;
} AER_PROG_STATUS;
typedef AER_PROG_STATUS *PAER_PROG_STATUS;

Members
See inline comments in the structure listed above. The dwStatus member is broken into a series of bits, each of which can be true or false. The nCodeFilled and nLoadComplete members are not relevent for queued programs (if Queue is true), and the qBufferFull and qBufferEmpty bits are not relevent for unqueued programs (if Queue is false). See AerCompilerDownloadQueue for details on queues. nLoadComplete is true if and only if nCodeFilled and LabelFilled are true. LabelFilled is true only if MemoryAllocated is true, and AerProgramLoadLable has been called for each label specified in the program header passed in through AerProgramAllocate. nCodeFilled is true only if MemoryAllocated is true, and AerProgramLoadLine has been called for each line specified in the program header passed in through AerProgramAllocate.

See
AerProgramGetInfo
C.39. AER_PSO_D2A

Structure
typedef struct tagAER_PSO_D2A
{
    WORD wChannel;        // Channel Number (0-1)
    union
        {
            double dVoltage;      // Immediate Write Voltage
            struct
                {
                    double dZeroVolts;  // Voltage at Zero Velocity
                    double dTargVolts;  // Voltage at Target Velocity
                    DWORD  dTarget;     // Target Velocity
                } tVel;
            struct
                {
                    double dCurrVolts; // Voltage at ZeroVelocity
                    double dTargVolts;  // Voltage at Target Velocity
                    ULONG  dTarget;     // Target Velocity
                } tPos;
        }
} AER_PSO_D2A;
typedef AER_PSO_D2A *PAER_PSO_D2A;

Members
    See inline comments in the structure listed above.

See
    AerPSOSetMultiAnalogOutput
C.40. AER_PSO_D2A_PACKET

Structure

typedef union
    {
        DWORD dwOutputValue;       // DWORD containing fields below
        struct
            {
                BYTE byValue[3];       // Voltage to output II.FF FFv
                BYTE byNumber;         // D/A Channel Number
            } tOutput;
    } tOutput;
AER_PSO_D2A_PACKET;
typedef AER_PSO_D2A_PACKET *PAER_PSO_D2A_PACKET;

Members

See inline comments in the structure listed above.
C.41. AER_PULSE_DEFINE

Structure
typedef struct tagAER_PULSE_DEFINE
{
    DWORD dwLead;        // De-assertion Time Preceding Ramp-up
    DWORD dwWidth;       // Assertion Time at Ramp Completion
    DWORD dwTail;        // De-assertion Time Following Ramp-down
} AER_PULSE_DEFINE;

Members
See inline comments in the structure listed above.

See
AerPSOSetPulse
C.42. AER_PULSE_DEFINE_RAMP

**Structure**

```c
typedef struct tagAER_PULSE_DEFINE_RAMP
{
  DWORD dwLead;     // De-assertion Time Preceding Ramp-up
  DWORD dwWidth;    // Assertion Time at Ramp Completion
  DWORD dwTail;     // De-assertion Time Following Ramp-down
  WORD  wRampTime;  // Pulse Ramp Increment
  WORD  wRampInterval; // Interval between Ramp Increments
} AER_PULSE_DEFINE_RAMP;
```

**Members**

See inline comments in the structure listed above.

**See**

`AerPSOSetPulse`
C.43. AER_PULSE_WIDTH

Structure
typedef struct tagAER_PULSE_WIDTH
{
    DWORD dwWidth; // Duration of Laser Firing Pulse
} AER_PULSE_WIDTH;

Members
See inline comments in the structure listed above.

See

AerPSOSetPulse
C.44. AER_REG_DEVICE_INFO

The AER_REG_DEVICE_INFO structure defines device information used by some of the Aerotech Registry functions.

**Structure**

```c
typedef struct tagAER_REG_DEVICE_INFO
{
    TCHAR szBootImageName[MAX_TEXT_LEN+1]; // Boot image name
    TCHAR szImageName[MAX_TEXT_LEN+1]; // Image name
    TCHAR szSymbolicName[MAX_TEXT_LEN+1]; // Symbolic Link name
    AER_UNIDEX_INFO tUNIDEX; // Device info
} AER_REG_DEVICE_INFO;

typedef AER_REG_DEVICE_INFO *PAER_REG_DEVICE_INFO;
```

**Members**

- **szBootImageName**
  - Name of boot image file used by axis processor. This is not used by UNIDEX 631.

- **szImageName**
  - Name of image file (firmware) used by axis processor.

- **szSymbolicName**
  - This is the name used by the system to establish communications with the device driver and axis processor.

  For Windows 95 this is the name of device driver (.vxd) including path and extension (i.e. “C:\U600\VU600D.VXD”).

  For Windows NT this is the name of the device driver (.sys) without path and extension (i.e. used by the system “U600”).

**See**

- AerRegSetDeviceInfo
- AerRegGetDeviceInfo
C.45. AER_STRIP_SAMPLE

Structure

typedef struct tagAER_STRIP_SAMPLE
{
    LONG  lPos;       // Position
    LONG  lPosCommand; // Position command
    LONG  lSpare;     // “Accel-term”
    WORD  wTorque;    // Torque
} AER_STRIP_SAMPLE;
typedef AER_STRIP_SAMPLE  *PAER_STRIP_SAMPLE;

Members

Position and position command are in units of counts.
Torque is in units of -32K to +32K, times the maximum torque for that motor.
The “Accel-term” is the commanded acceleration times the accel gain. The commanded
acceleration is that determined by the users axis parameters concerning acceleration and
deceleration (see ACCELRATE, ACCELMODE axis parameters) The commanded
acceleration is in units of counts per millisecond squared. The accel gain is an axis parameter
available to the user, (the AFF_GAIN axis parameter) for the purposes of affecting the servo
loop operation.

See

    AerStripGetSample
C.46. AER_TASK_DATA

Structure
typedef struct tagAER_TASK_DATA
{
   AER_TASK_STATUS Status;    // Task status information
   AER_TASK_MODE Mode; // Task mode information
   DWORD dwCallStackDepth; // Call stack depth level
   DWORD dwCurrentProgNumber; // Currently executing program number
   DWORD dwCurrentLine960; // Currently executing 960 line number
   DWORD dwCurrentLineUser; // Currently executing user line number
   DWORD dwCurrentPriorityLevel; // Currently priority level
   AERERR_CODE Fault; // Task fault
} AER_TASK_DATA;
typedef AER_TASK_DATA *PAER_TASK_DATA;

Members
See inline comments in the structure listed above.

See
TASKPARM_TaskFault
TASKPARM_MaxCallStack
AER_TASK_MODE
AER_TASK_STATUS
AerDCCGetTaskDirect
C.47. **AER_TASK_MODE**

**Structure**

// Please see documentation on the associated G/M codes listed on the
// right, for more details.

typedef struct tagAER_TASK_MODE
{
    union
    {
        DWORD dwMask;
        struct
        {
            unsigned English:1; // 0 - (G70 mode)
            unsigned Absolute:1; // 1 - (G90 mode)
            unsigned AccelModeLinear:1; // 2 - (G64 mode)
            unsigned AccelModeRate:1; // 3 - (G68 mode)
            unsigned RotaryDominant:1; // 4 - (G98 mode)
            unsigned MotionContinuous:1; // 5 - (G108 mode)
            unsigned InverseCircular:1; // 6 - (G111 mode)
            unsigned SpindleShutDownOnProgHalt:1; // 7 (G101 mode)
            unsigned BlockDelete:1; // 8 - (G112 mode)
            unsigned OptionalStop:1; // 9 - (G114 mode)
            unsigned BreakPoint:1; // 10 - Currently on a
            // breakpoint
            unsigned MFOLock:1; // 11 - (M48 mode)
            unsigned MSOLock:1; // 12 - (M50 mode)
            unsigned Reserved1:1; // not used
            unsigned Retrace:1; // 14 - Currently in
            // ReTrace mode
            unsigned AutoMode:1; // 15 - Currently not in
            // single step mode
            unsigned ProgramFeedRateMPU:1; // 16 - (G93 mode)
            unsigned ProgramFeedRateUPR:1; // 17 - (G94 mode)
            unsigned ProgramSFeedRateSurf1:1; // 18 - (G95 mode)
            unsigned ProgramSFeedRateSurf2:1; // 19 - (G195 mode)
            unsigned ProgramSFeedRateSurf3:1; // 20 - (G295 mode)
            unsigned ProgramSFeedRateSurf4:1; // 21 - (G395 mode)
            unsigned BlockDelete2:1; // 22 - (G212 mode)
            unsigned RunOverMode:1; // 23 - Currently in
            // Step over mode
        } Bit;
    } DW1;

typedef AER_TASK_MODE *PAER_TASK_MODE;

**Members**

See TASKMODE definitions for bit descriptions.

**See**

TASKMODE1_xxxx
AerTaskGetModeName
AerDCGetTaskDirect
AER_TASK_DATA
C.48. AER_TASK_STATUS

Structure

typedef struct tagAER_TASK_STATUS
{
    union
    {
        DWORD    dwMask;
        struct
        {
            unsigned ProgramAssociated:1;
            unsigned ProgramActive:1;
            unsigned ProgramExecuting:1;
            unsigned ImmediateCodeExecuting:1;
            unsigned ReturnMotionExecuting:1;
            unsigned NotUsed:1;
            unsigned SingleStepInto:1;
            unsigned SingleStepOver:1;
            unsigned InterruptFaultPending:1;
            unsigned InterruptCallBackPending:1;
            unsigned EStopInputActive:1;
            unsigned FeedHoldInputActive:1;
            unsigned FeedHoldInputLatch:1;
            unsigned CallBackHoldActive:1;
            unsigned CallBackResponding:1;
            unsigned ProgramCleanup:1;
            unsigned ProgramCodeCleanup:1;
            unsigned OnGosubPending:1;
        } Bit;
    } DW1;

    union
    {
        DWORD    dwMask;
        struct
        {
            unsigned SpindleActive:MAX_SPINDLES;
            unsigned MSOChange:1;
            unsigned SpindleFeedHoldActive:1;
            unsigned ASyncFeedHoldActive:1;
        } Bit;
    } DW2;

    union
    {
        DWORD    dwMask;
        struct
        {
            unsigned RotationActive:1;
            unsigned RThetaPolarActive:1;
            unsigned RThetaCylindricalActive:1;
            unsigned OffsetPresetActive:1;
            unsigned OffsetFixtureActive:1;
            unsigned OffsetManualActive:1;
            unsigned MotionType:2;        // MOTIONTYPE_
            unsigned MotionActive:1;
        }
    }
}
unsigned MotionContinuous:1;
unsigned MFOChange:1;
unsigned MotionFeedHoldActive:1;
unsigned CutterEnabling:1;
unsigned CutterActiveLeft:1;
unsigned CutterActiveRight:1;
unsigned CutterDisabling:1;
unsigned NormalcyActiveLeft:1;
unsigned NormalcyActiveRight:1;
unsigned NormalcyAlignment:1;
unsigned ProgramFeedRateMPU:1;
unsigned ProgramFeedRateUPR:1;
unsigned LimitFeedRateActive:1;
unsigned LimitMFOActive:1;
} Bit;
} DW3;
} AER_TASK_STATUS;
typedef AER_TASK_STATUS *PAER_TASK_STATUS;

Members
See TASKSTATUS definitions for bit descriptions.

See
TASKSTATUS1_xxxx
TASKSTATUS2_xxxx
TASKSTATUS3_xxxx
AerTaskGetStatusName
AerDCGetTaskDirect
AER_TASK_DATA
C.49. AER_TORQ_PACKET

Structure

typedef struct tagAER_TORQ_PACKET
{
    WORD  wMode;  // 0/1 to disable/enable torque mode
    WORD  wChannel;  // A/D channel number for temperature sensor.
    FLOAT  dAmpsVel;  // Velocity compensation term expressed in
    // D/A bits per count/sec
    FLOAT  dAmpsTemp;  // Temperature compensation
    WORD  wNomTemp;  // Nominal (room) operating temperature
    // expressed in A/D bits
}  AER_TORQ_PACKET;

typedef AER_TORQ_PACKET  *PAER_TORQ_PACKET;

Members

See inline comments in the structure listed above.

See

AerTorqueSetModePacket
AerTorqueGetModePacket
C.50. AER_U600_INFO

The AER_U600_INFO structure defines device information specific to the UNIDEX 600 card.

Structure
typedef struct tagAER_U600_INFO
{
    DWORD dwDSC;  // DRAM size code
} AER_U600_INFO;
typedef AER_U600_INFO *PAER_U600_INFO;

Members
    dwDSC
        DRAM size code.

See
    AER_UNIDEX_INFO
C.51. AER_U631_INFO

The AER_U631_INFO structure defines device information specific to the UNIDEX 631 controller.

Structure
typedef struct tagAER_U631_INFO
{
    DWORD dwVMEAddr; // VME Address
} AER_U631_INFO;
typedef AER_U631_INFO *PAER_U631_INFO;

Members
    dwVMEAddr
        VME address.

See
    AER_UNIDEX_INFO
C.52. AER_UNIDEX_INFO

The AER_UNIDEX_INFO structure defines device information used to describe how a device is setup.

Structure

typedef struct tagAER_UNIDEX_INFO
{
    DWORD    dwDeviceID;        // Device Identifier
    (AER_UNIDEX_xxx constant)
    DWORD    dwATWindow;  // AT Window
    DWORD    dwIOBase;       // I/O Base address
    DWORD    dwIRQ;          // IRQ number
    union {
        AER_U600_INFO   tU600;
        AER_U631_INFO   tU631;
    } tDevice;
} AER_REG_CARD_INFO;

typedef AER_REG_CARD_INFO  *PAER_REG_CARD_INFO;

Members

dwDeviceID
    Device identifier. Determines which member of tDevice to use.
dwATWindow
    AT window that the axis processor uses to address data.
dwIOBase
    Base I/O address setting of axis processor.
dwIRQ
    Interrupt the axis processor will use.
dwDSC
    DRAM size code.
tDevice
    Device specific data that is setup based on the DeviceID.

See

AerRegSetDeviceInfo
AerRegGetDeviceInfo
C.53. AER_VERSION

Structure

typedef struct tagAER_VERSION
{
    WORD wUNIDEX;    // AER_UNIDEX_xxxx constant
    WORD wInternal;  // Internal Version Number
    WORD wMajor;     // Major Version Number
    WORD wMinor;     // Minor Version Number
} AER_VERSION;
typedef AER_VERSION    *PAER_VERSION;

Members

See inline comments in the structure listed above.

See

AerVerGetImgVersion
C.54. AERERR_CODE

**Typedef**

A 32 bit unsigned number, defined as a ulong.

**See**

PAERERR_CODE
C.55. AERVIRT_BINARY_DATA

This union is used to help create a map of the virtual binary input and output data. By using this structure, binary I/O can be accessed on BYTE, WORD, or DWORD boundaries.

Structure

```c
typedef union tagAERVIRT_BINARY_DATA
{
    BYTE tByte[MAX_BINARY_BYTES];  // Map of binary bytes
        // (8 bit)
    WORD tWord[MAX_BINARY_WORDS];  // Map of binary words
        // (16 bit)
    DWORD tDWord[MAX_BINARY_DWORDS];  // Map of binary dwords
        // (32 bit)
} AERVIRT_BINARY_DATA;
typedef AERVIRT_BINARY_DATA   *PAERVIRT_BINARY_DATA;
```

Members

See inline comments in the structure listed above.

See

- AerVirtGetBinaryIO
- AerVirtSetBinaryIO
C.56. AERVIRT_REGISTER_DATA

This union is used to help create a map of the virtual register input and output data. By using this structure, register I/O can be accessed on WORD boundaries.

Structure

```c
typedef union tagAERVIRT_REGISTER_DATA
{
    WORD  tWord[MAX_REGISTERS];  // Map of registers (16 bit)
} AERVIRT_REGISTER_DATA;
typedef AERVIRT_REGISTER_DATA *PAERVIRT_REGISTER_DATA;
```

Members

See inline comments in the structure listed above.

See

```
AerVirtGetRegisterI0
AerVirtSetRegisterI0
```
C.57. ANALOGINDEX

**Typedef**

An ANALOGINDEX is a 0 based number specifying the analog channel.

**See**

PANALOGINDEX
C.58. AXISINDEX

**Typedef**

An AXISINDEX is a 0 based number referring to a specific physical axis. It is defined to be a DWORD.

**See**

AXISMASK
AXISINDEXxxxx
PAXISINDEX
C.59. AXISMASK

**Typedef**
An AXISMASK is a bitwise mask representing a set of physical axis. Bit 0 corresponds to axis 1, etc. It is defined to be a DWORD.

**See**
AXISINDEX
PAXISMASK
C.60. BOOL

typedef

A logical value being 1 or 0, defined as an int.

See

PBOOL
C.61. BYTE

Typedef
An 8 bit signed number, defined as a byte.

See
PBYTE
C.62. CHAR

**Typedef**

An 8 bit unsigned character, defined as a char.

**See**

PCHAR


C.63. CSPARMINDEX

**Typedef**

A 0 based number specifying a call stack parameter, defined as a DWORD. The maximum range is determined by the `MaxCallStack` task parameter (−1).

**See**

PCSPARMINDEX
CSPARMINDEX_xxxx
C.64. CSPARMMASK

**Typedef**
A bitmask representing multiple call stack parameters, defined as a DWORD.

**See**
PCSPARMMASK
C.65.  DOUBLE

Typedef
   A double precision number defined as a double.

See
   PDOUBLE
C.66. DWORD

typedef

A 32 bit unsigned number, defined as a ulong.
C.67. FLOAT

Typedef
    A floating point number, defined as a float.

See
    PFLOAT
C.68. HAERCTRL

Typedef
A HAERCTRL is a handle to the axis processor card created by the AerSysOpen function, which is defined as DECLARE_HANDLE (C language definition).

See
AerSysOpen and/or PHAERCTRL
C.69. HANDLE

**Typedef**

A handle is a 32 bit value.

**See**

PHANDLE
C.70. HCOMPILER

**Typedef**

A HCOMPILER is a handle to an Aerotech compiler session created by the `AerCompilerCreate` function. HCOMPILER is defined as DECLARE_HANDLE (C language definition).

**See**

`AerCompilerCreate`
C.71. LONG

Typedef
A 32 bit signed number, defined as a long.

See
PLONG
C.72. LPARAM

typedef

A 32 bit value, defined as a long.
C.73.  LPCTSTR

**Typedef**

A constant pointer to a null terminated string of characters, defined as a char *. 
C.74. LPTSTR

Typedef

A pointer to a null terminated string of characters, defined as a char *.
C.75. **PAERERR_CODE**

**Typedef**

A pointer to a 32 bit signed number, defined as a ulong *.

**See**

AERERR_CODE
C.76. PANALOGINDEX

Typedef

A pointer to an analog channel specifier.

See

ANALOGINDEX
C.77. PAXISINDEX

Typedef    
A pointer to an axis specifier, defined as a DWORD *. The range is 0 through 15.

See        
AXISINDEX
C.78. PAXISMASK

typedef

A pointer to a bitmask specifying multiple axes. It is defined to be a DWORD *.

See

AXISMASK
C.79. PBOOL

Typedef

A pointer to a logical value being 1 or 0, defined as an int.

See

BOOL
C.80. **PBYTE**

**Typedef**

A pointer to an 8 bit signed number, defined as a byte *.

**See**

BYTE
C.81. PCHAR

Typedef
   A pointer to an 8 bit unsigned character, defined as a char *.

See
   CHAR
C.82. PCSPARMINDEX

Typedef

A pointer to a 0 based number specifying a call stack parameter, defined as a DWORD *.

See

CSPARMINDEX
C.83.  PCSPARMMASK

typedef
   A pointer to a bitmask representing multiple call stack parameters, defined as a
   DWORD *.

See
   CSPARMMASK
C.84. PDOUBLE

Typedef
   A pointer to a double precision number defined as a double *.

See
   DOUBLE
C.85. PDWORD

**Typedef**

A pointer to a 32 bit signed number, defined as a ulong *.

**See**

DWORD
C.86. **PFLOAT**

**Typedef**

A pointer to a floating point number, defined as a `float *`.

**See**

`FLOAT`
C.87. PHAERCTRL

Typedef
A pointer to a handle to the axis processor card created by the AerSysOpen function, which is defined as DECLARE_HANDLE *(C language definition).

See
AerSysOpen and/or HAERCTRL
C.88. PHANDLE

typedef

A pointer to a handle, which is a 32 bit value.

See

HANDLE
C.89. PHYSAXISINDEX

typedef

An axis specifier, defined as a DWORD. The range is 0 thru 15 to specify all 16 axes.

See

PHYSAXISMASK
PHYSAXISINDEX_xxxx
PPHYSAXISINDEX
C.90. PHYSAXISMASK

**Typedef**

A pointer to a bitmask specifying multiple axes. It is defined to be a DWORD. The first axis is represented by bit 0 and the last axis by bit 15.

**See**

PHYSAXISINDEX
PPHYSAXISMASK
C.91. PLONG

Typedef

A pointer to an 8 byte signed number, defined as a long double *.

See

LONG
C.92. **PPHYSAXISINDEX**

**Typedef**

A pointer to an axis specifier, defined as a *DWORD*.

**See**

PHYSAXISINDEX
C.93. PPHYSAXISMASK

**Typedef**

A pointer to a bitmask representing multiple axes, defined as a `DWORD *`.

**See**

PHYSAXISINDEX
C.94. **PSPINDLEINDEX**

**Typedef**

A pointer to a 0 based number specifying a spindle, defined as a `DWORD *`.

**See**

SPINDLEINDEX
C.95. PSPINDLEMASK

Typedef
A pointer to a bitmask specifying multiple spindles, defined as a DWORD *.

See
SPINDLEMASK
C.96. PSZ

Typedef

A pointer to a NULL terminated character string, defined as a char *.

Aerotech, Inc
C.97.  PTASKAXISINDEX

**Typedef**  
A pointer to a 0 based number specifying a task number, defined as a *DWORD*.

**See**  
TASKAXISINDEX
C.98. **PTASKAXISMASK**

**Typedef**

A pointer to a bitmask specifying multiple tasks, defined as a `DWORD *`.

**See**

`TASKAXISMASK`
C.99. **PTASKINDEX**

Typedef

A pointer to a task number specifier, which is defined as `DWORD *`. 
C.100. PTASKMASK

**Typedef**

A pointer to a bitmask specifying multiple task number’s, defined as `DWORD *`.

**See**

TASKMASK
C.101. PUINT

typedef A pointer to an unsigned integer, defined as an unsigned *

See

UINT
C.102. PVOID

typedef

A pointer to an unknown data type, defined as VOID *. 
C.103. PWORD

Typedef
   A pointer to a 16 bit signed number, defined as a *ushort.

See
   WORD
C.104. SHORT

**Typedef**

A signed 16 bit number, defined as a short.
C.105. SPINDLEINDEX

**Typedef**

A 0 based number specifying a spindle, defined as a *DWORD*. There are four spindles represented as 0 thru 3.

**See**

PSPINDLEINDEX
C.106. SPINDLEMASK


typedef

A bitmask specifying multiple spindles, defined as a DWORD. Bit 0 represents spindle 1.

See

PSPINDLEMASK
C.107. TASKAXISINDEX

Typedef

A 0 based number specifying a task number, defined as a DWORD. There are four possible tasks represented by 0 thru 3.

See

PTASKAXISINDEX
C.108. TASKAXISMASK

**Typedef**
A bitmask specifying multiple tasks, defined as a *DWORD*. Task 0 is represented by bit 0.

**See**
PTASKAXISMASK
C.109. TASKINDEX

**Typedef**

A TASKINDEX is a 0 based number referring to a specific task. It is defined to be a DWORD.

**See**

TASKMASK
TASKINDEX xxxx
PTASKINDEX
C.110. TASKMASK

typedef

A TASKMASK is a bitwise mask representing a set of tasks. Bit 0 corresponds to task 1, etc. It is defined to be a DWORD.

See

TASKINDEX
PTASKINDEX
C.111. UINT

**Typedef**

An unsigned 16 bit integer, defined as an unsigned.

**See**

PUINT
C.112. WORD

**Typedef**

A 16 bit signed number, defined as a ushort.

**See**

PWORD
APPENDIX D: WARRANTY AND FIELD SERVICE

In This Section:
- Laser Products...................................................... D-1
- Return Procedure..................................................... D-1
- Returned Product Warranty Determination.................. D-1
- Returned Product Non-warranty Determination............. D-2
- Rush Service.......................................................... D-2
- On-site Warranty Repair ........................................... D-2
- On-site Non-warranty Repair ..................................... D-2

Aerotech, Inc. warrants its products to be free from defects caused by faulty materials or poor workmanship for a minimum period of one year from date of shipment from Aerotech. Aerotech's liability is limited to replacing, repairing or issuing credit, at its option, for any products that are returned by the original purchaser during the warranty period. Aerotech makes no warranty that its products are fit for the use or purpose to which they may be put by the buyer, where or not such use or purpose has been disclosed to Aerotech in specifications or drawings previously or subsequently provided, or whether or not Aerotech's products are specifically designed and/or manufactured for buyer's use or purpose. Aerotech's liability or any claim for loss or damage arising out of the sale, resell or use of any of its products shall in no event exceed the selling price of the unit.

Aerotech, Inc. warrants its laser products to the original purchaser for a minimum period of one year from date of shipment. This warranty covers defects in workmanship and material and is voided for all laser power supplies, plasma tubes and laser systems subject to electrical or physical abuse, tampering (such as opening the housing or removal of the serial tag) or improper operation as determined by Aerotech. This warranty is also voided for failure to comply with Aerotech's return procedures.

Claims for shipment damage (evident or concealed) must be filed with the carrier by the buyer. Aerotech must be notified within (30) days of shipment of incorrect materials. No product may be returned, whether in warranty or out of warranty, without first obtaining approval from Aerotech. No credit will be given nor repairs made for products returned without such approval. Any returned product(s) must be accompanied by a return authorization number. The return authorization number may be obtained by calling an Aerotech service center. Products must be returned, prepaid, to an Aerotech service center (no C.O.D. or Collect Freight acceptable). The status of any product returned later than (30) days after the issuance of a return authorization number will be subject to review.

After Aerotech's examination, warranty or out-of-warranty status will be determined. If upon Aerotech's examination a warranted defect exists, then the product(s) will be repaired at no charge and shipped, prepaid, back to the buyer. If the buyer desires an air freight return, the product(s) will be shipped collect. Warranty repairs do not extend the original warranty period.
After Aerotech’s examination, the buyer shall be notified of the repair cost. At such time
the buyer must issue a valid purchase order to cover the cost of the repair and freight, or
authorize the product(s) to be shipped back as is, at the buyer’s expense. Failure to obtain
a purchase order number or approval within (30) days of notification will result in the
product(s) being returned as is, at the buyer’s expense. Repair work is warranted for (90)
days from date of shipment. Replacement components are warranted for one year from
date of shipment.

At times, the buyer may desire to expedite a repair. Regardless of warranty or out-of-
 warranty status, the buyer must issue a valid purchase order to cover the added rush
service cost. Rush service is subject to Aerotech’s approval.

If an Aerotech product cannot be made functional by telephone assistance or by sending
and having the customer install replacement parts, and cannot be returned to the Aerotech
service center for repair, and if Aerotech determines the problem could be warranty-
related, then the following policy applies:

Aerotech will provide an on-site field service representative in a reasonable amount of
time, provided that the customer issues a valid purchase order to Aerotech covering all
transportation and subsistence costs. For warranty field repairs, the customer will not be
charged for the cost of labor and material. If service is rendered at times other than
normal work periods, then special service rates apply.

If during the on-site repair it is determined the problem is not warranty related, then the
terms and conditions stated in the following "On-Site Non-Warranty Repair" section
apply.

If any Aerotech product cannot be made functional by telephone assistance or purchased
replacement parts, and cannot be returned to the Aerotech service center for repair, then
the following field service policy applies:

Aerotech will provide an on-site field service representative in a reasonable amount of
time, provided that the customer issues a valid purchase order to Aerotech covering all
transportation and subsistence costs and the prevailing labor cost, including travel time,
necessary to complete the repair.

Aerotech, Inc.
101 Zeta Drive
Pittsburgh, PA  15238-2897
USA

∇  ∇  ∇
Absolute position, 12-6
AER_AUX_POINT, C-2
AER_AXIS_DATA_EX, C-5
AER_AXISCAL_PACKET, C-3
AER_AXISCAL_STATUS_PACKET, C-4
AER_BP, A-2
AER_CAM_GETPOINT, 5-11, C-6
AER_CAM_SETPOINT, 5-17, C-7
AER_CARD, A-3
AER_CFG_FBPACKET_ENCODER, C-8
AER_CFG_FBPACKET_ENCODER_HALL, C-9
AER_CFG_FBPACKET_HPVME, C-10
AER_CFG_FBPACKET_NULL, C-11
AER_CFG_FBPACKET_RESOLVER, C-12
AER_CFG_FBPACKET_RESOLVER_HALL, C-13
AER_CFG_FBPACKET_STEPPER, C-14
AER_CFG_IOPACKET_D2A, C-15
AER_CFG_MASTER_PACKET, C-16
AER_CFG_PACKET, 7-1, C-17
AER_CFG_SP1PACKET_ENCODER, C-18
AER_CFG_SP1PACKET_RESOLVER, C-19
AER_COMPILE_ERROR_DATA, C-20
AER_COMPILE_STATUS_DATA, C-21
AER_CPUSTAT, A-4
AER_DRV600_DEFAULT, A-5
AER_EVENT, A-6
AER_FBTYPE_XXXX, A-7
AER_GSTRIP_AXIS_DATA, C-22
AER_GSTRIP_SAMPLE, C-23
AER_GSTRIP_SYSTEM_DATA, C-24
AER_IOTYPE_XXXX, A-8
AER_MFBTYPE_XXXX, A-10, B-38
AER_MOTN_LINEAR_PACKET, C-25
AER_MOTN_PACKET, C-26
AER_PARM_INFO, C-27
AER_PROBE_POS_PACKET, C-28
AER_PROBE_STATUS_PACKET, C-29
AER_PROBESTAMP_XXXX, A-9
AERPROFILE, C-30
AER_PROGFALULT, C-31
AER_PROGHANDLE, C-32
AER_PROGHEADER, C-33
AER_PROGINFO, C-34
AER_PROGLABEL, C-35
AER_PROGLABEL_INFO, C-36
AER_PROGSTATUS, C-37, C-38
AER_PSO_DATA, 18-6
AER_PULSE_DEFINE, 18-8, C-41
AER_PULSE_DEFINE_RAMP, 18-8, C-42
AER_PULSE_WIDTH, 18-8, C-43
AER_REGDEVICE_INFO, C-44
AER_SPARETYPE1_XXXX, A-11
AER_SPARETYPE2_XXXX, A-12
AER_STRIP_SAMPLE, C-45
AER_TASK_DATA, C-46
AER_TASK_MODE, C-47
AER_TASK_STATUS, C-48
AER_TORQ_PACKET, C-50
AER_U600_INFO, C-51
AER_U631_INFO, C-52
AER_UNIDEX, A-13
AER_UNIDEX_INFO, C-53
AER_VERSION, C-54, C-56, C-57
AerAnalogIndex, B-2
AerAnalogMask, B-3
AerAutoProgSystem, B-4
AerAutoProgType, B-5
AerAuxTableAllocate, 3-1, 3-2
AerAuxTableGetMode, 3-4
AerAuxTableGetMultPoints, 3-6
AerAuxTableGetPoint, 3-5
AerAuxTableGetStatus, 3-7
AerAuxTableGetTemp, 3-1, 3-9
AerAuxTableGetTempMultPoints, 3-11
AerAxisCal2DAllocateTable, 4-22
AerAxisCal2DDataDownload, 4-23
AerAxisCal2DFreeTable, 4-24
AerAxisCal2DGetData, 4-25
AerAxisCal2DGetMode, 4-26
AerAxisCal2DGetPoint, 4-27
AerAxisCal2DSetData, 4-28
AerAxisCal2DSetMode, 4-29
AerAxisCal2DSetTemp, 4-30
AerAxisCalAllocatableTable, 4-4
AerAxisCalDataDownload, 4-5
AerAxisCalFreeTable, 4-7
AerAxisCalGet, 4-8
AerAxisCalGetItems, 4-9
AerAxisCalGetMode, 4-10
AerAxisCalGetRedZone, 4-11
AerAxisCalGetRawPosition, 4-13
AerAxisCalGetStatus, 4-14
AerAxisCalGetStatusPacket, 4-15
AerAxisCalRemoveAll, 4-16
AerAxisCalReset, 4-17
AerAxisCalSet, 4-18
AerAxisCalSetMode, 4-19
AerAxisCalSetTemp, 4-20
AerAxisIndex, B-6
AerAxisMask, B-7
AerAxisParm, B-8
AerAxisState, B-11
AerAxisType, B-12
AerBit, B-13
AerBrkPnt, B-14
AERCAM, A-14
AerCamTableAllocate, 5-4
AerCamTableCalcCoeff, 5-2, 5-5
AerCamTableDataDownload, 5-6
AerCamTableFree, 5-9
AerPSOSetDigitalOutput, 18-4
AerPSOSetFiringDist, 18-5
AerPSOSetMultAnalogOutput, 18-6
AerPSOSetPulse, 18-7
AerPSOSetTracking, 18-9
AerPSOStopFiring, 18-10
AerPSOTable, B-47
AerPSOWriteMultFiringPoints, 18-11
AerPtrType, B-48
AerPulseWidth, B-49
AerRegGetDefDeviceInfo, 19-4
AerRegGetDeviceInfo, 19-5
AerRegGetDeviceInfoEx, 19-5
AerRegGetFileName, 7-7, 19-7
AerRegID, B-50
AERREGID, A-17
AerRegQueryCardCount, 19-8
AerRegQueryCardList, 19-9
AerRegSetDefDevice, 19-10
AerRegSetFileName, 19-12
AerResp, 10-1
AerReturnInfo, B-51
AerRIOT, B-52
AerServoStat, B-53
AerSPIType, B-57
AerSPO2Type, B-58
AerStat, B-54
AerStatus, B-56
AerStripAllocate, 20-2
AerStripFree, 20-3
AerStripGetIOPosLatchStatus, 20-4
AerStripGetSample, 20-5
AerStripGetStatus, 20-6
AerStripGlobalAllocate, 20-7
AerStripGlobalFree, 20-9
AerStripGlobalFreeTrigger, 20-10
AerStripGlobalGetImmediate, 20-11
AerStripGlobalGetQueue, 20-12
AerStripGlobalGetQueueDecimate, 20-13
AerStripGlobalGetQueueRecent, 20-14
AerStripGlobalGetSample, 20-15
AerStripGlobalGetTriggerPoint, 20-17
AerStripGlobalGetTriggerStatus, 20-18
AerStripGlobalHalt, 20-19
AerStripGlobalHold, 20-20
AerStripGlobalRelease, 20-21
AerStripGlobalSetTrigger, 20-22
AerStripGlobalSetTriggerPoint, 20-24
AerStripHalt, 20-25
AerStripIOPosLatch, B-55
AerStripSetIOPosLatch, 20-26
AerStripSetTrigger, 20-28
AerStripTrigger, 20-29
AerSysClose, 21-2
AerSysCloseAll, 21-3
AerSysDownLoad, 21-4
AerSysFaultAck, 21-5
AerSysInitSystem, 21-7
AerSysOpen, 21-8
AerSysOpenEx, 21-9
AerSysReset, 21-10
AerTaskAxisIndex, B-59
AerTaskCallBackContinue, 22-4
AerTaskCallBackContinueEx, 22-5
AerTaskCallBackGetData, 22-6
AerTaskCallBackReturnSetDouble, 22-7
AerTaskCallBackReturnSetString, 22-8
AerTaskCallBackValueDouble, 22-9
AerTaskCallBackValueGetDWORD, 22-10
AerTaskCallBackValueGetString, 22-11
AerTaskCallBackValueMakeString, 22-12
AerTaskCallBackValueSetDouble, 22-13
AerTaskCallBackValueSetDWORD, 22-14
AerTaskCallBackValueSetString, 22-15
AerTaskCallBackValueValidateNumArgs, 22-16
AerTaskExec, B-60
AerTaskGetPhysAxisFromTaskAxis, 22-17
AerTaskGetTaskAxisFromPhysAxis, 22-18
AerTaskImmediateBindAxis, 22-19
AerTaskImmediateCaptureAxis, 22-20
AerTaskImmediateFreeAxis, 22-21
AerTaskImmediateMapAxis, 22-22
AerTaskIndex, B-61
AerTaskMask, B-62
AerTaskMode, B-63
AerTaskModeGetName, 22-23
AerTaskParm, B-64
AerTaskProgramAbort, 22-24
AerTaskProgramAssociate, 22-25
AerTaskProgramDeAssociate, 22-26
AerTaskProgramExecute, 22-27
AerTaskProgramGetLineUser, 22-28
AerTaskProgramReset, 22-29
AerTaskProgramSetLineUser, 22-30
AerTaskProgramStop, 22-32
AerTaskReset, 22-33
AerTaskSetExecuteMode, 22-34
AerTaskSetRetraceMode, 22-35
AerTaskSetSlewMode, 22-36
AerTaskStatus, B-67
AerTaskStatusGetName, 22-37
AerTool, B-68
AERTOOL, A-18
AerTorqueGetMode, 24-2
AerTorqueSetMode, 24-3
AerUtilAlignMaster, 25-2
AerUtilHitWatchdogTimer, 25-3
AerUtilMotorSet, 25-4
AerUtilSet1khzServoLoop, 25-5
AerUtilSet4khzServoLoop, 25-6
AerVarGlobalGetDouble, 26-3
AerVarGlobalGetString, 26-4
AerVarGlobalSetDouble, 26-5
AerVarGlobalSetString, 26-6
AerVarProgramGetDouble, 26-7
AerVarProgramSetDouble, 26-8
AerVarStackGetDouble, 26-9
AerVarStackSetDouble, 26-10
AerVarTaskGetDouble, 26-11
AerVarTaskGetString, 26-12
AerVarTaskSetDouble, 26-13
AerVerGetBootVersion, 27-2
AerVerGetCpuStatus, 27-3
AerVerGetDrvVersion, 27-4
AerVerGetDrvVersionEx, 27-4
AerVerGetDrvVersionString, 27-4
AerVerGetImgVersion, 27-5
AerVerGetImgVersionEx, 27-5
AerVerGetImgVersionString, 27-5
AerVerGetLibVersion, 27-7
AerVerGetLibVersionEx, 27-7
AerVerGetOS, 27-9
AerVersion, B-69
AerVerStringToVersion, 27-10
AerVerToString, 27-11
AERVIRT_BINARY_DATA, C-56
AERVIRT_REGISTER_DATA, C-57
AerVirtGetBinaryInput, 28-2
AerVirtGetBinaryInputByteEx, 28-3
AerVirtGetBinaryInputDWordEx, 28-4
AerVirtGetBinaryInputWordArray, 28-5
AerVirtGetBinaryInputWordEx, 28-6
AerVirtGetBinaryIO, 28-7
AerVirtGetBinaryOutput, 28-8
AerVirtGetBinaryOutputByteEx, 28-9
AerVirtGetBinaryOutputDWordEx, 28-10
AerVirtGetBinaryOutputWordArray, 28-11
AerVirtGetBinaryOutputWordEx, 28-12
AerVirtGetRegisterInput, 28-13
AerVirtGetRegisterInputArray, 28-14
AerVirtGetRegisterInputEx, 28-15
AerVirtGetRegisterIO, 28-16
AerVirtGetRegisterOutput, 28-17
AerVirtGetRegisterOutputArray, 28-18
AerVirtGetRegisterOutputEx, 28-19
AerVirtSetBinaryInput, 28-20
AerVirtSetBinaryInputByteEx, 28-21
AerVirtSetBinaryInputDWordEx, 28-22
AerVirtSetBinaryInputWordArray, 28-23
AerVirtSetBinaryInputWordEx, 28-24
AerVirtSetBinaryIO, 28-25
AerVirtSetBinaryOutput, 28-26
AerVirtSetBinaryOutputByteEx, 28-27
AerVirtSetBinaryOutputDWordEx, 28-28
AerVirtSetBinaryOutputWordArray, 28-29
AerVirtSetRegisterInput, 28-31
AerVirtSetRegisterInputArray, 28-32
AerVirtSetRegisterInputEx, 28-33
AerVirtSetRegisterIO, 28-34
AerVirtSetRegisterOutput, 28-35
AerVirtSetRegisterOutputArray, 28-36
AerVirtSetRegisterOutputEx, 28-37
ALT_STAT, A-19
ANALOGINDEX, A-20, C-58
Application
  Debugging, 1-9
Auxiliary tables, 3-1
AUXOFFSET, 3-12
Axis calibration tables, 4-1
AXISINDEX, A-21, C-59
AXISMASK, A-22, C-60
AXISPM, A-23
AXISTYPE, A-26

B

BOOL, C-61
Building, 1-4
BYTE, C-62

C

C/C++ Language Programming Quick Start, 1-7
Calibration file, +5
Calling, 1-5, 2-2
Cam table, 3-1, 5-1, 5-6
Cam table ASCII file, 5-6
CAMADVANCE, 5-20
CAMOFFSET, 5-18, 5-20
CHAR, C-63
Closing communications, 1-3
Communication, 1-1
Configuration, 1-2
CSPARMINDEX, A-27, C-64
CSPARMASK, C-65

D

Data Center functions, 8-1
Debugging, 1-9
Device driver, 1-1
DOUBLE, C-66
DRIVE, 4-31
DWORD, C-67

E

Encoder, 7-3
Error codes, 1-2
Error handling, 1-11
Establishing communications, 1-2
Event functions, 10-1
Events, 10-1

F

Fault conditions, 1-3
Faults, 1-12
Index

Field Service Policy, D-1
Firmware, 1-1, 1-2
FLOAT, C-68
FLT, A-28

G
GLOBPARM, A-29

H
HAERCTRL, 1-5, 2-2, 21-2, 21-3, 21-10, C-69
HANDLE, C-70
HCOMPILER, 6-1, 6-15, 6-36, C-71
HOMETYPE, A-30
Homing, 12-7

I
ICMDPOLARITY, 7-3
Image file, 1-2
Inductosyn, 7-3
Initializing the U600 card, 21-7
Install
Libraries and Utilities, 1-2
IOLEVEL, 3-12

J
Joystick enable, 22-36

L
Lab View Programming Quick Start, 1-8
Laser functions, 18-1
LONG, C-72
LPARAM, C-73
LPTCSTR, C-74
LPTSTR, C-75

M
MACHPARM, A-31
MASTERLENGTH, 3-12, 5-20
MASTERPOS, 3-12, 5-20
MAX_, A-32
MAXCAMACCEL, 5-20
Memory functions, 11-1
Motion, 1-2
MOTION_, A-33
Move functions, 12-7

P
PAERERR_CODE, C-76
PANALOGINDEX, C-77

Parameter functions, 13-1
PARM_ATTR, A-34, B-39
PAXISINDEX, C-78
PAXISMASK, C-79
PBOOL, C-80
PBYTE, C-81
PCCHAR, C-82
PCSPARMINDEX, C-83
PCSPARMMASK, C-84
PDWORD, C-85
PFLOAT, C-87
PHANDLE, C-89
PHOMETYPE, A-90
PHYSAXISINDEX, A-35, C-91
PHYSAXISMASK, C-91
PLONG, C-92
PPHYSAXISINDEX, C-93
PPHYSAXISMASK, C-94

Probe functions, 14-1
Profile functions, 15-1
Program functions, 17-1
Programming error functions, 16-1
Programming Errors, 1-12
PROGTYPE, A-36
PSO functions, 18-1
PSO_CHANNEL_MASK, A-37
PSO_MODE, A-38
PSO_TABLE, A-39
PSPINDLEINDEX, C-95
PSPINDLEMASK, C-96
PSZ, C-97
PTASKAXISINDEX, C-98
PTASKAXISMASK, C-99
PTASKINDEX, C-100
PTASKMASK, C-101
PUINT, C-102
PULSE_, A-40
PVOID, C-103
PWORD, C-104

Queue mode, 6-41
Quick Start
C/C++ Language Programming, 1-7
Developing and application, 1-7
LabView Programming, 1-8
Example Programs, 1-8
Visual Basic, 1-6
Developing an application, 1-6
Qxxxx, B-70

Resolver, 7-3
REVERSALMODE, 4-31
REVERSALVALUE, 4-31
Revision History, R-1

S
SERVO_, A-41
SHORT, C-105
SPINDLEINDEX, A-42, C-106
SPINDLEMASK, C-107
STAT, A-43
Status information, 1-3
Strip functions, 20-1
STRIP_IOPOSTATCH, A-44
STRIP_STATUS, A-45
STRIP_TRIGGER, A-46
STRIPGLOBAL_MODE, A-47
STRIPGLOBAL_STATUS, A-48
System DLLs, 1-1
System functions, 21-1

T
Task functions, 22-1
TASKAXISINDEX, A-49, C-108
TASKAXISMASK, C-109
TASKEXEC, A-50
TASKINDEX, A-51, C-110
TASKMASK, C-111
TASKMODE, A-52
TASKPARM, A-53
TASKSTATUS, A-56
Torque functions, 23-1, 24-1
Two dimensional (2D) axis calibration, 4-21

U
UINT, C-112
Utility functions, 25-1

V
Variable functions, 26-1
Version functions, 27-1
Virtual I/O functions, 28-1
Visual Basic Quick Start, 1-6

W
Warranty Policy, D-1
Win32, 1-1
WORD, C-113
REVISION HISTORY

Revisions

The following section provides the user with general information regarding the latest changes to this manual. Extensive changes, if made, may not be itemized – instead, the section or chapter will be listed with “extensive changes / additions” in the corresponding General Information cell.

Table R-1. Revisions

<table>
<thead>
<tr>
<th>Revision</th>
<th>Section(s) Affected</th>
<th>General Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.4</td>
<td>All pages</td>
<td>Full revision – extensive changes / additions / deletions throughout manual.</td>
</tr>
</tbody>
</table>

∇ ∇ ∇
READER’S COMMENTS

Please answer the questions below and add any suggestions for improving this document. Is the information:

Yes  No

Adequate to the subject?  ____  ____
Well organized?  ____  ____
Clearly presented?  ____  ____
Well illustrated?  ____  ____
Would you like to see more illustrations?  ____  ____
Would you like to see more text?  ____  ____

How do you use this document in your job? Does it meet your needs?
What improvements, if any, would you like to see? Please be specific or cite examples.
_____________________________________________________________________
_____________________________________________________________________
_____________________________________________________________________
_____________________________________________________________________
_____________________________________________________________________

Your name  ________________
Your title  ________________
Company name  ________________
Address  ________________

Remove this page from the document and fax or mail your comments to the technical writing department of Aerotech.

AEROTECH, INC.
Technical Writing Department
101 Zeta Drive
Pittsburgh, PA. 15238-2897 U.S.A.
Fax number (412) 967-6870