Table of Contents

GETTING STARTED ................................................................................................................................. 9
SYSTEM REQUIREMENTS ......................................................................................................................... 9
INSTALLATION ............................................................................................................................................ 10
FILES IN THE OASIS LIBRARY DEVELOPER KIT .................................................................................. 10

USING THE OASIS SOFTWARE .................................................................................................................. 12
MOVING AXES AND COMPONENTS ........................................................................................................... 14
  Positional Units and Component Calibration ....................................................................................... 14
  The Coordinate System ....................................................................................................................... 14
  Moving a Single Axis .......................................................................................................................... 15
  Waiting for Movement Completion ..................................................................................................... 15
  Moving the XY Stage and Focus .......................................................................................................... 16
  Acceleration Tables ............................................................................................................................ 21
  Cruising Speed ................................................................................................................................... 23
  Encoder Support ................................................................................................................................. 24
SAVING SETTINGS .................................................................................................................................. 26
  Saving and Loading Settings .............................................................................................................. 26
  Saving and Loading Positions ............................................................................................................ 27
RETURN VALUES ........................................................................................................................................ 27

ADVANCED TOPICS .................................................................................................................................... 29
USING MULTIPLE OASIS CONTROLLERS ..................................................................................................... 29
  Counting the Number of Installed OASIS Controllers ........................................................................ 29
  Routing Commands to a Controller ..................................................................................................... 29
  Using General Axis Commands with Multiple Controllers ............................................................... 30
  Special Considerations When Using Multiple Controllers .................................................................. 31
GENERAL PURPOSE I/O ................................................................................................................................. 31

FUNCTION DESCRIPTIONS ........................................................................................................................ 33
HARDWARE CONTROL ................................................................................................................................. 34
  OI_Close ................................................................................................................................................ 35
  OI_CloseComponent ............................................................................................................................. 35
  OI_Configure ....................................................................................................................................... 36
  OI_CountCards .................................................................................................................................. 37
  OI_EmergencyStopAll .......................................................................................................................... 38
  OI_EnableMotorPower ........................................................................................................................ 38
  OI_EnableMotorPowerEx ...................................................................................................................... 38
  OI_GetAFFitted .................................................................................................................................. 39
  OI_GetAHMDelay ................................................................................................................................. 39
  OI_GetAutoFocusHWMode ................................................................................................................... 40
  OI_GetCardAxisCount .......................................................................................................................... 40
  OI_GetCardType .................................................................................................................................. 41
  OI_GetComponentStatus ....................................................................................................................... 41
  OI_GetConfiguration ............................................................................................................................ 43
  OI_GetDefaultAbortKeys ..................................................................................................................... 44
  OI_GetDefaultWaitCursorEnabled ....................................................................................................... 44
  OI_GetDriverOpen ............................................................................................................................... 45
  OI_GetFlashChecksum .......................................................................................................................... 45
  OI_GetHardwareMode ........................................................................................................................... 46
  OI_GetMultiAxisMode .......................................................................................................................... 46
  OI_GetSelectedCard ............................................................................................................................. 47
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>OI_HaltXY</td>
<td>95</td>
</tr>
<tr>
<td>OI_InitilizeXY</td>
<td>96</td>
</tr>
<tr>
<td>OI_LookupSpeedXY</td>
<td>96</td>
</tr>
<tr>
<td>OI_MoveToXY</td>
<td>97</td>
</tr>
<tr>
<td>OI_MoveToXY_Abs</td>
<td>98</td>
</tr>
<tr>
<td>OI_MoveToXY_Auto</td>
<td>98</td>
</tr>
<tr>
<td>OI_ReadLimitAlarmsXY</td>
<td>99</td>
</tr>
<tr>
<td>OI_ReadStatusXY</td>
<td>99</td>
</tr>
<tr>
<td>OI_ReadXY</td>
<td>100</td>
</tr>
<tr>
<td>OI_ReadXY_Abs</td>
<td>101</td>
</tr>
<tr>
<td>OI_SelectSpeedXY</td>
<td>101</td>
</tr>
<tr>
<td>OI_SetCruiseXY</td>
<td>102</td>
</tr>
<tr>
<td>OI_SetDriveSenseXY</td>
<td>103</td>
</tr>
<tr>
<td>OI_SetOriginXY</td>
<td>103</td>
</tr>
<tr>
<td>OI_SetPitchXY</td>
<td>104</td>
</tr>
<tr>
<td>OI_SetPositionXY</td>
<td>104</td>
</tr>
<tr>
<td>OI_SetRampXY</td>
<td>105</td>
</tr>
<tr>
<td>OI_SetUserLimitGuardDistanceXY</td>
<td>106</td>
</tr>
<tr>
<td>OI_SetUserLimitsXY</td>
<td>107</td>
</tr>
<tr>
<td>OI_StepX</td>
<td>107</td>
</tr>
<tr>
<td>OI_StepXY</td>
<td>108</td>
</tr>
<tr>
<td>OI_StepY</td>
<td>108</td>
</tr>
<tr>
<td>OI_WaitForStoppedXY</td>
<td>109</td>
</tr>
</tbody>
</table>

**Z / FOCUS CONTROL**

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>OI_CloseMouseWheelForFocus</td>
<td>110</td>
</tr>
<tr>
<td>OI_DriveContinuousZ</td>
<td>110</td>
</tr>
<tr>
<td>OI_GetBacklashZ</td>
<td>111</td>
</tr>
<tr>
<td>OI_GetCruiseZ</td>
<td>111</td>
</tr>
<tr>
<td>OI_GetDriveSenseZ</td>
<td>112</td>
</tr>
<tr>
<td>OI_GetMouseWheelPars</td>
<td>112</td>
</tr>
<tr>
<td>OI_GetMouseWheelZ</td>
<td>113</td>
</tr>
<tr>
<td>OI_GetRampZ</td>
<td>113</td>
</tr>
</tbody>
</table>

**SIMULTANEOUS THREE AXIS CONTROL**

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>OI_DriveContinuousXYZ</td>
<td>83</td>
</tr>
<tr>
<td>OI_HaltAllAxes</td>
<td>84</td>
</tr>
<tr>
<td>OI_MoveToXYZ</td>
<td>84</td>
</tr>
<tr>
<td>OI_MoveToXYZ_Auto</td>
<td>85</td>
</tr>
<tr>
<td>OI_ReadMaxMoveXYZ</td>
<td>86</td>
</tr>
<tr>
<td>OI_ReadXYZ</td>
<td>87</td>
</tr>
<tr>
<td>OI_SetPitchFromFlashXYZ</td>
<td>87</td>
</tr>
<tr>
<td>OI_SetPositionXYZ</td>
<td>88</td>
</tr>
<tr>
<td>OI_WaitForStoppedXYZ</td>
<td>88</td>
</tr>
</tbody>
</table>

**XY STAGE CONTROL**

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>OI_ClearUserLimitsXY</td>
<td>89</td>
</tr>
<tr>
<td>OI_DriveContinuousXY</td>
<td>90</td>
</tr>
<tr>
<td>OI_GetBacklashXY</td>
<td>91</td>
</tr>
<tr>
<td>OI_GetCruiseXY</td>
<td>91</td>
</tr>
<tr>
<td>OI_GetDriveSenseXY</td>
<td>92</td>
</tr>
<tr>
<td>OI_GetFullTravelXY</td>
<td>92</td>
</tr>
<tr>
<td>OI_GetPitchXY</td>
<td>93</td>
</tr>
<tr>
<td>OI_GetRampXY</td>
<td>93</td>
</tr>
<tr>
<td>OI_GetSpeedXY</td>
<td>94</td>
</tr>
<tr>
<td>OI_GetUserLimitGuardDistanceXY</td>
<td>94</td>
</tr>
<tr>
<td>OI_GetUserLimitsXY</td>
<td>95</td>
</tr>
<tr>
<td>OI_HaltXY</td>
<td>95</td>
</tr>
<tr>
<td>OI_LookupSpeedXY</td>
<td>96</td>
</tr>
<tr>
<td>OI_MoveToXY</td>
<td>97</td>
</tr>
<tr>
<td>OI_MoveToXY_Abs</td>
<td>98</td>
</tr>
<tr>
<td>OI_MoveToXY_Auto</td>
<td>98</td>
</tr>
<tr>
<td>OI_ReadLimitAlarmsXY</td>
<td>99</td>
</tr>
<tr>
<td>OI_ReadStatusXY</td>
<td>99</td>
</tr>
<tr>
<td>OI_ReadXY</td>
<td>100</td>
</tr>
<tr>
<td>OI_ReadXY_Abs</td>
<td>101</td>
</tr>
<tr>
<td>OI_SelectSpeedXY</td>
<td>101</td>
</tr>
<tr>
<td>OI_SetCruiseXY</td>
<td>102</td>
</tr>
<tr>
<td>OI_SetDriveSenseXY</td>
<td>103</td>
</tr>
<tr>
<td>OI_SetOriginXY</td>
<td>103</td>
</tr>
<tr>
<td>OI_SetPitchXY</td>
<td>104</td>
</tr>
<tr>
<td>OI_SetPositionXY</td>
<td>104</td>
</tr>
<tr>
<td>OI_SetRampXY</td>
<td>105</td>
</tr>
<tr>
<td>OI_SetUserLimitGuardDistanceXY</td>
<td>106</td>
</tr>
<tr>
<td>OI_SetUserLimitsXY</td>
<td>107</td>
</tr>
<tr>
<td>OI_StepX</td>
<td>107</td>
</tr>
<tr>
<td>OI_StepXY</td>
<td>108</td>
</tr>
<tr>
<td>OI_StepY</td>
<td>108</td>
</tr>
<tr>
<td>OI_WaitForStoppedXY</td>
<td>109</td>
</tr>
</tbody>
</table>
OI_GetSpeedZ
OI_GetUserLimitsZ
OI_HaltZ
OI_InitializeZ
OI_InitializeZLimits
OI_LookupSpeedZ
OI_MoveToZ
OI_MoveToZ_Abs
OI_OpenMouseWheelForFocus
OI_ReadLimitAlarmsZ
OI_ReadRangeZ
OI_ReadStatusZ
OI_ReadSyncZ
OI_ReadZ
OI_ReadZ_Abs
OI_RockZ
OI_SetCruiseZ
OI_SetDriveSenseZ
OI_SetMouseWheelPars
OI_SetMouseWheelZ
OI_SetOriginZ
OI_SetPitchZ
OI_SetPositionZ
OI_SetRampZ
OI_SetUserLimitsZ
OI_StepZ
OI_WaitForStoppedZ

F-AXIS (4th AXIS) CONTROL

OI_ClearUserLimitsF
OI_GetCruiseF
OI_GetDriveSenseF
OI_GetRampF
OI_GetSpeedF
OI_GetUserLimitsF
OI_HaltF
OI_InitializeF
OI_InitializeFRange
OI_LookupSpeedF
OI_MoveToF
OI_ReadF
OI_ReadLimitAlarmsF
OI_ReadRangeF
OI_ReadStatusF
OI_SelectSpeedF
OI_SetCruiseF
OI_SetDriveSenseF
OI_SetOriginF
OI_SetPitchF
OI_SetPositionF
OI_SetRampF
OI_SetUserLimitsF
OI_StepF
OI_WaitForStoppedF

T-AXIS (5th AXIS) AND S-AXIS (6th AXIS) CONTROL
OI_GetPredictiveFlag
OI_GetMultiPredictiveZ
OI_GetAutoPredictiveZ
OI_WaitForAutoFocus
OI_SetAutoFocusEx
OI_ReadFocusScoreEx
OI_ReadFocusScore
OI_GetFineFocusSamples
OI_GetAutoFocusThreshold
OI_AutoFocus_Step
OI_SetEncoderEnabledZ
OI_SetEncoderClosedLoopResponseXYZ
OI_ReadEncoderModule
OI_GetAxisEncoderFitted
OI_WaitForStoppedT
OI_StepT
OI_SetUserLimitsT
OI_SetPositionT
OI_SetPitchT
OI_SetDriveSenseT
OI_ReadT
OI_ReadRangeT
OI_GetAutoFocusEx
OI_ReadLimitAlarmsT
OI_ReadStatusT
OI_SelectSpeedT
OI_SetCrusTeT
OI_GetAutoFocus
OI_GetAutoFocus_Fine
OI_GetAutoFocus_Step
OI_GetAutoFocusEx
OI_GetAutoFocusThreshold
OI_GetFineFocusSamples
OI_GetAutoFocusProfile
OI_GetAutoFocusScore
OI_RequestAutoFocusStatus
OI_SetAutoFocus
OI_SetAutoFocusEx
OI_SetAutoFocusThreshold
OI_SetFineFocusSamples
OI_WaitForAutoFocus
OI_GetAutoPredictiveZ
OI_GetCoincDomain
OI_GetMultiPredictiveZ
OI_GetPredictiveFlag
OASIS4I DLL Manual Version 3.1.3

HARDWARE JOYSTICK AND TRACKBALL FUNCTIONS

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>OI_GetTrackballControl</td>
<td>216</td>
</tr>
<tr>
<td>OI_WaitForStoppedFilter</td>
<td>215</td>
</tr>
<tr>
<td>OI_SetShutterMulti</td>
<td>214</td>
</tr>
<tr>
<td>OI_SetShutterEx</td>
<td>213</td>
</tr>
<tr>
<td>OI_SetFilterTimeout</td>
<td>211</td>
</tr>
<tr>
<td>OI_SetFilterOffset</td>
<td>211</td>
</tr>
<tr>
<td>OI_SetFilterLocation</td>
<td>210</td>
</tr>
<tr>
<td>OI_SetFilterHomeOffset</td>
<td>210</td>
</tr>
<tr>
<td>OI_SelectFilterChanger</td>
<td>208</td>
</tr>
<tr>
<td>OI_InitFilterChanger</td>
<td>205</td>
</tr>
<tr>
<td>OI_GetShutterMulti</td>
<td>205</td>
</tr>
<tr>
<td>OI_GetAvailableShutterCount</td>
<td>202</td>
</tr>
<tr>
<td>OI_GetFilterOffset</td>
<td>201</td>
</tr>
<tr>
<td>OI_GetFilterName</td>
<td>201</td>
</tr>
<tr>
<td>OI_GetFilterOffset</td>
<td>201</td>
</tr>
<tr>
<td>OI_GetFilterPosition</td>
<td>202</td>
</tr>
<tr>
<td>OI_GetFilterTimeout</td>
<td>202</td>
</tr>
<tr>
<td>OI_GetAvailableShutterCount</td>
<td>202</td>
</tr>
<tr>
<td>OI_GetShutter</td>
<td>203</td>
</tr>
<tr>
<td>OI_GetShutterEx</td>
<td>203</td>
</tr>
<tr>
<td>OI_GetMultiPredictiveZ</td>
<td>182</td>
</tr>
<tr>
<td>OI_InitPredictiveZ</td>
<td>185</td>
</tr>
<tr>
<td>OI_UpdatePredictiveZ</td>
<td>185</td>
</tr>
</tbody>
</table>

VIDEO CAMERA INTERFACE FUNCTIONS

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>OI_GetVideoWindow</td>
<td>187</td>
</tr>
<tr>
<td>OI_IsVideoDetected</td>
<td>188</td>
</tr>
<tr>
<td>OI_GetVideoData</td>
<td>188</td>
</tr>
<tr>
<td>OI_GetVideoResults</td>
<td>189</td>
</tr>
<tr>
<td>OI_GetVideoResultsEx</td>
<td>190</td>
</tr>
<tr>
<td>OI_GetVideoResultsXY</td>
<td>191</td>
</tr>
<tr>
<td>OI_GetVideoResultsXYZF</td>
<td>192</td>
</tr>
<tr>
<td>OI_GetVideoResultsZ</td>
<td>193</td>
</tr>
<tr>
<td>OI_GetAvailableVideoCount</td>
<td>194</td>
</tr>
<tr>
<td>OI_GetVideoSettings</td>
<td>194</td>
</tr>
<tr>
<td>OI_GetVideoWindow</td>
<td>195</td>
</tr>
</tbody>
</table>

FILTER CHANGER FUNCTIONS

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>OI_ClearFilterHomeInfo</td>
<td>197</td>
</tr>
<tr>
<td>OI_DeleteFilter</td>
<td>198</td>
</tr>
<tr>
<td>OI_GetFilterChanger</td>
<td>198</td>
</tr>
<tr>
<td>OI_GetFilterChangerCount</td>
<td>199</td>
</tr>
<tr>
<td>OI_GetFilterCount</td>
<td>199</td>
</tr>
<tr>
<td>OI_GetFilterDescription</td>
<td>200</td>
</tr>
<tr>
<td>OI_GetFilterHomeOffset</td>
<td>200</td>
</tr>
<tr>
<td>OI_GetFilterName</td>
<td>201</td>
</tr>
<tr>
<td>OI_GetFilterChanger</td>
<td>201</td>
</tr>
<tr>
<td>OI_GetFilterChanger</td>
<td>201</td>
</tr>
<tr>
<td>OI_GetFilterPosition</td>
<td>202</td>
</tr>
<tr>
<td>OI_GetFilterTimeout</td>
<td>202</td>
</tr>
<tr>
<td>OI_GetAvailableShutterCount</td>
<td>202</td>
</tr>
<tr>
<td>OI_GetShutter</td>
<td>203</td>
</tr>
<tr>
<td>OI_GetShutterEx</td>
<td>203</td>
</tr>
<tr>
<td>OI_GetMultiPredictiveZ</td>
<td>205</td>
</tr>
<tr>
<td>OI_InitPredictiveZ</td>
<td>205</td>
</tr>
<tr>
<td>OI_MoveToFilter</td>
<td>206</td>
</tr>
<tr>
<td>OI_ReadFilterChangerInfo</td>
<td>207</td>
</tr>
<tr>
<td>OI_ReadFilterHomeInfo</td>
<td>208</td>
</tr>
<tr>
<td>OI_SelectFilterChanger</td>
<td>208</td>
</tr>
<tr>
<td>OI_SetFilterCount</td>
<td>209</td>
</tr>
<tr>
<td>OI_SetFilterDescription</td>
<td>209</td>
</tr>
<tr>
<td>OI_SetFilterHomeOffset</td>
<td>210</td>
</tr>
<tr>
<td>OI_SetFilterLocation</td>
<td>210</td>
</tr>
<tr>
<td>OI_SetFilterName</td>
<td>211</td>
</tr>
<tr>
<td>OI_SetFilterOffset</td>
<td>211</td>
</tr>
<tr>
<td>OI_SetFilterTimeout</td>
<td>211</td>
</tr>
<tr>
<td>OI_SetShutter</td>
<td>212</td>
</tr>
<tr>
<td>OI_SetShutterEx</td>
<td>213</td>
</tr>
<tr>
<td>OI_SetShutterMulti</td>
<td>214</td>
</tr>
<tr>
<td>OI_WaitForStoppedFilter</td>
<td>215</td>
</tr>
</tbody>
</table>

HARDWARE JOYSTICK AND TRACKBALL FUNCTIONS

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>OI_ClearTrackballStatus</td>
<td>215</td>
</tr>
<tr>
<td>OI_GetJoystickEnabled</td>
<td>216</td>
</tr>
<tr>
<td>OI_GetTrackballControl</td>
<td>216</td>
</tr>
<tr>
<td>OI_GetTrackballEnabled</td>
<td>217</td>
</tr>
</tbody>
</table>
OI_ReadTrackballStatus ................................................................. 217
OI_SetJoystickEnabled ............................................................... 219
OI_SetTrackballControl .............................................................. 219
OI_SetTrackballEnabled .............................................................. 220
TIMEOUTS ..................................................................................... 221
OI_GetAutoFocusTimeout ............................................................ 221
OI_GetMoveTimeout ................................................................... 222
OI_GetVideoTimeout ................................................................... 222
OI_SetAutoFocusTimeout ............................................................ 223
OI_SetMoveTimeout ................................................................... 223
OI_SetVideoTimeout ................................................................... 223
FILE I/O AND SETTINGS .............................................................. 224
OI_LoadPositions ......................................................................... 224
OI_LoadSettings .......................................................................... 225
OI_LoadSettingsEx ....................................................................... 225
OI_SavePositions ......................................................................... 226
OI_SaveSettings .......................................................................... 226
OI_SaveSettingsEx ....................................................................... 227
ERROR HANDLING ........................................................................ 227
OI_GetLastError .......................................................................... 227
OI_EnableMsgReportDlg ............................................................. 228
MICRONS / STEP CONVERSION .................................................. 229
OI_MicronsToAbsoluteX ............................................................. 229
OI_MicronsToAbsoluteY ............................................................. 229
OI_MicronsToAbsoluteZ ............................................................. 229
OI_MicronsToAbsoluteF ............................................................. 229
OI_MicronsToStepsX ................................................................. 230
OI_MicronsToStepsY ................................................................. 230
OI_MicronsToStepsZ ................................................................. 230
OI_MicronsToStepsF ................................................................. 230
OI_StepsToMicronsX ................................................................. 231
OI_StepsToMicronsY ................................................................. 231
OI_StepsToMicronsZ ................................................................. 231
OI_StepsToMicronsF ................................................................. 231
GENERAL PURPOSE I/O .............................................................. 231
OI_ReadInputPorts ....................................................................... 232
OI_ReadIO .................................................................................. 233
OI_WriteIO .................................................................................. 234
Getting Started

The OASIS automation library is a Windows Dynamic Link Library providing full control and access to the Objective Imaging OASIS range of microscope automation controllers. An OASIS controller provides independent, microstepping control of 4 axes of movement as well as optional capabilities via various plug-in daughter modules for high-performance applications in automated microscopy and digital image analysis.

The OASIS DLL simplifies the task of automation control by:

- Managing critical set-up and maintenance tasks of the OASIS controller
- Providing easy-to-use functions for positional control by employing a real-world co-ordinate system
- Organisation of functions according to application-specific tasks, such as stage control and automatic focus

The OASIS DLL manages fundamental controller interface tasks such as initialisation of the device driver, reads and writes to hardware control registers, and exchange of data with the on-board OASIS DSP.

Co-ordinate positions for each axis are maintained in microns, matching physical distances. The conversion from microns to actual controller micro- or half-steps is handled by the OASIS DLL and is transparent to the user.

The facilities found in the OASIS DLL are organised generally according to those characteristics of the microscope that may be automated:

- Motorised XY Stage
- Motorised Focus (Z-Axis)
- Automatic Focussing via Video Signal
- Extra Device Control (filter wheels, etc.)

In addition to these functional groups, the OASIS DLL provides functions for general hardware set-up and inquiry and general-purpose control of each axis separately.

System Requirements

To use the OASIS DLL, you will need the following:

- A Pentium or better computer running Windows 2000, Windows XP, or 32-bit Windows Vista
An OASIS-4i or OASIS-blue Controller installed into an available PCI slot
- Microsoft® Visual C++ 6.0 or higher
- The OASIS Installation CD-ROM

## Installation

The OASIS DLL Developer Kit is installed from CD-ROM. To install the Kit:

1. Insert the OASIS Installation CD into your system’s CD drive.
2. Using Windows Explorer, navigate to the “SDK” directory on the CD.
3. Start the “SETUP.EXE” application in the SDK directory on the CD.
4. Follow the instructions on the screen to specify where you want the files for the OASIS DLL Developer Kit copied on your system.

Note that to use the OASIS hardware, you need to have successfully installed the OASIS controller hardware. See the documentation for your OASIS hardware for more information on installing the controller board.

Also note that the option to “Install OASIS Tools” from the OASIS CD’s main menu will install the OASIS SDK onto your system.

## Files in the OASIS Library Developer Kit

The following files are copied to your system as part of the OASIS DLL Developer Kit.

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OASIS.EXE</td>
<td>Application to setup, test and demonstrate the facilities of the OASIS controller.</td>
</tr>
<tr>
<td>OIFLASHCFG.EXE</td>
<td>Application to setup the FLASH memory of the OASIS controller.</td>
</tr>
<tr>
<td>OASIS_DLL_MANUAL.PDF</td>
<td>The OASIS DLL Developer Kit manual, in Adobe’s PDF format.</td>
</tr>
<tr>
<td>OASIS4I.H</td>
<td>Header file that prototypes all the OASIS DLL control functions.</td>
</tr>
<tr>
<td>OI_CONST.H</td>
<td>Header file defining various constant values used by the OASIS DLL.</td>
</tr>
<tr>
<td>OASIS4I.LIB</td>
<td>Import library file for linking your application to the</td>
</tr>
<tr>
<td>File</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>example\OASISTST.DSP</td>
<td>Project file for the OASISTST MFC example application.</td>
</tr>
<tr>
<td>example\OASISTST.CPP</td>
<td>Source file for the OASISTST MFC example application.</td>
</tr>
<tr>
<td>example\OASISTST.H</td>
<td>Primary header file for the OASISTST MFC example application.</td>
</tr>
<tr>
<td>example\OASISTST.RC</td>
<td>Resource template for the OASISTST MFC example application.</td>
</tr>
<tr>
<td>example\RESOURCES.H</td>
<td>The resources header file defining the IDs used by the OASISTST MFC example application.</td>
</tr>
<tr>
<td>example\StdAfx.h</td>
<td>Standard system includes header for the OASISTST MFC example application.</td>
</tr>
<tr>
<td>example\StdAfx.cpp</td>
<td>Source file for standard precompiled headers.</td>
</tr>
<tr>
<td>example\Release\OASISTST.EXE</td>
<td>The OASISTST MFC example application executable.</td>
</tr>
</tbody>
</table>
Using the OASIS Software

Using the OASIS DLL in your Visual C++ project is relatively simple. Follow these steps:

1. Include the OASIS4I.H header file in the source files where you need to make calls to the OASIS automation controller.

2. Add the OASIS4I.LIB import library as an additional library in your projects Linker settings.

3. Ensure that the application calls the `OI_SetHardwareMode` and `OI_Open` functions before you make any calls that access the OASIS controller.

4. Ensure that you call the `OI_Close` function when your application is finished using the OASIS controller.

Figure 1 shows a basic example of a Win32 application created with Microsoft Visual C++ using the Win32 application wizard. The code to open the OASIS controller has been added. In this example, the stage is initialised once the controller is opened.
```c
#include "stdafx.h"
#include "oasis4i.h"

int WINAPI WinMain(HINSTANCE hInstance,
                   HINSTANCE hPrevInstance,
                   LPSTR     lpCmdLine,
                   int       nCmdShow)
{
    // Set the hardware mode to use the controller
    OI_SetHardwareMode(OI_OASIS);

    // Open the controller
    int iret = OI_Open();

    // if the open failed, exit
    if(OIFAILED(iret))
    {
        MessageBox(NULL,           
                    "Failed to open driver",  
                    "Error",                   
                    MB_OK );
        return 0;
    }

    // Now we can do what we like...

    // For instance, initialize the stage
    iret = MessageBox(NULL,       
                      "Make sure everything is clear of the stage travel!",  
                      "Initialize stage?",                
                      MB_OKCANCEL | MB_ICONWARNING );
    if( iret == IDOK)
        OI_InitializeXY();

    // We're finished, so close things down
    OI_Close();
    return 0;
}
```

Figure 1. Basic example using the OASIS DLL.

Though these simple steps allow you to start using the OASIS hardware, there are a number of important considerations relating to general automation techniques, as well as the details of how the OASIS DLL implements these. The following sections highlight the details relating to configuring the controller and performing movement operations.
Moving Axes and Components

The most fundamental application of the OASIS controller is of course to move one or more axes by driving motors. A number of factors arise when considering this movement, including:

♦ The calibration of the axis, where real-word distances are converted to basic motor steps
♦ The way the axis is accelerated and decelerated
♦ The cruising speed that defines the desired maximum rate of travel
♦ The assistance of any positional feedback mechanisms during the move
♦ The method by which an application can determine the status of the movement

Each of these factors must be considered in order to achieve the best performance of the motion control system and are described below.

Positional Units and Axis Calibration

The OASIS DLL manages the conversion of position values—i.e. based on microns—into internal microstep values using conversion factors that can be set up in the DLL.

All positions are specified in microns, thereby simplifying overall operation by allowing movements and readouts to be in the actual physical dimensions of the microscope stage and focus mechanisms.

Typically for XY stages, the lead screw pitch is used to determine the actual distance of travel for a given microstep. The `OI_SetPitchXY` function may be used to indicate the actual lead screw pitch of a given stage, at which time the OASIS DLL will calculate the corresponding microstep resolution.

For the Focus and F-Axis, `OI_SetAxisStepSize` may be used to indicate the actual distance of travel for each microstep. Also, these axes have complementary `OI_SetPitchZ` and `OI_SetPitchF` functions. For instance, you may wish to all `OI_SetPitchZ` with a pitch value of 0.1 mm, i.e., the typical 100 microns per fine focus revolution.

The Coordinate System

As mentioned above, the coordinate system for each axis is defined in units of microns. Each axis has a range of travel, which is defined by both negative and positive software limit values. In cases where hard limit switches are fitted, as with a motorised XY stage, an automatic initialisation may be performed to search for these limit switches in X and Y.

Figure 2 gives a graphical example of the physical situation. A motor is connected a lead screw which is used to convert the rotational motion of the motor into a translation of a device such as a XY stage. The physical, hard limit switches are found near the end of the physical limits of travel. Within that range are the software limits, defining the range in which the controller allows movement. An axis origin defines the 0 position value, to which all other
positions are referenced. In reality, the range of travel is broken down into a larger number of very fine steps, corresponding to the microstepping resolution of the motor controller.

![Diagram of physical and software limits with range of travel](image)

Figure 2. Physical and software limits, with range of travel.

## Moving a Single Axis

Once the axis calibration and range of motion are defined, you may go about moving the axis. For a general-purpose move, you may use the `OI_MoveAxis` function. For instance, the function call:

```c
OI_MoveAxis( OI_ZAXIS, 10.5, 1 )
```

moves the Z-axis to an absolute position of 10.5 microns. The last parameter, set to 1 in this example, tells the function to wait until the move is complete before returning.

You may also wish to move a relative distance from the current position, for which you may use the `OI_StepAxis` function. For instance, the function call:

```c
OI_StepAxis( OI_XAXIS, -600, 0 )
```

moves the X-axis 600 microns in the negative direction. The 0 passed as the last parameter tells the function to return immediately, without waiting until the move is complete.

## Waiting for Movement Completion

Each function that moves an axis contains a parameter `nWait` that indicates if the function should return immediately or should wait until the desired position is reached before returning.

Specifying an `nWait` value of 0 causes the function to return immediately (i.e., as soon as the move command is read and acknowledged by the controller), without waiting for the move to complete.

Specifying a non-zero `nWait` value will cause the function to poll the affected axes’ status until the move is complete before returning.
Note that at anytime during the wait for the move to complete, the user may press either the ESCAPE or CTRL-C keys to abort the movement. If either of these occurs, all axes are immediately halted (using the deceleration ramps so that position integrity is maintained) and the function will return with an OI_ABORT error code.

If you have called a movement function without waiting, but at some later time need to wait for the axis to stop, you may use the “OI_WaitForStopped” functions. These functions are:

- OI_WaitForStoppedXYZ( int xstop, int ystop, int zstop )
- OI_WaitForStoppedXY()
- OI_WaitForStoppedZ()
- OI_WaitForStoppedF()

and each deal with a given axis or set of axes.

Alternatively, you may use one of the “OI_ReadStatus” functions, such as OI_ReadAxisStatus, OI_ReadStatusXY, etc. to read the status of a given axis and check the S_MOVING status bit to see if the axis is currently in motion.

---

**Moving the XY Stage and Focus**

In many cases, the OASIS controller is used in a 3-axis scenario corresponding the XY stage and Z focus drive of a microscope. The OASIS DLL offers a set of functions organised around these logical components, i.e., “XY” functions for two-axis operation of the stage, “Z” functions for the focus, and “XYZ” functions that deal with all three axes simultaneously.

---

**XY Stage Initialisation**

To automatically initialise the stage, call:

- OI_InitializeXY()

which causes the automatic limit search to begin. The function first moves in the negative direction until the negative limits are detected for both axes. Then the position limits are found by a move in the positive direction. Once both limits are found, the stage moves to the centre of the range of travel. The negative physical limit is set to a position of XY=[0,0], and the software limits are defined to be just inside the physical limits (to prevent loss of position during open loop movements if the axis were driven into the hard limit).

**Caution:** Please ensure that the full range of stage travel is free of all optical and mechanical obstructions—such as objective lenses and the condenser optical system—prior to calling OI_InitializeXY!

You may manually define the software limits of the stage by calling:

- OI_SetUserLimitsXY( double dXMin, double dXMax, double dYMin, double dYMax )

where dXMin and dXMax define the negative and positive soft limits for the X axis, and dYMin and dYMax define the corresponding Y-axis values.

To clear the software limits, call:

- OI_ClearUserLimitsXY
This is disable the software limits, i.e., only the physical limit switches will be used to limit travel.

You may also redefine the origin location with a call to either:

\[ \text{OI\_SetOriginXY()} \]

or

\[ \text{OI\_SetPositionXY( double XPos, double YPos )} \]

The \text{OI\_SetOriginXY} function defines the current position as the origin, i.e., \( XY=[0,0] \). The \text{OI\_SetPosition} function defines the current position to be the specified \( XY=[XPos, YPos] \), and the coordinate system origin is adjusted to accommodate the new position. In each case the software limits are modified in order to maintain the same physical locations.

**XY Stage Movement**

Once you’ve established the range of stage travel, you may begin moving the stage around. For instance, function call:

\[ \text{OI\_MoveToXY( 1000, 5000, 1 )} \]

moves to an absolute XY position of \( X=1000, Y=5000 \). The last parameter of 1 indicates the function should wait until the move is completed before returning.

To move the stage relative distance from the current location, you can call:

\[ \text{OI\_StepX( double dXDistance, int nWait )} \]
\[ \text{OI\_StepY( double dYDistance, int nWait )} \]

or

\[ \text{OI\_StepXY( double dXDistance, double dYDistance, int Wait )} \]

The first two functions step either the X or the Y axis, respectively, while the third variation steps both the X and Y axis simultaneously.

Many automated microscopy applications that use motorised scanning require 3-axis control for XY stage and focus manipulation. You may perform simultaneous 3-axis moves using calls such as:

\[ \text{OI\_MoveToXYZ( 1000, 1500, 10, 0 )} \]

which moves to an XYZ location of \( X=1000, Y=1500, \) and \( Z=10 \). The 0 passed in the last parameter indicates the function should return immediately.

Figure 3 shows a very simple rectangular raster scanning example. In the example the current stage position is read, and a 10x10 field scan is made from that location, using a step size for both X and Y of 500 microns.
```c
void ScanRectangle(void)
{
    // Setup a 10x10 field scan
    // fields are 500 microns apart
    int nXFields = 10;
    int nYFields = 10;
    double dXStep = 500;
    double dYStep = 500;

    // Read the current XY position
    // as the starting point
    double dXStart, dYStart;
    double dXPos, dYPos;
    OI_ReadXY( &dXStart, &dYStart );

    // Now do the scanning
    for( int nY=0; nY<nYFields; nY++ )
    {
        // Calc Y position of field
        dYPos = dYStart + nY*dYStep;
        for( int nX=0; nX<nXFields; nX++ )
        {
            // Calc X position of field
            dXPos = dXStart + nX*dXStep;

            // Move to the field
            OI_MoveToXY( dXPos, dYPos, 1 );
        }
    }

    // Finally, move back to start position
    OI_MoveToXY( dXStart, dYStart, 0 );
}
```

Figure 3. Simple XY rectangular scanning example.

You may imagine variations on this scanning theme were every other row is scanned retrograde for a serpentine movement, etc.

**Driving the Stage Continuously**

In some instances you may wish to set the stage moving at a given direction, without a particular target destination. One example that uses this behaviour is a software joystick, where the stage should be driven in the direction of joystick deflection, until, say, the user releases a mouse button.

To drive the stage continuously, use a call to

```
OI_DriveContinuousXY( int nXSpeed, int nYSpeed )
```

where `nXSpeed` and `nYSpeed` give the speed, in half-steps per second, at which to drive the X and Y axes, respectively. These speed values range from \(-4096\) half-steps / sec to \(+4096\) half-steps / sec, where the value's sign indicates the direction of travel.
To stop the moving stage, call

\[ OI_{\text{HaltXY}}() \]

which will stop the X and Y axes if they are moving.

**Z Focus Initialisation**

Like the XY stage functions, a number of functions allow control of the Z focus drive. These functions have a “Z” suffix, such as:

\[ OI_{\text{SetOriginZ}}() \]

which sets the current position of the Z axis to be the origin, i.e., \( Z=0 \).

Like all axes, the Z focus maintains negative and positive software limits that define the range of travel. However, focus drives normally do not contain limit switches, and therefore automatic initialisation is not possible. To initialise the Z focus to a known position and range, call:

\[ OI_{\text{InitializeZ}}(\text{double } Z\text{RangeAbove}, \text{double } Z\text{RangeBelow}) \]

The \( OI_{\text{InitializeZ}} \) function does the following:

1. Sets the current position to the origin \( (Z=0) \);
2. Sets the positive software limit a distance of \( Z\text{RangeAbove} \) microns above the current position;
3. Sets the negative software limit a distance of \( Z\text{RangeBelow} \) microns below the current position.

The call should be made when the specimen is nominally in focus, and once initialised, the Z focus coordinate system and range of travel will be defined as is shown in Figure 4.

![Focus Initialisation Diagram](image)

**Figure 4.** Focus Initialisation.

This configuration, where the focus origin is considered the nominal in-focus position and specific ranges of travel are defined above and below that, works well for microscopy applications, where consideration must be given to prevent large movements that may damage the specimen or the optical system.
For instance, on a typical microscope, movement of the focus mechanism moves the entire stage and condenser sub-system together, towards or away from the objective lens. In such a configuration, larger moves in the negative direction, away from the objective, are possible, while movements towards the objective, where the small working distances of the objective lens are in effect, lead to a much smaller range of safe travel.

In order to further protect against large movements that may damage the optical system components of a microscope, the OASIS controller also uses a “Maximum Move” value, which is a microstep value the DSP uses to reject larger move requests. The actual Maximum Move value is set in the Flash memory, but you can read the current value for a given axis by calling:

```c
OI_GetAxisMaxMove( int AxisID, LPDWORD lpdwValue )
```

This method helps prevent physical collisions when for instance in a situation where the software limits have not been properly set and a very large move has been called. For example, if the Z axis has not been initialised and is in an unknown state. A call to move to an absolute position may in such a case result in a very large movement, potentially causing damage to the specimen or the optical system. If such a move is beyond the Maximum Move value, the DSP refuses the move, i.e., the axis is not driven at all.

**Z Focus Movement**

To move the Z-axis to an absolute position, call:

```c
OI_MoveToZ( double Z, int nWait )
```

where \( Z \) gives the Z-axis position and \( nWait \) tells the function whether to wait until the position is achieved before returning. You can do a relative move with:

```c
OI_StepZ( double ZDistance, int nWait )
```

For example, Figure 5 shows an example user function that steps the focus through a given range. The `OI_StepZ` function is used first to move to one end of the range, then the specified number of steps is taken through to the end of the range. At each Z position, the application could do some processing, such as acquiring successive Z images to create a focus stack for an extended focus calculation.
void StepFocus( double dRange, int nSteps )
{
    // get the size of each step
    double dStepSize = dRange / nSteps;

    // Move half range from current pos
    OI_StepZ( -dRange/2, 1 );

    // Now step through range
    for( int i=0; i<nSteps; i++ )
    {
        // Could do something here
        // e.g., acquire an image

        // now step to next position
        OI_StepZ( dStepSize, 1 );
    }
}

Figure 5.  OI_StepZ example.

---

**Acceleration Tables**

Typically, movement is performed by accelerating a stationary axis to some top speed, then decelerating as the destination position is approached so that the axis is stopped at the target position. Consider for instance Figure 6.
Figure 6 shows a linear acceleration profile. Note the velocity increases linearly until a given cruising speed is reached. Once the target destination approaches, the axis is decelerated in a similar fashion in order to stop at the desired location.

The OASIS controller uses four pre-defined ramp profile lookup tables to define acceleration and deceleration. Each table consists of 512 values indicating a timer value and step size. By default, these tables are configured for Slow, Normal, Fast, and User-defined acceleration profiles, and are normally referred to using these names.

Figure 7 shows an example ramp table, showing a sequence of microsecond timer intervals and step sizes. The acceleration is performed by running through successive indices in the table after stepping the indicated steps size and waiting the corresponding timer interval.

In the Figure 7 example, note the transition from 1 microstep to 2 microsteps that occurs at index 3, and the corresponding increase in the timer interval to compensate. This is required in order to keep the timer interval above a 200 microsecond value, which allows for simultaneous servicing of up to four axes by the DSP.

<table>
<thead>
<tr>
<th>Index</th>
<th>Timer Interval (microseconds)</th>
<th>Step Size (microsteps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>244</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>227</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>212</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>397</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>374</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>354</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>335</td>
<td>2</td>
</tr>
</tbody>
</table>
Selecting the Table

Each axis uses one of the four pre-defined tables for its acceleration profile. To set which of the four acceleration tables to use for a given axis, use the `OI_SetAxisRamp` function, for instance, the following function call:

`OI_SetAxisRamp( OI_ZAXIS, OI_RAMP_SLOW )`

sets the Z-axis ramp to use the slow table. You may also use the ramp definition for each component, such as `OI_SetRampXY` or `OI_SetRampZ`.

The default ramp table for each axis is stored in the Flash memory, and may be modified using the Flash configuration application.

Defining the Table

Each of the four ramp tables are stored in the Flash memory of the OASIS controller. The Flash configuration application allows you to calculate new linear and S-curve tables, or specify your own individual table values, and save them to any of these locations. See the documentation for the Flash memory configuration application for more information on defining your own ramp tables.

Cruising Speed

The cruising speed defines the maximum speed at which a given axis will be driven. The OASIS controller allows you to specify the desired cruise speed separately for each axis. The cruise speed is a value between 0 and 511, which corresponds to the desired maximum index to ramp to in the acceleration ramp table.

For example, a cruise speed of 300 means that the controller will ramp up to index 300 in the acceleration ramp table, then continue to drive at the rate found at index 300 until deceleration is required near the final destination.

To set the cruise speed for a given axis call:

`OI_SetAxisCruise( int AxisID, int nCruise )`

For instance, the following call:

`OI_SetAxisCruise( OI_ZAXIS, 200 )`

sets the Z axis cruise speed to 200.

You may also set the cruise speed using the logical components such as the XY stage and Z focus functions:

`OI_SetCruiseXY( int nXCruise, int nYCruise )`

`OI_SetCruiseZ( int nZCruise )`
The default cruise speed for each axis is stored in the Flash memory, and may be modified using the Flash configuration application.

**Encoder Support**

Encoders are position-sensing devices that provide feedback that indicates movement of a sensor. Encoders may be fitted to a given axis in order to provide an independent feedback mechanism which may be used to sense manual movements of the axis (for instance if the stage hardware permits turning by hand) and also may be used during movements to ensure accuracy of positioning.

The OASIS controller accepts encoder inputs and may be configured to use these to perform such closed-loop operations. The setup of the encoder parameters is accomplished in the Flash memory configuration application. Please refer to that application for further details on properly configuring the controller for closed-loop operation using encoders.

You may read if an encoder has been configured for an axis with a call to:

```
OI_GetAxisEncoderFitted( int AxisID, LPBOOL lpbFitted )
```

To read the configured encoder step size

```
OI_GetAxisEncoderStepSize( int AxisID, double *pdMicrons )
```

Note that these functions retrieve the settings as currently defined in the Flash memory and do not necessarily indicate that the encoder inputs have been sensed by the hardware, but instead depend on an accurate configuration in the Flash memory.

The encoder step size is actually calculated by looking that the microstep to encoder step ratio, as defined in the Flash memory. For accurate stepping, it is important to ensure the microstepping resolution is some multiple of the encoder resolution, and the Flash memory configuration application allows you to select from various microstepping resolutions in order to achieve the appropriate ratio for a given encoder.

For instance, if an encoder with 0.1 micron resolution is fitted to an axis with a 2 millimetre pitch leadscrew, then the microstepping resolution should be set to 40,000 steps per revolution to ensure a 2:1 ratio of microsteps to encoder inputs. The `OI_GetAxisEncoderStepSize` function uses the encoder to microstepping factor and the current microstepping resolution and step size in order to return the resulting encoder resolution.

**Enabling Encoder Inputs**

A secondary counter in the OASIS controller, maintained in addition to the normal microstepping position counter, manages encoder inputs. Closed-loop operation is achieved when the OASIS controller uses the encoder input counter to correct the position information maintained by the microstepping counter.

If an encoder has been configured for an axis, the use of the encoder input signals may be enabled or disabled via software. The function:

```
OI_SetAxisEncoderEnabled( int AxisID, BOOL bEnabled, BOOL bAutoCorrect )
```
is used to enable or disable the use of the encoder input counter, as set by the \textit{bEnabled} parameter. The second \textit{bAutoCorrect} parameter indicates that all movements on that axis should be corrected based on the encoder feedback.

To determine the status of encoder use for a given axis, call:

\begin{verbatim}
OI_GetAxisEncoderEnabled( int AxisID, LPBOOL lpbEnabled, LPBOOL lpbAutoCorrect )
\end{verbatim}

When encoder inputs are enabled, all position readouts are given based on the encoder input counter. Therefore the position information is given by the encoder resolution rather than the microstepping resolution. For instance, if a 2 mm pitch axis is configured for 40,000 microsteps per rev and a 0.1 micron encoder is also fitted and enabled, the position values will be provided to the nearest 0.1 micron, rather than the 0.05 micron resolution of the microstepping.

For the X, Y, and Z axes, you can specify whether the encoders are used to perform closed-loop position maintenance. In closed-loop mode, the OASIS controller uses the encoder feedback to ensure that movements are made to within a specified tolerance. Also, the controller will "serve" the current position, using the encoder signals to ensure that the current position is not changed by any external forces (other than controller movement commands or joystick-type of inputs).

To enable closed-loop operation, use the functions:

\begin{verbatim}
OI_SetEncoderEnabledXY( BOOL bEnabledX, int nTolX, BOOL bEnabledX, int nTolX )
OI_SetEncoderEnabledZ( BOOL bEnabledZ, int nTolZ )
\end{verbatim}

These functions allow you to enable the use of the encoder counters as well as specify the counter tolerance over which the controller's servo mode takes effect.

\section*{Reading the Microstepping Resolution}

The microstepping resolution for each axis is normally 12,800 microsteps per rev, but may be modified to other values in the Flash memory. This is done by using the extended Sine-Cosine lookup tables, which is done in by the Flash memory configuration application. The extended Sine-Cosine LUTs may be configured to give 10,000, 20,000, 40,000 or 50,000 microsteps per revolution. The Flash memory provides for two extended Sine-Cosine LUTs in addition to the 12,800 steps / rev default LUTs for each axis. The actual LUT in use for each axis is configured in the Flash memory as well.

To read the current microstepping resolution for a given axis, call:

\begin{verbatim}
OI_GetAxisStepsPerRev( int AxisID, LPDWORD lpdwStepsPerRev )
\end{verbatim}

For more information on configuring the microstepping resolution, please refer to the Flash memory configuration application.
Saving Settings

Many fundamental settings of the controller are stored in the Flash memory and loaded immediately once the OASIS initialises upon PC power on. These Flash settings ensure the board is functional prior to any application software usage, for instance allowing immediate trackball or joystick operation of the motorised hardware.

However, your application may call various DLL functions to modify settings, such as adjusting the cruise speed for an axis. Also, the DLL maintains some values that are not stored in the flash memory, such as the currently defined origin for an axis. In order to allow an application to easily store and retrieve these values, the OASIS DLL offers functions for writing and reading the settings and position information to disk.

Typically you would load settings and positions just after calling `OI_Open` to open the controller. Loading settings prior to a call to `OI_Open` will have no effect on most settings because they are maintained in the DSP of the controller. Similarly, you will want to make a call to save settings before you call `OI_Close`, because many settings are taken from the current DSP values and therefore require the hardware driver to be open.

---

Saving and Loading Settings

To save the current DLL system settings, call:

```c
OI_SaveSettings( LPCTSTR sFile )
```

The `sFile` parameter indicates the name a file to use to hold the settings. The file does not necessarily need to exist, and after the call the results will be stored in a typical Windows INI file format.

You may also pass an empty string to the `OI_SaveSettings` function, in which case the settings are stored into the Windows Registry. For instance, the call:

```c
OI_SaveSettings("")
```

will cause the DLL settings to be stored into the Registry. This alleviates the application from concerns about the path to configuration files, accidental deleting of settings files, etc. However, the ability to optionally save the settings to a named file is handy for easily creating backups of current settings or for offering multiple configurations based on various previously saved files.

To restore the settings, call:

```c
OI_LoadSettings( LPCTSTR sFile )
```

where `sFile` is the name of a previously stored settings file. You may also pass an empty string to restore settings from the Registry.
Saving and Loading Positions

The position counters of the DSP are maintained as long as the board has power and has not been reset. However, this information is lost when the host PC is powered off. The DLL offers functions for saving and reloading the position values, including the current position, the origin definition, and the software limits, for the axes.

To save the position information, call:

```c
OI_SavePositions( LPCTSTR sFile )
```

where `sFile` is the name of the file used to hold the settings. The format of the file is a typical Windows INI file. You may also pass an empty string to the function, in which case the position information is stored to the Registry.

To restore the position information, call:

```c
OI_LoadPositions( LPCTSTR sFile )
```

You provide either the name of a previously stored position file or an empty string to reload settings from the Registry.

Return Values

Each OASIS function returns an integer value indicating the success or failure of the function.

A value of `OI_OK` indicates the function was acknowledged and accepted by the control hardware and completed successfully. Non-zero return values indicated failure. A variety of conditions could lead to command failure, including lack of the required hardware device, timeout, or user abort.

The following table lists the return codes defined in the `OI_CONST.H` header file:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>OI_OK</td>
<td>0</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>OI_FAILED</td>
<td>1</td>
<td>The operation failed.</td>
</tr>
<tr>
<td>OI_ABORT</td>
<td>2</td>
<td>The operation failed due to a user abort.</td>
</tr>
<tr>
<td>OI_NOHARDWARE</td>
<td>4</td>
<td>The operation failed because the required hardware is not fitted.</td>
</tr>
<tr>
<td>OI_TIMEOUT</td>
<td>8</td>
<td>The operation failed due to a timeout.</td>
</tr>
<tr>
<td>OI_INVALIDARG</td>
<td>16</td>
<td>An invalid argument was passed into the function.</td>
</tr>
<tr>
<td>OI_HARDWAREBUSY</td>
<td>32</td>
<td>The command could not be executed because the hardware was busy with another request.</td>
</tr>
<tr>
<td>--------------------</td>
<td>----</td>
<td>-------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>OI_ACCESSDENIED</td>
<td>64</td>
<td>Access to the desired functionality is not available.</td>
</tr>
<tr>
<td>OI_NOTSUPPORTED</td>
<td>128</td>
<td>The operation is not supported by the current hardware platform.</td>
</tr>
<tr>
<td>OI_ILLEGALAXIS</td>
<td>256</td>
<td>The operation attempted to use an axis that is not available.</td>
</tr>
<tr>
<td>OI_INVALIDCONFIG</td>
<td>512</td>
<td>The requested configuration is not allowed.</td>
</tr>
<tr>
<td>OI_MAXMOVEFAIL</td>
<td>1024</td>
<td>A move command failed because it exceeded the maximum allowed move for the specified axis, as defined in the flash memory.</td>
</tr>
</tbody>
</table>
Advanced Topics

Using Multiple OASIS Controllers

Versions 2.02 and later of the OASIS DLL provide support for up to four OASIS controllers in a single computer. This potentially gives four times the capabilities of a single controller, e.g., up to 16 independent axes, four separate video processors, etc. Special considerations are made to ensure a straightforward interface as well as backwards compatibility with single-board operation.

Counting the Number of Installed OASIS Controllers

Use the \texttt{OI\_CountCards} function to determine how many OASIS cards are installed in a system:

\begin{verbatim}
OI_CountCards( int* pnNumber )
\end{verbatim}

The single argument returns the count.

Routing Commands to a Controller

The OASIS DLL uses a “routing” method to send commands to a particular controller. The selected controller is known as the “active card”, i.e., the OASIS card that will be target for the API calls to perform various operations.

To set the active card, call:

\begin{verbatim}
OI_SelectCard( int nCard )
\end{verbatim}

The \texttt{nCard} argument is a zero-based index of the card, e.g., for \textit{N} installed cards, the first card has an index of 0, the last card has an index of \textit{N}-1.

To determine which card is currently active, call:

\begin{verbatim}
OI_GetSelectedCard( int* pnCard )
\end{verbatim}

Except in the case where general axis drive commands (such as the \texttt{OI\_MoveAxis} and \texttt{OI\_ReadAxis} functions), once a card is selected, all the API functions will be routed to that controller.

For instance, consider the instance where 2 controllers are installed in a system. Say that an XY stage and Z focus are connected to the first controller, while two stepper motors for specimen handling have been attached to the X and Y axes on the second controller. Figure 8 shows how movement commands would be routed for each board separately.
void MoveExampleTwoCards()
{
    // move to xy = 1000, 1000 on first card
    // then do an autofocus
    OI_SelectCard( 0 );
    OI_MoveToXY( 1000, 1000, 1 );
    OI_AutoFocus();
    OI_WaitForAutoFocus();

    // move to xy = 50, 50 on the second card
    OI_SelectCard( 1 );
    OI_MoveToXY( 50, 50, 1 );
}

Figure 8. Multiple Card Example

Note that since the `OI_SelectCard` function acts globally, actions must be performed serially for each controller. You must ensure that all your actions are complete on one controller before switching to another one for further commands. However, parallel access to multiple controllers is possible using the general axis commands.

### Using General Axis Commands with Multiple Controllers

The “general axis” commands provide additional functionality beyond the basic command routing given by the `OI_SelectCard` function. See “Table 3. API Function Categories” and the “General, Single Axis Control” section below for more information about the general axis functions.

Each of the general axis functions takes an Axis ID as an input parameter. For instance, for a general read of an axis’ position, you may call:

```
OI_ReadAxis(int AxisID, double* pdVdalue)
```

The AxisID argument is typically a value ranging from 1 to 4, indicating which axis is to be read.

However, in the case of multiple controllers, the valid AxisID values range from 1 to 4*N, where N is the number of controllers installed. For example, if three OASIS controllers are present, then the total number of available axes are 4*3 = 12 axes. The `OI_GetTotalAxisCount` function returns the maximum number of axes available for the current configuration:

```
OI_GetTotalAxisCount( int* pnAxisCount )
```

By default, you may pass AxisID values up to the total available count and the OASIS DLL will automatically route the command to the correct controller.

Note that this is independent of the currently active card as set by the `OI_SelectCard` function. This provides additional functionality beyond the global command routing provided by `OI_SelectCard`.

To ensure compatibility with existing applications and to enable the ability to use the command routing
Ol_SetMultiAxisMode( int nMode )

Ol_GetMultiAxisMode( int* pnMode )

where the nMode argument is either of two values indicating the desired functionality. See Table 1 below for a description of the constants.

<table>
<thead>
<tr>
<th>nMode</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OI_MULTI_MODE_ID</td>
<td>0</td>
<td>Use 1 to 4*N axes, overrides OI_SelectCard setting (default).</td>
</tr>
<tr>
<td>OI_MULTI_MODE_ROUTE</td>
<td>1</td>
<td>Use 1 to 4 axes, routed to a particular card via the OI_SelectCard setting</td>
</tr>
</tbody>
</table>

Table 1. General Axis Routing Constants

**Special Considerations When Using Multiple Controllers**

Care must be taken when using the global command routing provided by the OI_SelectCard function, particularly in multi-threaded situations where one area of code may set the active card using OI_SelectCard, while another area of code is in the middle of a set of operations on a particular board. Since OI_SelectCard works globally, it affects all subsequent calls, with the exception of general axis moves routed via the AxisID.

It is recommended to use the general axis commands wherever possible. When using the OI_MULTI_MODE_ID mode of axis specification, these commands allow multiple cards to be accessed simultaneously, in parallel, without consideration to which board is currently selected for command routing using OI_SelectCard.

**General Purpose I/O**

The OASIS controller provides a number of ports that may be used for general purpose input and/or output facilities. In particular, the OASIS controller offers:

<table>
<thead>
<tr>
<th>Port</th>
<th>Number Available</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>General I/O</td>
<td>2</td>
<td>TTL compatible Input and Output</td>
</tr>
<tr>
<td>Open Collector</td>
<td>2</td>
<td>Output only</td>
</tr>
<tr>
<td>PL4 Input Ports</td>
<td>4</td>
<td>Input only</td>
</tr>
</tbody>
</table>

Table 2. OASIS I/O Capabilities

These ports may be used to control or trigger external devices or to sense the status of switches, etc.
Three API functions provide access to these inputs and outputs. The following functions correspond to the 2 General I/O and 2 Open Collector ports:

- OI_WriteIO( BYTE byVal )
- OI_ReadIO( LPBYTE lpbyVal )

The logic values are set and read using bits in the BYTE argument.

The input ports found on connector PL4 are read using the function:

- OI_ReadInputPorts( LPBYTE lpbyVal )

with each input corresponding to a bit in the BYTE argument.
Function Descriptions

The facilities of the OASIS DLL are organised below into functional groups. These groups are:

<table>
<thead>
<tr>
<th>Group</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hardware Control</td>
<td>These functions deal with the overall initialisation, communication, and status of the OASIS hardware.</td>
</tr>
<tr>
<td>Version Information</td>
<td>These functions return version information about the OASIS hardware and the OASIS DLL.</td>
</tr>
<tr>
<td>General, Single Axis Control</td>
<td>Though many of the functions in the OASIS DLL are organized according to the physical components of a microscope system (such as stage and focus), the DLL also offers a representation of the controller in which each axis may be accessed individually. This section lists those functions</td>
</tr>
<tr>
<td>Simultaneous Three Axis Control</td>
<td>Many automated microscopy applications use 2 primary motorized components: The XY stage and the Focus mechanism. This leads to a 3-dimensional, “X-Y-Z”, class of functions that are described in this section.</td>
</tr>
<tr>
<td>XY Stage Control</td>
<td>These functions deal exclusively with the two axes of a motorized XY stage.</td>
</tr>
<tr>
<td>Z / Focus Control</td>
<td>These functions deal exclusively with the single, Z-axis of a motorized focus mechanism.</td>
</tr>
<tr>
<td>F-Axis Control</td>
<td>These functions deal exclusively with the single fourth, or “F”, axis available in the OASIS controller.</td>
</tr>
<tr>
<td>Encoder and Closed-loop</td>
<td>These functions relate to the use of encoder feedback for position information and closed-loop operation.</td>
</tr>
<tr>
<td>Automatic Focus</td>
<td>When an OASIS-AF hardware module is fitted, the OASIS controller provides facilities for automatically focusing a specimen. This section describes the functions used for AutoFocus.</td>
</tr>
<tr>
<td>Predictive Focus</td>
<td>These functions allow you to define and use predictive focusing, where the focus is automatically adjusted based on orientation of the plane of the sample relative to the objective lens.</td>
</tr>
<tr>
<td><strong>Video and Digital Camera Interface</strong></td>
<td>When an OASIS-AF hardware module is fitted, the OASIS controller provides real-time measurements of the total area and maximum chord length of detected features in the video signal. This section describes the functions for setting up and reading out these measurements.</td>
</tr>
<tr>
<td><strong>Filter Changer Functions</strong></td>
<td>The functions provide an interface for controlling filter changer devices, such as a rotating filter wheel.</td>
</tr>
<tr>
<td><strong>Hardware Joystick / Trackball Functions</strong></td>
<td>If hardware XY and/or Z axis joystick is fitted to the OASIS controller, these functions may be used to enable its operation.</td>
</tr>
<tr>
<td><strong>Timeout Functions</strong></td>
<td>This section describes the functions used to specify the timeout durations used for movement, automatic focus, and video functions.</td>
</tr>
<tr>
<td><strong>File I/O Functions</strong></td>
<td>These functions are used to save and restore system settings to and from file.</td>
</tr>
<tr>
<td><strong>Error Handling</strong></td>
<td>The OASIS DLL maintains extra information about errors when they occur. These functions are used to obtain this error information and also specify how general errors are to be reported.</td>
</tr>
<tr>
<td><strong>Micron / Step Conversion Functions</strong></td>
<td>In some instances an application may need to convert from the native micron-based coordinate system of the controller to the low-level microstep values of the OASIS DSP. Several functions used to perform these conversions are listed here.</td>
</tr>
<tr>
<td><strong>General Purpose Input / Output</strong></td>
<td>Access to the general I/O hardware of the OASIS controller.</td>
</tr>
</tbody>
</table>

Table 3. API Function Categories

---

**Hardware Control**

The hardware control functions deal with the basic initialisation and setup of the controller/host communication. They can be used to get information about the status of various facilities of the controller, such as the status of the motor voltage supply, whether an autofocus module is fitted, or that the Flash memory is properly configured (and not corrupt), to name a few.

These functions also allow you to place the controller in various modes of operation, such as full hardware communication or simulated operations. You can also configure some components, such as the Z-drive and the filter changer, to use a different controller such as an integrated automated microscope rather than the OASIS controller.
**OI_Close**

**Syntax**

OI_API OI_Close(void)

**Description**

Closes the OASIS driver for the current session.

**Parameters**

None.

**Return Value**

This function always returns OI_OK.

**Comments**

An application that has opened the OASIS hardware driver using a call to **OI_Open** should call the **OI_Close** function before terminating.

**See Also**

**OI_Open**

---

**OI_CloseComponent**

**Syntax**

OI_API OI_CloseComponent(int nComponent)

**Description**

Closes a component.

**Parameters**

*nComponent*  
The component ID.

**Return Value**

OI_OK if successful.

**Comments**

Currently, only two components, the Z focus and the filter changer, support configurations to controllers other than the OASIS system. These components are specified by the following values:

<table>
<thead>
<tr>
<th><em>nComponent</em> value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>OI_CFG_FOCUS</td>
<td>The Z-axis focus control. All functions dealing the Z-axis are routed to the specified controller.</td>
</tr>
<tr>
<td>OI_CFG_FILTER</td>
<td>The filter changer. All filter changer functions are routed to the specified controller.</td>
</tr>
</tbody>
</table>

Note that normally, the component will be closed automatically when the **OI_Close** function is called, so that a call to **OI_CloseComponent** is not needed.
**OI_Configure**

**Syntax**

```c
OI_API OI_Configure( int nComponent, int nControl )
```

**Description**

Configures a component for a specific controller.

**Parameters**

- `nComponent` The component to be configured, as defined in the comments below.
- `pnControl` The type of controller to use for the given component. Note that there are limitations on which components support a given controller.

**Return Value**

- `OI_OK` if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

Version 2.0 of the OASIS DLL extends the concept of components, like the XY stage, Z focus, and the autofocus to include the filter changer. Also included is support for configuring some components to use a secondary controller found in the Leica Microsystems DMR range of automated microscopes, where the Z-drive and filter changer automation are built into the microscope stand itself.

The OASIS DLL provides for manipulating the Leica Microsystems DMR Z-drive and filter changer via the same functions used for the OASIS controller. Currently, only the Z-axis and filter changer allow configuration for control via the DMR microscope, and by default these components are configured to use the OASIS controller.

<table>
<thead>
<tr>
<th><code>nComponent</code> value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>OI_CFG_FOCUS</td>
<td>The Z-axis focus control. All functions dealing the Z-axis are routed to the specified controller.</td>
</tr>
<tr>
<td>OI_CFG_FILTER</td>
<td>The filter changer. All filter changer functions are routed to the specified controller.</td>
</tr>
</tbody>
</table>

Note that the Autofocus system will also be affected by the selection of the Focus component's controller.

The controller device to use for these components may be one of the following values:

See Also: OI_Close, OI_Configure, OI_OpenComponent
### OI_CountCards

**Syntax**

```
OI_API OI_CountCards( int* pnFound )
```

**Description**
Retrieves the number of OASIS controllers installed in the system.

**Parameters**
- `pnFound` Returns the number of cards found.

**Return Value**
- OI_OK if successful.
- OI_NOHARDWARE if an OASIS board is not found.

**Comments**
Version 2.02 of the OASIS DLL adds support for multiple OASIS cards. Use the `OI_CountCards` function to determine the number of cards that have been installed.

Use the `OI_SelectCard` to set which card is the target for API commands.

**See Also**
- `OI_SelectCard`, `OI_GetSelectedCard`
**OI_EmergencyStopAll**

Syntax: `OI_API OI_EmergencyStopAll(void)`

Description: Immediately stops all axes.

Parameters: None.

Return Value: OI_OK if successful.

Comments: This function does not use deceleration ramps and therefore could cause loss of positional accuracy. The OASIS hardware will clear the Initialised bits for all axes to reflect this.

See Also: OI_HaltAllAxes, OI_HaltXY, OI_HaltZ, OI_HaltF

---

**OI_EnableMotorPower**

Syntax: `OI_API OI_EnableMotorPower(BOOL bXYEnabled, BOOL bZFEnabled)`

Description: Enables or disable power to a given set of motors.

Parameters:
- `bXYEnabled`: Flag indicating whether power to the X and Y axes is to be enabled.
- `bZFEnabled`: Flag indicating whether power to the Z and F axes is to be enabled.

Return Value: This function returns OI_OK.

Comments: Motor power can be enabled and disabled via software, but must be done for the pair of axes XY or ZF.

---

**OI_EnableMotorPowerEx**

Syntax: `OI_API OI_EnableMotorPowerEx (LPWORD pwMotors)`

Description: Enables and disables motor power for the axes.

Parameters: `pwMotors`: Array of motors to enable/disable.

Return Value: OI_OK if successful.

If unsuccessful, a combination of error codes may be returned to indicate the
reason for failure.

**Comments**
Use the `OI_EnableMotorPowerEx` to turn on or off the motor power for a given motor. The input array values to 1 to enable, 0 (zero) to disable. The index assignments are as follows:

<table>
<thead>
<tr>
<th>Index</th>
<th>Axis</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
</tr>
<tr>
<td>1</td>
<td>Y</td>
</tr>
<tr>
<td>2</td>
<td>Z</td>
</tr>
<tr>
<td>3</td>
<td>F</td>
</tr>
</tbody>
</table>

See Also `OI_EnableMotorPower`

---

**OI_GetAFFitted**

**Syntax**

```c
OI_API OI_GetAFFitted(BOOL* pbFitted)
```

**Description**
Retrieves the hardware status indicating the presence or absence of the OASIS-AF video board.

**Parameters**

- `pbFitted` Returns TRUE if the OASIS-AF module is fitted. This parameter is set to FALSE otherwise.

**Return Value**
This function returns `OI_OK`.

**Comments**
The `OI_GetAFFitted` functions can be used to determine whether the OASIS-AF module is fitted to the system.

See Also `OI_SetAutoFocusHWMode`, `OI_GetAutoFocusHWMode`, `OI_ReadPCBStatus`

---

**OI_GetAHMDelay**

**Syntax**

```c
OI_API OI_GetAHMDelay (LPDWORD pdwMSecs )
```

**Description**
Retrieves the base delay used when using Leica AHM components.

**Parameters**

- `pdwMSecs` Returned base delay for Leica AHM-related calls.
OI_GetAutoFocusHWMode

Syntax:  
OI_API OI_GetAutoFocusHWMode(int* pnMode)

Description:  
Retrieves the current mode of operation for the automatic focus, either hardware access or simulated operation.

Parameters:
- pnMode: Pointer to value Parameter indicating the hardware status.
  - OI_OASIS (a value of 1), or
  - OI_SIM (a value of 0).

Return Value:  
OI_OK if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

Comments:  
See the Comments for the OI_SetAutoFocusHWMode function for more information regarding the OASIS-AF automatic focus settings.

See Also:  
OI_SetAutoFocusHWMode

OI_GetCardAxisCount

Syntax:  
OI_API OI_GetCardAxisCount (int nCard, int* pnAxisCount )

Description:  
Returns the number of available axes on a given card in the system.

Parameters:
- nCard: Zero-based index of the card in the system.
- pnAxisCount: The number of axes available on the specified card.

Return Value:  
OI_OK if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.
The OASIS-4i and OASIS-blue controller typically support 4 axes on board, but options modules are available to expand the number of axes supported on a given card. Use the `OI_GetCardAxisCount` function to determine the number of axes available on a given card.

**See Also** `OI_CountCards`

---

### OI_GetCardType

**Syntax**

```c
OI_API OI_GetCardType( int nCard, int* pnType )
```

**Description**

Indicates whether a specified module is fitted to the OASIS controller.

**Parameters**

- `nCard`  
  Zero-based index of the card in the system.
- `pnType`  
  Returns the type of card fitted.

**Return Value**

- `OI_OK` if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

The OASIS library supports the fitting of one or more of OASIS-4i and/or OASIS-blue controllers in a system. To determine the type of card fitted, use the `OI_GetCardType` function. The returned type will be one of the values listed in the following table:

<table>
<thead>
<tr>
<th><code>nType</code></th>
<th><code>value</code></th>
<th>Controller</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>OASIS_4I</code></td>
<td>Value</td>
<td>OASIS-4i</td>
</tr>
<tr>
<td><code>OASIS_BLUE</code></td>
<td>Value</td>
<td>OASIS-blue</td>
</tr>
</tbody>
</table>

**See Also** `OI_CountCards`

---

### OI_GetComponentStatus

**Syntax**

```c
OI_API OI_GetComponentStatus( int nComponent, LPDWORD lpdwStatus )
```

**Description**

Retrieves the current status report for a given component.

**Parameters**

- `nComponent`  
  The component.
- `lpdwStatus`  
  The returned status DWORD.
Return Value

OI_OK if successful.

OI_NOHARDWARE if an OASIS board is not found.

Comments

Use the OI_GetComponentStatus to find out whether a given component has been opened properly.

The following components are supported by this function:

<table>
<thead>
<tr>
<th>nComponent value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>OI_CFG_FOCUS</td>
<td>The Z-axis focus control.</td>
</tr>
<tr>
<td>OI_CFG_FILTER</td>
<td>The filter changer.</td>
</tr>
<tr>
<td>OI_CFG_FILTER2</td>
<td>The second filter changer.</td>
</tr>
<tr>
<td>OI_CFG_TURRET</td>
<td>The objective lens turret.</td>
</tr>
<tr>
<td>OI_CFG_LAMP1</td>
<td>The lamp channel 1.</td>
</tr>
</tbody>
</table>

The returned status parameter will be a combination of the following:

<table>
<thead>
<tr>
<th>Status</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>OI_OK</td>
<td>The operation completed successfully.</td>
</tr>
<tr>
<td>OI_FAILED</td>
<td>The operation failed.</td>
</tr>
<tr>
<td>OI_ABORT</td>
<td>The operation failed due to a user abort.</td>
</tr>
<tr>
<td>OI_NOHARDWARE</td>
<td>The operation failed because the required hardware is not fitted.</td>
</tr>
<tr>
<td>OI_TIMEOUT</td>
<td>The operation failed due to a timeout.</td>
</tr>
<tr>
<td>OI_INVALIDARG</td>
<td>An invalid argument was passed into the function.</td>
</tr>
<tr>
<td>OI_HARDWAREBUSY</td>
<td>The command could not be executed because the hardware was busy with another request.</td>
</tr>
<tr>
<td>OI_ACCESSDENIED</td>
<td>Access to the desired functionality is not available.</td>
</tr>
<tr>
<td>OI_NOTSUPPORTED</td>
<td>The operation is not supported by the current hardware platform.</td>
</tr>
</tbody>
</table>
OI_ILLEGALAXIS The operation attempted to use an axis that is not available.

OI_INVALIDCONFIG The requested configuration is not allowed.

See Also OI_OpenComponent, OI_CloseComponent, OI_Configure

**OI_GetConfiguration**

**Syntax**

```c
OI_API OI_GetConfiguration( int nComponent, int *pnControl )
```

**Description**

Returns the currently configured controller for a given component.

**Parameters**

- `nComponent` The component.
- `pnControl` The returned controller ID.

**Return Value**

- OI_OK if successful.

**Comments**

Currently, only two components, the Z focus and the filter changer, support configurations to controllers other than the OASIS system. These components are specified by the following values:

<table>
<thead>
<tr>
<th><code>nComponent</code> value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>OI_CFG_FOCUS</td>
<td>The Z-axis focus control. All functions dealing the Z-axis are routed to the specified controller.</td>
</tr>
<tr>
<td>OI_CFG_FILTER</td>
<td>The filter changer. All filter changer functions are routed to the specified controller.</td>
</tr>
</tbody>
</table>

The returned controller ID will be one of the following values:

<table>
<thead>
<tr>
<th><code>pnControl</code> value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>OI_OASIS</td>
<td>The OASIS controller is used for the component.</td>
</tr>
</tbody>
</table>
OI_SIM  The component’s operation is simulated.

OI_LEICA_DM  The component is an older style Leica Microsystems DMRXA, DMRXE, or DMIRBE

OI_LEICA_DM2  The component is uses the Leica Microsystems MICSTC controller.

See the OI_Congiure function for more information on component configurations.

See Also  OI_Open, OI_GetHardwareMode

---

**OI_GetDefaultAbortKeys**

Syntax  
OI_API  OI_GetDefaultAbortKeys( LPBOOL pbEnabled)

Description  
Retrieves whether the default abort key press handling has been enabled.

Parameters  

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pbEnabled</td>
<td>Returns the enabling flag.</td>
</tr>
</tbody>
</table>

Return Value  
OI_OK if successful.

Comments  
See the OI_SetDefaultAbortKeys function for a description of the handling of abort key presses.

See Also  OI_SetDefaultAbortKeys

---

**OI_GetDefaultWaitCursorEnabled**

Syntax  
OI_API  OI_GetDefaultWaitCursorEnabled( LPBOOL pbEnabled)

Description  
Retrieves whether the default wait cursors have been enabled.

Parameters  

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pbEnabled</td>
<td>Returns the enabling flag.</td>
</tr>
</tbody>
</table>

Return Value  
OI_OK if successful.

Comments  
See the OI_SetDefaultWaitCursorEnabled function for a description of the
default wait cursors.

See Also OI_SetDefaultWaitCursor

---

**OI_GetDriverOpen**

Syntax  
```
OI_API OI_GetDriverOpen(BOOL* pbOpen)
```

Description  
Determines whether the OASIS driver is already open for the current session.

Parameters  
```
pbOpen  
```
Flag returned indicating whether the OASIS driver has been opened for the current session.

Return Value  
- OI_OK if successful.
- OI_NOHARDWARE if an OASIS board is not found.

Comments  
The `pbOpen` BOOL value will set to TRUE if the OASIS hardware driver has previously been opened by the current process.

See Also  
- OI_Open, OI_Close

---

**OI_GetFlashCheckSum**

Syntax  
```
OI_API OI_GetFlashCheckSum(LPWORD pwCheckSum)
```

Description  
Retrieves the current checksum value from the user block of the flash memory.

Parameters  
```
pwCheckSum  
```
The checksum value.

Return Value  
- OI_OK if successful.

Comments  
Most physical settings for the OASIS controller are stored in the user block of the on-board flash memory. These settings include the configuration for motor currents, axis drive direction, limit switch polarity and direction, and so on.

The checksum value returned by OI_GetFlashCheckSum can help ensure that the user flash block contains the desired information, has not been corrupted, etc., by comparing the flash checksum of the current system with a previously stored value from a known standard setup.

See Also  
- OI_Open
**OI_GetHardwareMode**

**Syntax**

```c
OI_API OI_GetHardwareMode(int* pnMode)
```

**Description**
Retrieves the hardware mode of operation, indicating either hardware access or simulated operation.

**Parameters**

- `pnMode`  
  Pointer to value Parameter indicating the hardware status. This will be set to either:
  - OI_OASIS (a value of 1), or
  - OI_SIM (a value of 0).

**Return Value**
- OI_OK if successful.
- OI_NOHARDWARE if an OASIS board is not found.

**Comments**
The `OI_GetHardwareMode` function can be used to check what mode the OASIS DLL is currently using.

**See Also**
- `OI_SetHardwareMode`, `OI_Open`

---

**OI_GetMultiAxisMode**

**Syntax**

```c
OI_API OI_GetMultiAxisMode( int* pnMode )
```

**Description**
Retrieves the current mode of operation for functions using AxisID parameters in systems with multiple OASIS controllers fitted.

**Parameters**

- `pnMode`  
  Returns the current multiple axis mode.

**Return Value**
- OI_OK if successful.
- OI_NOHARDWARE if an OASIS board is not found.

**Comments**
The returned `nMode` will be either:

<table>
<thead>
<tr>
<th><code>nMode</code></th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OI_MULTI_MODE_ID</td>
<td>0</td>
<td>Use 1 to 4*N axes, overrides <code>OI_SelectCard</code> setting (default).</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OI_MULTI_MODE_ROUTE</td>
<td>1</td>
<td>Use 1 to 4 axes, routed to a particular card via the <code>OI_SelectCard</code> setting</td>
</tr>
</tbody>
</table>

See the `OI_SetMultiAxisMode` for a detailed description on using AxisID values with multiple controllers.
See Also  OI_SetMultiAxisMode, OI_SelectCard

**OI_GetSelectedCard**

Syntax  

```
OI_API OI_GetSelectedCard( int* pnCard )
```

Description  

Retrieves the currently selected card in a multi-card situation.

Parameters  

- `pnCard`  
  Returns the zero-based index of the active card.

Return Value  

- `OI_OK` if successful.
- `OI_NOHARDWARE` if an OASIS board is not found.

Comments  

In a multiple OASIS card installation, use the `OI_GetSelectedCard` function to determine which card is currently active, i.e., which card has been selected to receive the API commands.

See Also  OI_SelectCard, OI_CountCards

**OI_GetTotalAxisCount**

Syntax  

```
OI_API OI_GetTotalAxisCount( int* pnAxisCount )
```

Description  

Retrieves the total number of available axes.

Parameters  

- `pnAxisCount`  
  Returns the total number of available axes.

Return Value  

- `OI_OK` if successful.

Comments  

In systems where multiple OASIS controllers are installed, the total axis count will be `4*N`, where `N` is the number of controllers. The `OI_GetTotalAxisCount` function will return this value.

See Also  OI_SetMultiAxisMode, OI_GetMultiAxisMode

**OI_GetUseCount**

Syntax  

```
OI_API OI_GetUseCount( int* pnUsers)
```

See Also  OI_SetMultiAxisMode, OI_GetMultiAxisMode
**Description**
Retrieves the total number of users that have connected to the OASIS DLL.

**Parameters**
- *pnUsers* Returns the current number of users of the library.

**Return Value**
OI_OK if successful.

**Comments**
The value returned by `OI_GetUseCount` indicates the number of processes that have opened the OASIS hardware via calls to `OI_Open`.

**See Also**
`OI_Open`, `OI_Close`

---

**OI_IsModuleFitted**

**Syntax**
```c
OI_API OI_IsModuleFitted (int nModule, LPBOOL pbFitted )
```

**Description**
Indicates whether a specified module is fitted to the OASIS controller.

**Parameters**
- *nModule* The module in question.
- *pbFitted* Returns TRUE if the module is fitted.

**Return Value**
OI_OK if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**
A number of daughter modules may be fitted to the OASIS-4i and OASIS-blue controllers. These modules are:

<table>
<thead>
<tr>
<th>nModule</th>
<th>Meaning</th>
<th>Controller</th>
</tr>
</thead>
<tbody>
<tr>
<td>OASIS_AF</td>
<td>Video autofocus.</td>
<td>OASIS-4i</td>
</tr>
<tr>
<td>OASIS_XA1</td>
<td>Fifth axis.</td>
<td>OASIS-4i</td>
</tr>
<tr>
<td>OASIS_DC1</td>
<td>Trigger I/O.</td>
<td>OASIS-4i</td>
</tr>
<tr>
<td>BLUE_EXPIO</td>
<td>Encoder and Trigger I/O.</td>
<td>OASIS-blue</td>
</tr>
</tbody>
</table>

You can use the `OI_IsModuleFitted` function to determine if a particular hardware option is present.

**See Also**
`OI_ReadPCBStatus`, `OI_ReadPCBStatusEx`
**OI_Open**

**Syntax**

OI_API OI_Open (void)

**Description**

Initialises the OASIS driver for use in the current session.

**Parameters**

None.

**Return Value**

OI_OK if successful.

OI_NOHARDWARE if an OASIS board is not found.

**Comments**

The OI_Open function opens the OASIS hardware driver and therefore should be called before any functions that access the OASIS hardware.

**See Also**

OI_Close, OI_SetHardwareMode

**OI_OpenComponent**

**Syntax**

OI_API OI_OpenComponent( int nComponent )

**Description**

Opens a component for operation, using its configured controller.

**Parameters**

nComponent

The component.

**Return Value**

OI_OK if successful.

**Comments**

Currently, only two components, the Z focus and the filter changer, support configurations to controllers other than the OASIS system. These components are specified by the following values:

<table>
<thead>
<tr>
<th>nComponent value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>OI_CFG_FOCUS</td>
<td>The Z-axis focus control. All functions dealing the Z-axis are routed to the specified controller.</td>
</tr>
<tr>
<td>OI_CFG_FILTER</td>
<td>The filter changer. All filter changer functions are routed to the specified controller.</td>
</tr>
</tbody>
</table>

Note that normally, if the desired component’s controller has been selected prior to the call to OI_Open, the component will be opened automatically and the OI_OpenComponent is not needed.
See Also   OI_Open, OI_Configure, OI_CloseComponent

---

**OI_ReadCardStatus**

**Syntax**

OI_API OI_ReadCardStatus( int nCard, LPDWORD lpdwStatus )

**Description**

Retrieves the current hardware status report for a given board.

**Parameters**

- *nCard*  
  The zero-based index of the card for which the status report is desired.

- *lpdwStatus*  
  The returned status DWORD.

**Return Value**

- OI_OK if successful.
- OI_NOHARDWARE if an OASIS board is not found.

**Comments**

See the OI_ReadPCBStatus function for a description of the status DWORD values.

**See Also**   OI_ReadPCBStatus

---

**OI_ReadPCBName**

**Syntax**

OI_API OI_ReadPCBName( int nCard, LPSTR szName, int nStringLen )

**Description**

Retrieves the current hardware status report for a given board.

**Parameters**

- *nCard*  
  The zero-based index of the card for which the status report is desired.

- *szName*  
  Pointer to buffer to receive the controller name.

- *nStringLen*  
  The length of the passed-in szName string buffer.

**Return Value**

- OI_OK if successful.
- OI_NOHARDWARE if an OASIS board is not found.

**Comments**

The OI_ReadPCBName function is used to return a string value indicating the name of a given controller in the system.

For instance, the returned string will be "OASIS-blue" if an OASIS-blue controller is installed.

**See Also**   OI_ReadPCBType
OI_ReadPCBStatus

Syntax

OI_API OI_ReadPCBStatus(LPDWORD *lpdwHWState)

Description

Retrieves the current hardware status report.

Parameters

lpdwHWState

Returns a status DWORD indicating the current hardware status, as defined in the Comments section below.

Return Value

OI_OK if successful.

OI_NOHARDWARE if an OASIS board is not found.

Comments

The returned DWORD has the following indicator bits:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>S_PCB_MOTOR_VOLTS_OK</td>
<td>1 = Motor supply &gt;= 10V i.e. OK</td>
</tr>
<tr>
<td>1</td>
<td>S_PCB_TEMP_OK</td>
<td>1 = PCB Temperature too high</td>
</tr>
<tr>
<td>2</td>
<td>S_PCB_ADC_YIN_OK</td>
<td>1 = ADC analogue input Y is zero (correct) at switch on</td>
</tr>
<tr>
<td>3</td>
<td>S_PCB_ADC_XIN_OK</td>
<td>1 = ADC analogue input X is zero (correct) at switch on</td>
</tr>
<tr>
<td>4</td>
<td>S_PCB_AF_FITTED</td>
<td>1 = OASIS-AF Auto-Focus Module Detected</td>
</tr>
<tr>
<td>5</td>
<td>S_PCB_AF_TYPE0</td>
<td>2 bit code indicating module type</td>
</tr>
<tr>
<td>6</td>
<td>S_PCB_AF_TYPE1</td>
<td>2 bit code indicating module type</td>
</tr>
<tr>
<td>7</td>
<td>S_PCB_VIDEO_ENCODER_OK</td>
<td>1 = Video Encoder configured OK</td>
</tr>
<tr>
<td>8</td>
<td>S_PCB_CAMERA_DETECTED</td>
<td>1 = Camera input detected</td>
</tr>
<tr>
<td>9</td>
<td>S_PCB_CAMERA_CHANNEL</td>
<td>1 = Camera Channel 3, 0 = Camera channel 0 (default)</td>
</tr>
<tr>
<td>10</td>
<td>S_PCB_CAMERA_FREQ</td>
<td>1 = Camera Frequency is 50 Hz, 0 = 60 Hz</td>
</tr>
<tr>
<td>11</td>
<td>S_PCB_CAMERA_TYPE</td>
<td>1 = Colour Camera, 0 = Mono Camera</td>
</tr>
<tr>
<td>12</td>
<td>S_PCB_SERIAL_DEV_DETECTED</td>
<td>1 = Serial device detected on RS232_0</td>
</tr>
<tr>
<td></td>
<td>Code Value</td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>------------</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>S_PCB_MOUSE_DETECTED</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>S_PCB_TRACKBALL_FITTED</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>S_PCB_JOYSTICK_FITTED</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>S_PCB_FLASH_OI_OK</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>S_PCB_FLASH_USER_OK</td>
<td></td>
</tr>
</tbody>
</table>

The specified Code value may be used to check a specific bit. For instance, the following example function tests for the presence of a trackball controller:

```c
BOOL IsTrackBallFitted()
{
    DWORD dwStatus;
    OI_ReadPCBStatus( &dwStatus );
    if ( dwStatus & S_PCB_TRACKBALL_FITTED )
        return TRUE;
    else
        return FALSE;
}
```

**See Also**  
OI_Open

**OI_ReadPCBTemperature**

**Syntax**  
`OI_API OI_ReadPCBTemperature(double *pdTempC)`

**Description**  
Retrieves the current hardware status report.

**Parameters**  
- `pdTempC`: Returns the current OASIS PCB temperature, in degrees Celsius.

**Return Value**  
- OI_OK if successful.
- OI_NOHARDWARE if an OASIS board is not found.

**Comments**  
The returned temperature is derived from a reading of ADC channel 4. Use the `OI_ReadPCBStatus` function to determine if the board temperature is operating within design parameters.
See Also  OI_ReadPCBStatus

OI_ReadPCBType

Syntax  
OI_API OI_ReadPCBType (int* pnType)

Description  Retrieves the current hardware status report.

Parameters  pnType  Returns the type of OASIS controller fitted.

Return Value  
OI_OK if successful.
OI_NOHARDWARE if an OASIS board is not found.

Comments  The returned value indicates the type of OASIS controller fitted in the system. The value will be one of the following:

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OASIS_BLUE</td>
<td>1</td>
<td>The controller is an OASIS-blue card.</td>
</tr>
<tr>
<td>OASIS_4I</td>
<td>0</td>
<td>The controller is an OASIS-4i card.</td>
</tr>
</tbody>
</table>

See Also  OI_ReadPCBName, OI_ReadPCBStatus

OI_ResetHardware

Syntax  
OI_API OI_ResetHardware()

Description  Resets the controller hardware, similar to a power on sequence.

Parameters  None.

Return Value  
OI_OK if successful.

Comments  The OI_ResetHardware function resets the internal controller hardware to its initial state.

Note that the default settings are re-read from the flash memory, so current settings and position information may be lost. You may precede a call to OI_ResetHardware with calls to OI_SaveSettings and OI_SavePositions to store the current values before the reset. Subsequent calls to OI_LoadSettings and OI_LoadPositions after the reset will restore your previous settings and positions.

See Also  OI_SaveSettings, OI_SavePositions, OI_LoadSettings, OI_LoadPositions
**OI_SelectCard**

**Syntax**

\[ \text{OI\_API OI\_SelectCard( int nCard )} \]

**Description**

Selects which card is the target for API commands.

**Parameters**

- \( nCard \)
  
  The zero-based index of the card.

**Return Value**

- OI_OK if successful.
- OI_NOHARDWARE if an OASIS board is not found.

**Comments**

In a multiple-card situation, most API commands—except for general axis commands—are routed to the currently active board. By default this is board 0, i.e., the first board detected in the system. Use the `OI_SelectCard` function to select which board is to be the target for all subsequent API commands.

Note that the general axis functions, such as `OI_ReadAxis` and `OI_MoveAxis`, may instead use an AxisID parameter to determine the target axis and board, depending on the `OI_SetMultiAxisMode` setting.

**See Also**

- `OI_GetSelectedCard`, `OI_CountCards`, `OI_SetMultiAxisMode`

---

**OI_SetAHMDelay**

**Syntax**

\[ \text{OI\_API OI\_SetAHMDelay ( DWORD dwMSecs )} \]

**Description**

Sets the base delay used when using Leica AHM components.

**Parameters**

- \( dwMSecs \)
  
  Base delay for Leica AHM-related calls.

**Return Value**

- OI_OK if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

Some components such as focus, objective turret, and XY stage may be configured as `OI_LEICA_AHM` or `OI_LEICA_ISO`, which support commands via the Leica Microsystems AHM interface. When using these components, the OASIS DLL needs to apply some delays to ensure proper initialisation of components. The `OI_SetAHMDelay` function sets a base delay for these operations.

The default delay value is 1000 msec, i.e., 1 second. Reducing the delay value will result in faster startup times for AHM components but may also result in unreliable connection to the Leica components.
See Also  OI_GetAHMDelay

**OI_SetAutoFocusHWMode**

**Syntax**  
```c
OI_API OI_SetAutoFocusHWMode(int nMode)
```

**Description**  
Sets the mode of operation for the automatic focus either to hardware access or for simulated operation.

**Parameters**  
- `nMode`  
  Parameter indicating the hardware status. This should be set to either:
  - OI_OASIS (a value of 1), or
  - OI_SIM (a value of 0).

**Return Value**  
- OI_OK if successful.
- OI_NOHARDWARE if the OASIS-AF hardware is not installed and an attempt is made to set the AutoFocus system into OI_OASIS hardware mode.

**Comments**  
The OASIS-AF video board provides analysis of an incoming standard video signal for automatic focus operation and other measurements.

The **OI_SetAutoFocusHWMode** function allows simulated operation of the OASIS-AF facilities when the hardware is not fitted.

See Also  OI_GetAutoFocusHWMode, OI_AutoFocus, OI_ReadFocusScore, OI_ReadVideoResults

**OI_SetDefaultAbortKeys**

**Syntax**  
```c
OI_API OI_SetDefaultAbortKeys( BOOL bEnabled)
```

**Description**  
Enables the use of default abort key press handling during movement operations where a wait until stopped flag is used.

**Parameters**  
- `bEnabled`  
  The enabling flag.

**Return Value**  
- OI_OK if successful.

**Comments**  
The OASIS DLL normally processes keystroke messages when in a wait loop that check for movement to stop, for instance, any move function called with a non-zero wait parameter or the OI_WaitForStopped... type of functions. The keys that cause an aborted movement are the ESCAPE key and the CTRL-C key combination. By default, the wait loop will look at the current thread's message queue to see if these keys have been pressed and, if so, will abort the
movement.

However, some application may wish to disable these keys, in order to provide their own handling of movement aborts. The OI_SetDefaultAbortKeys function may be used to disable or re-enable the default keystroke checking behaviour.

By default, the keystroke checking is enabled, i.e., all movement wait loops will check the message queue for ESC or CTRL-C key presses and will stop the relevant motor drive and return from the wait loop if they are detected.

If you disable the default abort key behaviour, it is recommended that your application provide some means for the user to halt a movement action. This provides a safety mechanism by allowing the user to quickly terminate a movement if necessary.

See Also  OI_GetDefaultAbortKeys, OI_WaitForStoppedXYZ, OI_WaitForStoppedXY, OI_WaitForStoppedZ, OI_WaitForStoppedF, OI_WaitForAutoFocus

---

**OI_SetDefaultWaitCursorEnabled**

**Syntax**  
OI_API OI_SetDefaultWaitCursorEnabled( BOOL bEnabled)

**Description**  
Enables the use of default wait cursors during movement operations.

**Parameters**  
*bEnabled*  
The enabled flag.

**Return Value**  
OI_OK if successful.

**Comments**  
The OASIS DLL can be configured to show special wait cursors when various control actions are taking place. These actions are movements with wait flags enabled, waiting for a movement to finish, and waiting for an autofocus to finish.

The OI_SetDefaultWaitCursorEnabled function is used to enable or disable the display of these cursors during those actions. Enabling the cursors provides a simple means for an application to provide feedback to the user than an automation action is currently underway and the system is waiting for the action to complete.

See Also  OI_GetDefaultWaitCursorEnabled

---

**OI_SetHardwareMode**

**Syntax**  
OI_API OI_SetHardwareMode(int nMode)

**Description**  
Sets the hardware mode of operation, either for hardware access or for
simulated operation.

**Parameters**

Parameter indicating the hardware status. This should be set to either:

- OI_OASIS (a value of 1), or
- OI_SIM (a value of 0).

**Return Value**

OI_OK if successful.

**Comments**

Calls to OI_SetHardwareMode should be made prior to opening the OASIS driver via calls to OI_Open.

The OASIS hardware functionality may be simulated by the DLL when a board is not present in the system, for instance allowing development on systems that do not contain the OASIS hardware.

If the hardware mode is set to OI_OASIS but the OASIS hardware is not installed in the system, the OI_SetHardwareMode function will return OI_OK, but subsequent calls to OI_Open or any other function that accesses the OASIS hardware will fail.

**See Also**

OI_Open, OI_GetHardwareMode

---

**OI_SetMultiAxisMode**

**Syntax**

OI_API OI_SetMultiAxisMode( int nMode )

**Description**

Sets the mode of operation for general axis functions, i.e., those functions that use an AxisID parameter, when more than one OASIS controller is present.

**Parameters**

- **nMode**
  
  The desired mode of operation, as described in the comments below.

**Return Value**

OI_OK if successful.

**Comments**

In situations where multiple OASIS controllers are present, those functions taking an AxisID parameter either may be routed to the currently selected controller, as defined by the OI_SelectCard function, or be routed to the corresponding board based on the AxisID.

The following values may be used with the nMode parameter:

<table>
<thead>
<tr>
<th>nMode</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
</table>

---

OASIS4I DLL Manual

Version 3.1.3

PAGE 57
The OI_MULTI_MODE_ID option allows AxisID values to range from 1 to $4N$ axes, where $N$ is the number of OASIS cards fitted in the system. For instance, if 3 OASIS cards are fitted, then the AxisID values may range from 1 to 12. The OASIS DLL will automatically determine which board to use based on the AxisID value. That is, AxisID values 1-4 correspond to the four axes on card 0, while AxisID values 5-8 are the four axes on card 1, and so on.

The OI_MULTI_MODE_ID option works independently of the OI_SelectCard setting. Therefore, those functions that use an AxisID parameter may be used in conjunction with the other API functions without affecting the currently active card.

The OI_MULTI_MODE_ROUTE option always uses AxisID values ranging only from 1 to 4 that are routed to the active board, as selected by the OI_SelectCard function. This function provides compatibility with existing applications that use the AxisID-based functions and wish to have those functions routed to a given controller using the OI_SelectCard function.

See Also OI_GetMultiAxisMode, OI_SelectCard

---

**Version Information**

**OI_GetDriverVersion**

**Syntax**

```c
OI_API OI_GetDriverVersion( LPSTR lpszVersion, int nStringLen )
```

**Description**

Returns a string containing version information for the OASIS DLL.

**Parameters**

- `lpszVersion`: String buffer into which the version information will be copied.
- `nStringLen`: The size of the string buffer passed in the `lpszVersion` parameter.

**Return Value**

- `OI_OK` if successful.

**Comments**

The `OI_GetDriverVersion` returns the file version information for the OASIS4I.DLL file.
See Also  OI_ReadPCBVersion

---

**OI_ReadPCBID**

**Syntax**  

```c
OI_API OI_ReadPCBID ( LPSTR lpszBuffer, int nStringLen )
```

**Description**  

Returns a string containing the firmware ID information for the OASIS controller hardware.

**Parameters**  

- `lpszBuffer`  
  String buffer into which the version information will be copied.

- `nStringLen`  
  The size of the string buffer passed in the `lpszVersion` parameter.

**Return Value**  

`OI_OK` if successful.

**Comments**  

Use the `OI_ReadPCBID` function to read the extended firmware ID of the OASIS hardware.

**See Also**  

`OI_GetDriverVersion`

---

**OI_ReadPCBVersion**

**Syntax**  

```c
OI_API OI_ReadPCBVersion ( LPSTR lpszVersion, int nStringLen )
```

**Description**  

Returns a string containing the firmware version information for the OASIS controller hardware.

**Parameters**  

- `lpszVersion`  
  String buffer into which the version information will be copied.

- `nStringLen`  
  The size of the string buffer passed in the `lpszVersion` parameter.

**Return Value**  

`OI_OK` if successful.

**Comments**  

Use the `OI_ReadPCBVersion` function to read the firmware version currently in use by the OASIS hardware.

**See Also**  

`OI_GetDriverVersion`
OI_ReadSerialNum

Syntax

OI_API OI_ReadSerialNum( int nCard, LPSTR lpszSerialNum, int nStringLen )

Description

Returns a string containing the serial number for the OASIS controller hardware.

Parameters

- **nCard**
  The zero-based index of the desired card. For single card installations, use a value of 0.

- **lpszSerialNum**
  String buffer into which the serial number information will be copied.

- **nStringLen**
  The size of the string buffer passed in the lpszSerialNum parameter.

Return Value

OI_OK if successful.

Comments

Each OASIS controller will have a unique serial number. Use the OI_ReadSerialNum function to retrieve the serial number as a string.

This value may be useful in situations where multiple controllers are fitted, in order to uniquely distinguish between the cards.

See Also

OI_GetDriverVersion, OI_ReadPCBVersion

---

General, Single Axis Control

General-purpose, single axis functions provide independent access to each of the four available OASIS axis controllers. The desired axis is indicated by the **AxisID** parameter, as defined in OI_CONST.H:

<table>
<thead>
<tr>
<th>Axis Code</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>OI_XAXIS</td>
<td>1</td>
</tr>
<tr>
<td>OI_YAXIS</td>
<td>2</td>
</tr>
<tr>
<td>OI_ZAXIS</td>
<td>3</td>
</tr>
<tr>
<td>OI_FAXIS</td>
<td>4</td>
</tr>
<tr>
<td>OI_TAXIS</td>
<td>5</td>
</tr>
</tbody>
</table>
**OI_ClearAxisUserLimits**

**Syntax**

```
OI_API OI_ClearAxisUserLimits ( int AxisID )
```

**Description**

Clears the user (software) limit values, i.e., makes them unset. The controller will no longer obey the user limits once this function is called.

**Parameters**

- **AxisID**: The desired axis (see the introduction of this section for the appropriate constants).

**Return Value**

- **OI_OK** if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

Each axis of the OASIS controller can be set to use software limits, specifying a minimum and maximum position to be used for the range of travel. When the user software limits are set, the controller will obey these positions as if they were physical limits of travel. The `OI_ClearAxisUserLimits` function will clear (disable) the use of software user limits on the specified axis.

**See Also**

`OI_SetAxisUserLimits`, `OI_GetAxisUserLimits`

---

**OI_DriveAxisContinuous**

**Syntax**

```
OI_API OI_DriveAxisContinuous(int AxisID, int nSpeed)
```

```
OI_API OI_DriveAxisContinuousEx(int nCard, int AxisID, int nSpeed)
```

**Description**

Moves the axis continuously at a given rate and direction.

**Parameters**

- **nCard**: The zero-based index of the card to use, in a multi-card configuration.
- **AxisID**: The desired axis (see the introduction of this section for the appropriate constants).
- **nSpeed**: The speed in half-steps per second.

**Return Value**

- **OI_OK** if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

The `nSpeed` parameter specifies a speed in half-steps per second.

The `nSpeed` value is signed to indicate the direction of travel, i.e., a negative speed causes a continuous drive in the negative direction, and may be any
integer in the range of –4096 to +4096.

To stop the continuous movement, use a corresponding call to `OI_HaltAxis` function.

**Warning**

If the axis limits are not appropriately set for the physical limitations of the microscope, continuous movement of an axis could cause the collision of mechanical and optical components of the microscope system.

Caution should be taken by an application to ensure that appropriate safety mechanisms are in place to prevent damage to the optical system. Usually this means that safe user limits and/or limit switch positions for each axis have been set so as to prevent movements that would result in collisions.

The OASIS DLL provides a safety function, `OI_EmergencyStopAll`, to immediately stop all axes from being driven. An application should provide facilities allowing the user to effectively access this function in all appropriate situations in order to ensure hardware damage is prevented.

See Also `OI_HaltAxis, OI_HaltXY, OI_HaltZ, OI_HaltF, OI_EmergencyStopAll`

---

**OI_FlashReadAxisPitch**

**Syntax**

```c
OI_API OI_FlashReadAxisPitch( int AxisID, double *pdPitchMM )
```

**Description**

Retrieves the status of backlash correction for a given axis.

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AxisID</td>
<td>The desired axis (see the introduction of this section for the appropriate constants).</td>
</tr>
<tr>
<td>pdPitchMM</td>
<td>The returned pitch of the given axis, in millimetres.</td>
</tr>
</tbody>
</table>

**Return Value**

`OI_OK` if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

This function reads the pitch value as defined for a given axis in the controller’s user flash memory block. Note that this value may differ from the currently active pitch value for the axis, since the pitch specified by software calls to `OI_SetPitchXY, OI_SetPitchZ, OI_SetPitchF` supersedes whatever value is written in the flash memory.

See Also `OI_SetPitchFromFlashXYZ, OI_SetPitchXY, OI_SetPitchZ, OI_SetPitchF`

---

**OI_GetAxisBacklash**

**Syntax**

```c
OI_API OI_GetAxisBackslash(int AxisID, BOOL* pbEnabled)
```
Description
Retrieves the status of backlash correction for a given axis.

Parameters
- **AxisID**: The desired axis (see the introduction of this section for the appropriate constants).
- **pbEnabled**: A flag indicating the status of backlash correction:
  - A TRUE value indicates that backlash correction is enabled.
  - A FALSE value indicates that backlash correction is disabled.

Return Value
- **OI_OK** if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

Comments
See the OI_SetAxisBacklash function for more information regarding backlash correction.

See Also
- OI_SetAxisBacklash

**OI_GetAxisCruise**

Syntax
- **OI_API OI_GetAxisCruise(int AxisID, int* pnCruise)**
- **OI_API OI_GetAxisCruiseEx(int nCard, int AxisID, int* pnCruise)**

Description
Retrieves the current cruise speed index for a given axis.

Parameters
- **nCard**: The zero-based index of the card to use, in a multi-card configuration.
- **AxisID**: The desired axis (see the introduction of this section for the appropriate constants).
- **pnCruise**: Returns the current cruise index.

Return Value
- **OI_OK** if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

Comments
See the Comments for OI_SetAxisCruise for more information regarding cruise speed.

See Also
- OI_SetAxisCruise, OI_SetAxisRamp, OI_ReadRampValue
**OI_GetAxisInitMethod**

**Syntax**

```c
OI_API OI_GetAxisInitMethod (int AxisID, int *pnMethod)
```

**Description**
Sets the.

**Parameters**

- `AxisID`
  The desired axis (see the introduction of this section for the appropriate constants).

- `pnMethod`
  The returned value for the method to use for initialising the axis.

**Return Value**

- `OI_OK` if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

See the definition of `OI_SetAxisInitMethod` for a full description of the different methods for initialising an axis.

**See Also**

`OI_SetAxisInitMethod`, `OI_InitializeXY`, `OI_SetAxisTravel`, `OI_GetAxisTravel`

---

**OI_GetAxisMaxMove**

**Syntax**

```c
OI_API OI_GetAxisMaxMove(int AxisID, LPDWORD lpdwValue)
```

**Description**
Retrieves the current pre-defined ramp table in use for a given axis.

**Parameters**

- `AxisID`
  The desired axis (see the introduction of this section for the appropriate constants).

- `lpdwValue`
  Returns the maximum allowable move for the given axis, as described in the Comments section below.

**Return Value**

- `OI_OK` if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

The OASIS controller provides protection against extreme moves, e.g., moves that involve an unusually large distance for a typical microscope automation situation. If the user limits have not been properly defined, these moves could cause physical damage due to collisions such as the objective lens striking the specimen.

This protection is implemented as a maximum allowable move, defined in microsteps. The `OI_GetAxisMaxMove` function allows the current setting for a given axis to be returned.
The values for the maximum move for each axis are stored in the OASIS flash memory.

See Also  
OI_ReadMaxMoveXY, OI_ReadMaxMoveZ, OI_ReadMaxMoveF

---

**OI_GetAxisPitch**

**Syntax**

```
OI_API OI_SetAxisPitch(int AxisID, double* pdPitchMM)

OI_API OI_SetAxisPitchEx(int nCard, int AxisID, double* pdPitchMM)
```

**Description**

Sets the pre-defined acceleration / deceleration ramp table to use.

**Parameters**

- `nCard`  
  The zero-based index of the card to use, in a multi-card configuration.

- `AxisID`  
  The desired axis (see the introduction of this section for the appropriate constants).

- `pdPitchMM`  
  The returned axis pitch, in mm, i.e., the expected travel per revolution of the motor.

**Return Value**

- OI_OK if successful.

- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

Calibration data for an axis is defined by the amount of travel expected for either one step or for each turn of the motor. For the latter, the pitch value is used, defined in millimetres per turn.

See Also  
OI_GetAxisRamp, OI_SetAxisCruise, OI_GetAxisRampValue

---

**OI_GetAxisRamp**

**Syntax**

```
OI_API OI_GetAxisRamp(int AxisID, int* pnRamp)

OI_API OI_GetAxisRampEx(int nCard, int AxisID, int* pnRamp)
```

**Description**

Retrieves the current pre-defined ramp table in use for a given axis.

**Parameters**

- `nCard`  
  The zero-based index of the card to use, in a multi-card configuration.

- `AxisID`  
  The desired axis (see the introduction of this section for the appropriate constants).
\textit{pnRamp}

Returns the current ramp, as described in the Comments section below.

\textbf{Return Value}

- \texttt{OI_OK} if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

\textbf{Comments}

Three pre-defined tables may be selected for a given axis, as indicated by the \textit{pnRamp} parameter:

<table>
<thead>
<tr>
<th>\textit{pnRamp} value</th>
<th>Acceleration</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Slow</td>
</tr>
<tr>
<td>1</td>
<td>Medium</td>
</tr>
<tr>
<td>2</td>
<td>Fast</td>
</tr>
</tbody>
</table>

\textbf{See Also} \texttt{OI_SetAxisRamp}, \texttt{OI_SetAxisCruise}, \texttt{OI_GetAxisCruise}, \texttt{OI_GetAxisRampValue}

---

\textbf{\texttt{OI_GetAxisRange}}

\textbf{Syntax} \texttt{OI\_API OI\_GetAxisRange(int AxisID, double* pdMin, double* pdMax)}

\textbf{Description}

Read the available range of travel for the axis, as defined by the minimum and maximum position values of the User Limits.

\textbf{Parameters}

- \textit{AxisID} The desired axis (see the introduction of this section for the appropriate constants).
- \textit{pdMin} Returns the minimum coordinate value for the axis, in microns.
- \textit{pdMax} Returns the maximum coordinate value for the axis, in microns.

\textbf{Return Value}

- \texttt{OI_OK} if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

\textbf{Comments}

Each axis has available range of motion, typically set during the initialisation procedure for that axis. The range is defined by minimum and maximum values in the micron-based coordinate system of the axis. The values will be the current software limits, if set.

\textbf{See Also} \texttt{OI\_InitializeXY, OI\_InitializeZ, OI\_InitializeF}
### OI_GetAxisSense

**Syntax**

```
OI_API OI_GetAxisSense(int AxisID, int* pnSense)
```

**Description**

Retrieves the joystick drive sense for an axis.

**Parameters**

- **AxisID**
  The desired axis (see the introduction of this section for the appropriate constants).
- **pnSense**
  Returns the drive sense flag.
  - A value of zero (0) indicates standard movement.
  - A non-zero value indicates reversed movement.

**Return Value**

- OI_OK if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

See the Comments for `OI_SetAxisSense` for more information about drive sense.

**See Also**

`OI_SetAxisSense`

---

### OI_GetAxisStepSize

**Syntax**

```
OI_API OI_GetAxisStepSize(int AxisID, double* pdStepSize)
```

**Description**

Retrieves the current distance of travel, in microns, for each micro-step.

**Parameters**

- **AxisID**
  The desired axis (see the introduction of this section for the appropriate constants).
- **pdStepSize**
  Returns the current step size, in microns.

**Return Value**

- OI_OK if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

See the `OI_SetAxisStepSize` function for more information about step sizes.

**See Also**

`OI_SetAxisStepSize`, `OI_SetPitchXY`
**OI_GetAxisStepsPerRev**

Syntax  
OI_API OI_GetAxisStepsPerRev( int AxisID, LPDWORD lpdwStepsPerRev )

Description  
Retrieves the number of microsteps made per motor revolution.

Parameters  
- **AxisID**  
The desired axis (see the introduction of this section for the appropriate constants).
- **lpdwStepsPerRev**  
The returned number of microsteps per revolution

Return Value  
OI_OK if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

Comments  
The number of microsteps per revolution determines the resolution of motor stepping. The more microsteps made per revolution, the finer the step size.

This value is set in the Flash memory of the OASIS controller hardware, and may be changed using the Flash memory configuration utility application. Typically this value is set to achieve a desired minimum step size for a given configuration (for instance, 20,000 steps per rev with a 2 mm pitch lead screw gives a step size of 1 micron. Also this value should be set when using encoders to ensure an appropriate ratio of microsteps to encoder steps, such as 2:1.

See Also  
OI_SetPitchXY, OI_GetAxisStepSize, OI_SetAxisStepSize

---

**OI_GetAxisTravel**

Syntax  
OI_API OI_GetAxisTravel (int AxisID, double* pdMin, double* pdMax)

Description  
Sets the.

Parameters  
- **AxisID**  
The desired axis (see the introduction of this section for the appropriate constants).
- **pdMin**  
The returned value for the minimum available travel, in microns.
- **pdMax**  
The returned value for the maximum available travel, in microns.

Return Value  
OI_OK if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure. 
Comments  Each axis may have an associated range of travel, typically defined by the
distance found between the physical limits of travel. For instance, for the X and
Y axes, the range of travel is found automatically during the XY initialisation
process, which drives to each end of travel to locate the physical limit switches.
Once determined, the range of travel is defined to be the distance, in microns,
between the switches.

Use the OI_GetAxisTravel function to retrieve the current minimum and
maximum positions defined for a given axis

See Also  OI_SetAxisInitMethod, OI_InitializeXY, OI_SetAxisTravel

**OI_GetAxisUserLimits**

Syntax  OI_API OI_GetAxisUserLimits ( int AxisID, double* pdMin, double* pdMax )

Description  Clears the user (software) limit values, i.e., makes them unset. The controller
will no longer obey the user limits once this function is called.

Parameters  

AxisID  The desired axis (see the introduction of this section
for the appropriate constants).

pdMin  The returned position of the minimum user limit.

pdMax  The returned position of the maximum user limit.

Return Value  OI_OK if successful.

If unsuccessful, a combination of error codes may be returned to indicate the
reason for failure.

Comments  Each axis of the OASIS controller can be set to use software limits, specifying a
minimum and maximum position to be used for the range of travel. When the
user software limits are set, the controller will obey these positions as if they
were physical limits of travel. The OI_GetAxisUserLimits function returns the
current user limit values.

See Also  OI_SetAxisUserLimits, OI_ClearAxisUserLimits

**OI_HaltAxis**

Syntax  OI_API OI_HaltAxis(int AxisID)

IO_API OI_HaltAxisEx(int nCard, int AxisID)

Description  Stops the indicated axis.

Parameters  
nCard  The zero-based index of the card to use, in a multi-card configuration.
AxisID

The desired axis (see the introduction of this section for the appropriate constants).

Return Value

OI_OK if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

Comments

The OI_HaltAxis function will stop an axis using the currently defined deceleration ramp for that axis. This will ensure positional accuracy is maintained during the halt.

To immediately stop an axis, without using the deceleration ramp, use the OI_EmergencyStopAll function.

See Also OI_DriveAxisContinuous, OI_HaltXY, OI_HaltZ, OI_HaltF, OI_EmergencyStopAll

---

**OI_LookupAxisSpeed**

Syntax

OI_API OI_LookupAxisSpeed( int AxisID, int nCruise, double* pdSpeed )

Description

Returns the actual speed in mm/sec for a given cruise index.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>AxisID</code></td>
<td>The desired axis (see the introduction of this section for the appropriate constants).</td>
</tr>
<tr>
<td><code>nCruise</code></td>
<td>The cruise speed index.</td>
</tr>
<tr>
<td><code>pdSpeed</code></td>
<td>The returned speed in mm/sec.</td>
</tr>
</tbody>
</table>

Return Value

OI_OK if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

Comments

The cruise speed is an index into an acceleration LUT. To find the actual speed for a given cruise index, use the OI_LookupAxisSpeed function.

See Also OI_LookupSpeedXY, OI_LookupSpeedZ, OI_LookupSpeedF, OI_LookupSpeedT, OI_LookupSpeedS

---

**OI_MoveAxis**

Syntax

OI_API OI_MoveAxis(int AxisID, double dValue, int nWait)

OI_API OI_MoveAxisEx(int nCard, int AxisID, double dValue, int nWait)
Description
Move to the specified position, waiting for completion and settling as desired.

Parameters
- \textit{nCard}  
  The zero-based index of the OASIS controller to use.
- \textit{AxisID}  
  The desired axis (see the introduction of this section for the appropriate constants).
- \textit{dValue}  
  The desired position for the move. This value is specified in microns.
- \textit{nWait}  
  A flag indicating whether to wait before the move is completed before returning.

Return Value
- OI_OK if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

Comments
See the section “Waiting for Movement Completion” above for a description for more information about the \textit{nWait} parameter.

See Also  
OI_MoveToXY, OI_MoveToZ, OI_MoveToF

\textbf{OI_ReadAxis}

Syntax
\begin{verbatim}
OI_API OI_ReadAxis(int AxisID, double* pdValue)
OI_API OI_ReadAxisEx(int nCard, int AxisID, double* pdValue)
\end{verbatim}

Description
Read the current axis position.

Parameters
- \textit{nCard}  
  The zero-based index of the card to use, in a multi-card configuration.
- \textit{AxisID}  
  The desired axis (see the introduction of this section for the appropriate constants).
- \textit{pdValue}  
  Returns the current position, in microns.

Return Value
- OI_OK if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

Comments
All coordinate values are returned in microns and are relative to the origin of the
See Also OI_ReadXY, OI_ReadZ, OI_ReadF, OI_InitializeXY, OI_InitializeZ, OI_InitializeF

---

### OI_ReadAxisAtLimit

<table>
<thead>
<tr>
<th>Syntax</th>
<th>OI_API OI_ReadAxisAtLimit(int AxisID, BOOL* pbAtNegLimit, BOOL* pbAtPosLimit)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Reads whether the axis is at the negative or positive limit of travel.</td>
</tr>
<tr>
<td>Parameters</td>
<td>AxisID</td>
</tr>
<tr>
<td></td>
<td>pbAtNegLimit</td>
</tr>
<tr>
<td></td>
<td>pbAtPosLimit</td>
</tr>
<tr>
<td>Return Value</td>
<td>OI_OK if successful.</td>
</tr>
<tr>
<td></td>
<td>If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.</td>
</tr>
<tr>
<td>Comments</td>
<td>The OI_ReadAxisAtLimit function is used to quickly determine with the axis is located at either a user or physical limit of travel.</td>
</tr>
<tr>
<td></td>
<td>For a full report of the status of an axis, including whether the physical or user limit is reached, use the OI_ReadAxisStatus function.</td>
</tr>
<tr>
<td>See Also</td>
<td>OI_ReadAxisStatus, OI_ReadAxisMoving</td>
</tr>
</tbody>
</table>

---

### OI_ReadAxisMoving

<table>
<thead>
<tr>
<th>Syntax</th>
<th>OI_API OI_ReadAxisMoving(int AxisID, BOOL* pbIsMoving)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Reads whether the axis is currently moving.</td>
</tr>
<tr>
<td>Parameters</td>
<td>AxisID</td>
</tr>
<tr>
<td></td>
<td>pbIsMoving</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Return Value
OI_OK if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

Comments
Use the OI_ReadAxisStatus function for a full report on the status of a given axis, including whether the axis is moving, has been initialised, is at a user or physical limit, and the limits have been set.

See Also
OI_ReadAxisStatus

---

**OI_ReadAxisRampValue**

Syntax
OI_API OI_ReadAxisRampValue(int AxisID, WORD wIndex, LPWORD lpwInterval, LPWORD lpwStepSize)

Description
Reads the acceleration ramp value at a given index.

Parameters

- **AxisID**
  The desired axis (see the introduction of this section for the appropriate constants).

- **wIndex**
  The ramp table index to be read.

- **lpwInterval**
  The returned interval, in microseconds, during which that index is applied.

- **lpwStepSize**
  The returned step size, in microsteps.

Return Value
Returns OI_OK if successful.

Returns OI_INVALIDARG if an out of range index value is passed.

Comments
Each axis is assigned one of three pre-defined acceleration / deceleration ramp tables in the OASIS hardware. The ramp table determines how acceleration and deceleration are accomplished, and also specifies the actual speeds to be used. The three pre-defined tables allow Normal, Slow, or Fast acceleration profiles.

The ramp table holds 512 values, leading to valid indices of 0 to 511.

See Also
OI_SetAxisRamp, OI_GetAxisRamp, OI_SetAxisCruise, OI_GetAxisCruise, OI_SetCruiseXY, OI_SetCruiseZ, OI_SetCruiseF

---

**OI_ReadAxisStatus**

Syntax
OI_API OI_ReadAxisStatus(int AxisID, LPWORD lpwStatus)

OI_API OI_ReadAxisStatusEx(int nCard, int AxisID, LPWORD lpwStatus)

return
Description
Reads the current status of an axis.

Parameters
- **nCard**
  The zero-based index of the card to use, in a multi-card configuration.
- **lpwStatus**
  Returns the axis status value.

Return Value
- **OI_OK** if successful.
  - If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

Comments
The returned status value can be a bit wise combination of the following values:

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_LIMIT_PHY_NEG</td>
<td>The axis is at the negative physical limit</td>
</tr>
<tr>
<td>S_LIMIT_USR_NEG</td>
<td>The axis is at the negative user limit</td>
</tr>
<tr>
<td>S_LIMIT_PHY_POS</td>
<td>The axis is at the positive physical limit</td>
</tr>
<tr>
<td>S_LIMIT_USR_POS</td>
<td>The axis is at the negative user limit</td>
</tr>
<tr>
<td>S_LIMIT_USR_NEG_SET</td>
<td>The user negative limit has been set</td>
</tr>
<tr>
<td>S_LIMIT_USR_POS_SET</td>
<td>The user positive limit has been set</td>
</tr>
<tr>
<td>S_INITIALIZED</td>
<td>The axis has been initialised</td>
</tr>
<tr>
<td>S_DIRECTION</td>
<td>If set, the direction of travel is negative</td>
</tr>
<tr>
<td>S_MOVING</td>
<td>The axis is moving</td>
</tr>
<tr>
<td>S_MOTOR_DETECTED</td>
<td>A motor was detected on the axis on startup.</td>
</tr>
</tbody>
</table>

See Also
- **OI_ReadStatusXY**, **OI_ReadStatusZ**, **OI_ReadStatusF**

---

**OI_SetAxisBacklash**

Syntax

```c
OI_API OI_SetAxisBacklash(int AxisID, BOOL bEnabled)
```

Description
Enables or disables backlash correction for a given axis.

Parameters
- **AxisID**
  The desired axis (see the introduction of this section for the appropriate constants).
- **bEnabled**
  Flag indicating whether backlash correction is enabled.
A value of TRUE enables backlash correction.

A value of FALSE disables backlash correction.

Return Value
OI_OK if successful.
If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

Comments
Backlash correction can improve positional accuracy by ensuring that each movement always approaches the desired position from the same direction.

See Also
OI_GetAxisBacklash

OI_SetAxisCruise

Syntax
OI_API OI_SetAxisCruise(int AxisID, int nCruise)

OI_API OI_SetAxisCruiseEx(int nCard, int AxisID, int nCruise)

Description
Sets the current maximum speed for a given axis.

Parameters
nCard
The zero-based index of the card to use, in a multi-card configuration.

AxisID
The desired axis (see the introduction of this section for the appropriate constants).

nCruise
The cruise speed table index.

Return Value
OI_OK if successful.
If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

Comments
Each axis is assigned an associated ramp table in the OASIS hardware. This ramp table determines how acceleration and deceleration are accomplished, and also specifies the actual speeds to be used.

The ramp table has 512 entries, indexed from 0 to 511. The OI_SetAxisCruise function specifies which index in the table will be used as the maximum speed at which axis is moved.

See Also
OI_GetAxisCruise, OI_SetAxisRamp, OI_ReadRampValue

OI_SetAxisEncoderEnabled

Syntax
OI_API OI_SetAxisEncoderEnabled(int AxisID, BOOL bEnabled, BOOL
bAutoCorrect )

**Description**
Sets the pre-defined acceleration / deceleration ramp table to use.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AxisID</td>
<td>The desired axis (see the introduction of this section for the appropriate constants).</td>
</tr>
<tr>
<td>bEnabled</td>
<td>Indicates whether the encoder counter is enabled.</td>
</tr>
<tr>
<td>bAutoCorrect</td>
<td>Indicates whether moves are automatically corrected to the nearest encoder position.</td>
</tr>
</tbody>
</table>

**Return Value**

OI_OK if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

See the OI_GetAxisEncoderEnabled function for more information on encoder enabling.

**See Also**

OI_GetAxisEncoderEnabled, OI_GetAxisEncoderFitted

---

**OI_SetAxisInitMethod**

**Syntax**

OI_API OI_SetAxisInitMethod (int AxisID, int nMethod)

**Description**
Sets the.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AxisID</td>
<td>The desired axis (see the introduction of this section for the appropriate constants).</td>
</tr>
<tr>
<td>nMethod</td>
<td>The method to use for initialising the axis.</td>
</tr>
</tbody>
</table>

**Return Value**

OI_OK if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

Normally, an axis initialisation using limit switches is performed by driving to each end of travel until the physical limit switch is encountered. Once both limits have been encountered, the full range of travel and the current position relative to these limits are known.

In some cases, it may be necessary or convenient to only use one set of limits to initialise the stage. For instance, some XY stages only provide one set of limits. Also, once the full range of stage travel is known, some time may be saved by driving to only one set of limits, using the known travel to define the soft limit settings at the opposite set of limits.

Use OI_SetAxisInitMethod to define how the axis is to determine the range of travel, as specified in the table below. Note that in the current version, this
function is only relevant to the X and Y axis for stage initialisation.

<table>
<thead>
<tr>
<th>nMethod</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Normal, seek limit switches at each end of travel</td>
</tr>
<tr>
<td>1</td>
<td>Seek only the limit switch at the negative limit of travel.</td>
</tr>
</tbody>
</table>

See Also OI_GetAxisInitMethod, OI_InitializeXY, OI_SetAxisTravel, OI_GetAxisTravel

**OI_SetAxisPitch**

**Syntax**

```c
OI_API OI_SetAxisPitch(int AxisID, double dPitchMM)
```

```c
OI_API OI_SetAxisPitch(int nCard, int AxisID, double dPitchMM)
```

**Description**

Sets the pre-defined acceleration / deceleration ramp table to use.

**Parameters**

- **nCard**
  - The zero-based index of the card to use, in a multi-card configuration.
- **AxisID**
  - The desired axis (see the introduction of this section for the appropriate constants).
- **dPitchMM**
  - The axis pitch, in mm, i.e., the expected travel per revolution of the motor.

**Return Value**

OI_OK if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

Calibration data for an axis is defined by the amount of travel expected for either one step or for each turn of the motor. For the later, the pitch value is used, defined in millimetres per turn.

See Also OI_GetAxisRamp, OI_SetAxisCruise, OI_GetAxisRampValue

**OI_SetAxisRamp**

**Syntax**

```c
OI_API OI_SetAxisRamp(int AxisID, int nRamp)
```

```c
OI_API OI_SetAxisRampEx(int nCard, int AxisID, int nRamp)
```

**Description**

Sets the pre-defined acceleration / deceleration ramp table to use.
Parameters

**nCard**
The zero-based index of the card to use, in a multi-card configuration.

**AxisID**
The desired axis (see the introduction of this section for the appropriate constants).

**nRamp**
An identifier indicating the ramp table to use, as described in the comments below.

Return Value

OI_OK if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

Comments

Each axis is assigned a ramp speed table that defines how the axis is to be accelerated and decelerated, as well as the desired cruise speed.

Three pre-defined tables may be selected for a given axis, as indicated by the **nRamp** parameter:

<table>
<thead>
<tr>
<th>nRamp value</th>
<th>Acceleration</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Slow</td>
</tr>
<tr>
<td>1</td>
<td>Medium</td>
</tr>
<tr>
<td>2</td>
<td>Fast</td>
</tr>
</tbody>
</table>

See Also  
OI_GetAxisRamp, OI_SetAxisCruise, OI_GetAxisRampValue

**OI_SetAxisSense**

Syntax

OI_API OI_SetAxisSense(int AxisID, int nSense)

Description

Sets the axis drive sense, indicating the physical direction of travel for positive or negative movements.

Parameters

**AxisID**
The desired axis (see the introduction of this section for the appropriate constants).

**nSense**
The desired direction of travel:

- A value of zero indicates standard movement.
- A non-zero value indicates reversed movement.

Return Value

OI_OK if successful.

If unsuccessful, a combination of error codes may be returned to indicate the
reason for failure.

Comments
The motor driving a given axis can be driven in either a clockwise or counterclockwise motion for a given deflection direction of the joystick. The drive sense parameter sets which direction of rotation is associated with a given joystick deflection direction.

See Also OI_GetAxisSense

### OI_SetAxisStepSize

**Syntax**

OI_API OI_SetAxisStepSize(int AxisID, double dStepSize)

**Description**
Sets the minimum step size for a given axis.

**Parameters**

- **AxisID**
  The desired axis (see the introduction of this section for the appropriate constants).

- **dStepSize**
  The size of each microstep, in microns.

**Return Value**

OI_OK if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**
Internally, the OASIS controller maintains positional information in terms of micro-steps. The OI_SetAxisStepSize function sets the actual distance in microns for each micro-step. This calibrates the distance values for the axis.

See Also OI_SetPitchXY, OI_GetAxisStepSize

### OI_SetAxisToDefaults

**Syntax**

OI_API OI_SetAxisToDefaults(int AxisID)

**Description**
Sets the parameters for a given axis to factory default settings.

**Parameters**

- **AxisID**
  The desired axis (see the introduction of this section for the appropriate constants).

**Return Value**

OI_OK if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**
This function re-sets the values for acceleration ramp, cruise speed, step size, backlash, and drive sense to default settings.
The default settings are:

<table>
<thead>
<tr>
<th>Setting</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acceleration Ramp</td>
<td>1 (Normal)</td>
</tr>
<tr>
<td>Cruise Speed</td>
<td>200</td>
</tr>
<tr>
<td>Step Size</td>
<td>0.078</td>
</tr>
<tr>
<td>Backlash</td>
<td>FALSE</td>
</tr>
<tr>
<td>Drive Sense</td>
<td>0 (Normal)</td>
</tr>
</tbody>
</table>

See Also: OI_SetAxisRamp, OI_SetAxisCruise, OI_SetAxisStepSize, OI_SetAxisBacklash, OI_SetAxisSense

**OI_SetAxisTravel**

**Syntax**

```
OI_API OI_SetAxisTravel (int AxisID, double dMin, double dMax)
```

**Description**

Sets the.

**Parameters**

- `AxisID` The desired axis (see the introduction of this section for the appropriate constants).
- `dMin` The value for the minimum available travel, in microns.
- `dMax` The value for the maximum available travel, in microns.

**Return Value**

- `OI_OK` if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

Each axis may have an associated range of travel, typically defined by the distance found between the physical limits of travel. For instance, for the X and Y axes, the range of travel is found automatically during the XY initialisation process, which drives to each end of travel to locate the physical limit switches. Once determined, the range of travel is defined to be the distance, in microns, between the switches.

The positions defining the ends of travel may also be set using the OI_SetAxisTravel function. This function is particularly useful when using the abbreviated XY initialisation process that uses only one set of limits at the minimum range of travel then sets up the software limits based on the range of travel for each axis.
OI_SetAxisUserLimits

Syntax
OI_API OI_SetAxisUserLimits ( int AxisID, double dMin, double dMax )

Description
Clears the user (software) limit values, i.e., makes them unset. The controller will no longer obey the user limits once this function is called.

Parameters
- AxisID: The desired axis (see the introduction of this section for the appropriate constants).
- dMin: The position to use as the minimum user limit.
- dMax: The position to use as the maximum user limit.

Return Value
- OI_OK if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

Comments
Each axis of the OASIS controller can be set to use software limits, specifying a minimum and maximum position to be used for the range of travel. When the user software limits are set, the controller will obey these positions as if they were physical limits of travel. The OI_SetAxisUserLimits function defines the user limit values.

See Also
- OI_GetAxisUserLimits, OI_ClearAxisUserLimits

OI_StepAxis

Syntax
- OI_API OI_StepAxis (int AxisID, double dValue, int nWait)
- OI_API OI_StepAxisEx (int nCard, int AxisID, double dValue, int nWait)

Description
Moves the axis a relative distance from the current position, waiting for completion if necessary.

Parameters
- nCard: The zero-based index of the card to use, in a multi-card configuration.
- AxisID: The desired axis (see the introduction of this section for the appropriate constants).
- dValue: The relative distance to move, in microns.
- nWait: A flag indicating whether the function waits for the
move to be completed before returning.

**Return Value**

OI_OK if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**
The distance travelled is specified in microns from the current position. Use negative distance value to specify movements in the negative direction (as defined by the current drive sense for the axis).

**See Also**
OI_MoveAxis, OI_StepX, OI_StepY, OI_StepXY, OI_StepZ, OI_StepF

---

### OI_StepAxisAbs

**Syntax**

```
OI_API OI_StepAxisAbs(int AxisID, long lSteps, int nWait)
OI_API OI_StepAxisAbsEx(int nCard, int AxisID, long lSteps, int nWait)
```

**Description**

Moves the axis a relative distance from the current position in microsteps, waiting for completion if necessary.

**Parameters**

- **nCard**
  The zero-based index of the card to use, in a multi-card configuration.

- **AxisID**
  The desired axis (see the introduction of this section for the appropriate constants).

- **lSteps**
  The relative distance to move, in microsteps.

- **nWait**
  A flag indicating whether the function waits for the move to be completed before returning.

**Return Value**

OI_OK if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**
The distance travelled is specified in microsteps from the current position. Use negative distance value to specify movements in the negative direction (as defined by the current drive sense for the axis). Note that the actual distance travelled will depend on both the microstepping resolution of the controller as well as the mechanics of the axis, such as the leadscrew pitch. Use OI_StepAxis to command a step in calibrated units, i.e., microns.

**See Also**
OI_StepAxis, OI_MoveAxis, OI_StepX, OI_StepY, OI_StepXY, OI_StepZ, OI_StepF
**OI_WaitForAxisStopped**

<table>
<thead>
<tr>
<th>Syntax</th>
<th>OI_API OI_WaitForAxisStopped( int AxisID )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>OI_API OI_WaitForAxisStoppedEx( int nCard, int AxisID )</td>
</tr>
</tbody>
</table>

**Description**

Waits for a given axis to stop moving before returning.

**Parameters**

- **nCard**
  - The zero-based index of the card to use, in a multi-card configuration.

- **AxisID**
  - The desired axis (see the introduction of this section for the appropriate constants).

**Return Value**

- **OI_OK** if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

This function will continuously read the status of the specified axis until the axis has stopped moving, the move times out, or the user aborts.

**See Also**

- **OI_MoveAxis**, **OI_StepX**, **OI_StepY**, **OI_StepXY**, **OI_StepZ**, **OI_StepF**

---

**Simultaneous Three Axis Control**

The following functions provide for simultaneous movements of multiple axes, for instance a move to a given XYZ location where the three axes are driven simultaneously.

**OI_DriveContinuousXYZ**

<table>
<thead>
<tr>
<th>Syntax</th>
<th>OI_API OI_DriveContinuousXYZ(int nXSspeed, int nYSspeed, int nZSpeed)</th>
</tr>
</thead>
</table>

**Description**

Drives the X and Y axes at continuous speeds.

**Parameters**

- **nXSspeed**
  - A signed integer indicating the direction and speed at which to drive the X axis.

- **nYSspeed**
  - A signed integer indicating the direction and speed at which to drive the Y axis.

- **nZSpeed**
  - A signed integer indicating the direction and speed at which to drive the Z axis.

**Return Value**

- **OI_OK** if successful.

If unsuccessful, a combination of error codes may be returned to indicate the
reason for failure.

**Comments**
The \textit{nXSpeed}, \textit{nYSpeed} and \textit{nZSpeed} parameters specify the desired speed of movement in half-steps per second.

The speed values are signed to indicate the direction of travel, i.e., a negative speed causes a continuous drive in the negative direction, and may be any integer in the range of \(-4096\) to \(+4096\).

To stop the continuous movement, use a corresponding call to the \texttt{OI_HaltXY} and/or \texttt{OI_HaltZ} function.

**Warning**
If the axis limits are not appropriately set for the physical limitations of the microscope, continuous movement of an axis could cause the collision of mechanical and optical components of the microscope system.

Caution should be taken by an application to ensure that appropriate safety mechanisms are in place to prevent damage to the optical system. Usually this means that safe user limits and/or limit switch positions for each axis have been set so as to prevent movements that would result in collisions.

The OASIS DLL provides a safety function, \texttt{OI_EmergencyStopAll}, to immediately stop all axes from being driven. An application should provide facilities allowing the user to effectively access this function in all appropriate situations in order to ensure hardware damage is prevented.

**See Also** \texttt{OI_HaltXY}, \texttt{OI_HaltZ}, \texttt{OI_DriveAxisContinuous}, \texttt{OI_DriveContinuousXY}, \texttt{OI_DriveContinuousZ}

**OI_HaltAllAxes**

**Syntax**
\texttt{OI_API OI_HaltAllAxes(void)}

**Description**
Immediately stops all axes, using deceleration ramps.

**Parameters**
None.

**Return Value**
\texttt{OI_OK} if successful.

**Comments**
This function uses deceleration ramps and therefore maintains positional accuracy.

**See Also** \texttt{OI_EmergencyStopAll}, \texttt{OI_HaltXY}, \texttt{OI_HaltZ}, \texttt{OI_HaltF}

**OI_MoveToXYZ**

**Syntax**
\texttt{OI_API OI_MoveToXYZ(double dX, double dY, double dZ, int nWait)}
Description
Performs a simultaneous move to given X, Y and Z positions.

Parameters
- **dX**
  The desired X-axis position, in microns.
- **dY**
  The desired Y-axis position, in microns.
- **dZ**
  The desired Z-axis position, in microns.
- **nWait**
  A flag indicating whether the function waits for the move to be completed before returning.

Return Value
- **OI_OK** if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

Comments
The **OI_MoveToXYZ** function performs a simultaneous 3-axis move. This function for instance can be used to perform stage and focus relocation to a given field of view, where the 3 coordinate values had been previously recorded.

The **nWait** parameter tells the function whether to return immediately (i.e., **nWait=0**) or to wait until all the moves are complete (i.e., **nWait** is not zero).

If you wish to return immediately from the function, but later wish to wait until the move is complete, use **OI_WaitForStoppedXYZ**. This is useful when using the OASIS controller to multitask movements with other functions.

For instance, in an image analysis scanning application, you may wish to acquire an image, then set the stage moving to a new XYZ location without waiting. While the stage is moving, the PC CPU can carry on processing the previously acquired image. Once that field has been processed, the stage may be either still moving to the new location or already there. A call to **OI_WaitForStoppedXYZ** can ensure that the new location is obtained for a new field of view is acquired for processing.

See Also
- **OI_MoveToXYZ_Auto**, **OI_MoveToXY**, **OI_MoveToZ**, **OI_WaitForStoppedXYZ**

---

**OI_MoveToXYZ_Auto**

Syntax
- **OI_API OI_MoveToXYZ_Auto(double dX, double dY, double dZ, int nWait)**

Description
Performs a simultaneous move to given X, Y and Z positions, applying an autofocus once there.

Parameters
- **dX**
  The desired X-axis position, in microns.
- **dY**
  The desired Y-axis position, in microns.
The desired Z-axis position, in microns.

The flag indicating whether the function waits for the move to be completed before returning.

Return Value

OI_OK if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

Comments

The OI_MoveToXYZ_Auto function performs a 3-axis simultaneous move to a given X, Y, and Z position. Once the position is reached, an automatic focus is applied using the current automatic focus settings.

To test whether the move is complete, use the OI_WaitForStoppedXYZ, OI_WaitForAutoFocus functions.

NOTE: An OASIS-AF hardware module is required for automatic focus operation.

See Also OI_MoveToXYZ, OI_WaitForStoppedXYZ, OI_SetAutoFocus

### OI_ReadMaxMoveXYZ

Syntax

OI_API OI_ReadMaxMoveXYZ (LPDWORD lpdwXSteps, LPDWORD lpdwYSteps, LPDWORD lpdwZSteps)

Description

Retrieves the current pre-defined ramp table in use for a given axis.

Parameters

- **lpdwXSteps** Returns the maximum allowable move for the X axis, as described in the Comments section below.
- **lpdwYSteps** Returns the maximum allowable move for the Y axis, as described in the Comments section below.
- **lpdwZSteps** Returns the maximum allowable move for the Z axis, as described in the Comments section below.

Return Value

OI_OK if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

Comments

The OASIS controller provides protection against extreme moves, e.g., moves that involve an unusually large distance for a typical microscope automation situation. If the user limits have not been properly defined, these moves could cause physical damage due to collisions such as the objective lens striking the specimen.

This protection is implemented as a maximum allowable move, defined in microsteps. The OI_ReadMaxMoveXYZ function allows the current setting for
X, Y, and Z axes to be returned.

The values for the maximum move for each axis are stored in the OASIS flash memory.

See Also  
OI_ReadMaxMoveXY, OI_ReadMaxMoveZ, OI_ReadMaxMoveF,  
OI_GetAxisMaxMove

**OI_ReadXYZ**

**Syntax**  
OI_API OI_ReadXYZ(double* pdX, double* pdY, double* pdZ)

**Description**  
Reads the current position of the X, Y and Z axes.

**Parameters**  
- *pdX*  
The current X-axis position, in microns.
- *pdY*  
The current Y-axis position, in microns.
- *pdZ*  
The current Z-axis position, in microns.

**Return Value**  
OI_OK if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**  
The coordinate values are specified in microns and are relative to the axis origin, which is normally set during the initialisation process for each axis.

See Also  
OI_MoveToXYZ, OI_ReadXY, OI_ReadZ, OI_ReadF, OI_InitializeXY, OI_InitializeZ

**OI_SetPitchFromFlashXYZ**

**Syntax**  
OI_API OI_SetPitchFromFlashXYZ()

**Description**  
Sets the pitch for the X, Y and Z axes based on the current pitch values found in the OASIS flash memory.

**Parameters**  
None.

**Return Value**  
OI_OK if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**  
The OI_SetPitchFromFlashXYZ function reads the current pitch definitions from the flash memory and sets up the X, Y and Z axes according to those values.
As part of the system configuration, and in particular when encoders are configured, the pitch value used for each axis is stored in flash memory. Axis pitch information may also be set via software calls such as `OI_SetPitchXY`, and such values will be stored in the system registry along with other software settings such as the current cruise speed, acceleration ramp selection, etc.

The function `OI_SetPitchFromFlashXYZ` is used to ensure the software settings for the pitch of the X, Y, and Z axes are setup to match the flash memory values. This is useful for instance in applications that rely on the OASIS flash configuration utility application to perform the system configuration, where the 3rd party software may not include facilities for defining the pitch.

See Also `OI_SetPitchXY`, `OI_SetPitchZ`, `OI_FlashReadAxisPitch`

---

**OI_SetPositionXYZ**

**Syntax**

```c
OI_API OI_SetPositionXYZ(double dX, double dY, double dZ)
```

**Description**

Sets the current position for the X, Y and Z axes.

**Parameters**

- `dX` The desired X-axis position, in microns.
- `dY` The desired Y-axis position, in microns.
- `dZ` The desired Z-axis position, in microns.

**Return Value**

- `OI_OK` if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

The `OI_SetPositionXYZ` function resets the coordinate system for each of the X, Y, and Z axes. The current X-Y-Z position is redefined to be the values passed in `dX`, `dY`, and `dZ`.

Note that the physical positions of the limits for each axis are retained by this function. That is, the `OI_SetPositionXYZ` function maintains the same relation between the current position and the position of the negative and positive soft limits for each axis. Therefore, the coordinates values associated for these limits will be changed if a new position value is specified for the axis.

See Also `OI_SetPositionXY`, `OI_SetPositionZ`, `OI_SetPositionF`, `OI_SetOriginXY`, `OI_SetOriginZ`, `OI_SetOriginF`, `OI_InitializeXY`, `OI_InitializeZ`, `OI_InitializeF`

---

**OI_WaitForStoppedXYZ**

**Syntax**

```c
OI_API OI_WaitForStoppedXYZ(int nXWait, int nYWait, int nZWait)
```
Description
Waits for the X, Y, and Z axes to stop moving.

Parameters

- **nXWait**: Test for X axis moving. Set to zero if X axis is not to be tested. Set to one if X axis movement is to be tested.
- **nYWait**: Test for Y axis moving. Set to zero if Y axis is not to be tested. Set to one if Y axis movement is to be tested.
- **nZWait**: Test for Z axis moving. Set to zero if Z axis is not to be tested. Set to one if Z axis movement is to be tested.

Return Value
OI_OK if successful.
If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

Comments
The **OI_WaitForStoppedXYZ** function is useful after any move functions are called with zero wait parameters. **OI_WaitForStoppedXYZ** will not return until the indicated axes have completed their moves.

See Also **OI_WaitForAutoFocus**

---

**XY Stage Control**

Motorised stage control is a primary application for the OASIS controller. The following functions allow simplified control of XY stages.

**OI_ClearUserLimitsXY**

Syntax
OI_API OI_ClearUserLimitsXY(void)

Description
Clears the user limits for the X and Y axes.

Parameters
None.

Return Value
OI_OK if successful.
If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

Comments
The positive and negative software limits for the X and Y axes will be cleared by this function. Only a physical limit will restrict the range of travel.
See Also  

OI_SetUserLimitsXY, OI_InitializeXY

---

**OI_DriveContinuousXY**

**Syntax**

```c
OI_API OI_DriveContinuousXY(int nXSpeed, int nYSpeed)
```

**Description**

Drives the X and Y axes at continuous speeds.

**Parameters**

- **nXSpeed**
  A signed integer indicating the direction and speed at which to drive the X axis.

- **nYSpeed**
  A signed integer indicating the direction and speed at which to drive the Y axis.

**Return Value**

- OI_OK if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

The `nXSpeed` and `nYSpeed` parameters specify the desired speed of movement in half-steps per second.

The speed values are signed to indicate the direction of travel, i.e., a negative speed causes a continuous drive in the negative direction, and may be any integer in the range of -4096 to +4096.

To stop the continuous movement, use a corresponding call to `OI_HaltXY` function.

**Warning**

If the axis limits are not appropriately set for the physical limitations of the microscope, continuous movement of an axis could cause the collision of mechanical and optical components of the microscope system.

Caution should be taken by an application to ensure that appropriate safety mechanisms are in place to prevent damage to the optical system. Usually this means that safe user limits and/or limit switch positions for each axis have been set so as to prevent movements that would result in collisions.

The OASIS DLL provides a safety function, `OI_EmergencyStopAll`, to immediately stop all axes from being driven. An application should provide facilities allowing the user to effectively access this function in all appropriate situations in order to ensure hardware damage is prevented.

**See Also**

OI_HaltXY, OI_DriveAxisContinuous, OI_DriveContinuousZ, OI_DriveContinuousF
OI_GetBacklashXY

Syntax

OI_API OI_GetBacklashXY(double* pdX, double* pdY)

Description
Reads the current X-Y backlash correction information of the stage.

Parameters

pdX
The current X-axis backlash correction, in microns.

pdY
The current Y-axis backlash, in microns.

Return Value

OI_OK if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

Comments
Backlash correction may be employed by the controller to help reduce mechanical inaccuracies due to direction changes.

If the stage is configured as the OASIS controller (default), the values returned are the current calibrated settings from the OASIS card’s flash memory.

If the stage is configured as the Leica IsoPro, the returned values are the interpolated backlash corrections for the current XY position.

See Also
OI_GetBacklashZ

OI_GetCruiseXY

Syntax

OI_API OI_GetCruiseXY(int* pnXCruise, int* pnYCruise)

Description
Retrieves the current cruise speed settings for the X and Y axes.

Parameters

pnXCruise
The returned X axis cruise speed index.

pnYCruise
The returned Y axis cruise speed index.

Return Value

OI_OK if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

Comments
See the Comments for OI_SetCruiseXY for more information regarding cruise speeds.

See Also
OI_SetCruiseXY, OI_SetAxisCruise, OI_GetAxisCruise, OI_SetCruiseZ, OI_SetCruiseF
OI_GetDriveSenseXY

Syntax

OI_API OI_GetDriveSenseXY(int* pnXDir, int* pnYDir)

Description

Retrieves the current direction of rotation settings for the X and Y axes.

Parameters

pnXDir The returned X axis drive sense.

pnYDir The returned Y axis drive sense.

Return Value

OI_OK if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

Comments

See the Comments for OI_SetDriveSenseXY for further information about drive sense.

See Also OI_SetDriveSenseXY, OI_SetAxisSense, OI_GetAxisSense

OI_GetFullTravelXY

Syntax

OI_API OI_GetFullTravelXY( double* pdXMin, double* pdXMax, double* pdYMin, double* pdYMax, BOOL* pbInit )

Description

Returns the full available travel of the XY stage.

Parameters

pdXMin The returned X axis minimum value

pdXMax The returned X axis maximum value

pdYMin The returned Y axis minimum value

pdYMax The returned Y axis maximum value

pbInit Flag indicating if the stage has been initialised before.

Return Value

OI_OK if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

Comments

Usually XY stages use physical limit switches to define the limits of travel. The OI_InitializeXY function is used to automatically drive to these limits, so that the full range of travel may be measured. The OI_GetFullTravelXY function returns the position of the limit switches, with respect to the current stage origin.

The pbInit parameter indicates whether the stage has ever been initialised. If it has not, the full limits of travel are undefined.
See Also  

OI_GetPitchXY

OI_GetPitchXY

Syntax  

OI_API OI_GetPitchXY(double* pdXPitch, double* pdYPitch)

Description  

Returns the current lead screw pitch settings for the X and Y axes.

Parameters  

pdXPitch  

The returned X axis lead screw pitch, in millimetres.

pdYPitch  

The returned Y axis lead screw pitch, in millimetres.

Return Value  

OI_OK if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

Comments  

See the Comments for the OI_SetPitchXY function for more information about lead screw pitches.

See Also  

OI_SetPitchXY, OI_SetAxisStepSize, OI_GetAxisStepSize

OI_GetRampXY

Syntax  

OI_API OI_GetRampXY(int* pnXRamp, int* pnYRamp)

Description  

Retrieves the current ramp in use by the X and Y axes.

Parameters  

pnXRamp  

The returned current X-axis ramp, as described in the Comments below.

pnYRamp  

The returned current Y-axis ramp, as described in the Comments below.

Return Value  

OI_OK if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

Comments  

Three pre-defined tables may be selected for a given axis, as indicated by the nRamp parameter:

<table>
<thead>
<tr>
<th>nRamp value</th>
<th>Acceleration</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Slow</td>
</tr>
<tr>
<td>1</td>
<td>Medium</td>
</tr>
</tbody>
</table>
See Also  OI_SetRampXY, OI_SetAxisRamp, OI_GetAxisRamp

OI_GetSpeedXY

Syntax  
OI_API OI_GetSpeedXY( double* pdXSpeed, double* pnYSpeed )

Description  
Retrieves the current speeds, in mm per second, in use by the X and Y axes.

Parameters  

pdXSpeed  The returned current X-axis speed, in mm/s.

pdYSpeed  The returned current Y-axis speed, in mm/s.

Return Value  
OI_OK if successful.
If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

Comments  
The OI_GetSpeedXY returns the actual drive speed of the stage corresponding to the cruise speed values. The speed is derived from calibration values (i.e., the stage leadscrew pitch), the current cruise speed values, and is returned in mm per second.

See Also  OI_SelectSpeedXY, OI_LookupSpeedXY

OI_GetUserLimitGuardDistanceXY

Syntax  
OI_API OI_GetUserLimitGuardDistanceXY (double* pdXDistanceMicrons, double* pdYDistanceMicrons )

Description  
Sets the.

Parameters  

pdXDistanceMicrons  The returned value for the X-axis physical to software limit buffer region, in microns.

pdYDistanceMicrons  The returned value for the Y-axis physical to software limit buffer region, in microns.

Return Value  
OI_OK if successful.
If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

Comments  
See the OI_SetUserLimitGuardDistanceXY for more information regarding user vs. physical limit setup.
OI_GetUserLimitsXY

Syntax

```c
OI_API OI_GetUserLimitsXY(double* pdXMin, double* pdXMax, double* pdYMin, double* pdYMax)
```

Description

Retrieves the current user limit settings for the X and Y axes.

Parameters

- `pdXMin` The minimum coordinate for the X axis, in microns.
- `pdXMax` The maximum coordinate for the X axis, in microns.
- `pdYMin` The minimum coordinate for the Y axis, in microns.
- `pdYMax` The maximum coordinate for the Y axis, in microns.

Return Value

- OI_OK if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

Comments

See the Comments for the `OI_SetUserLimitsXY` for more information about user limits.

See Also `OI_SetUserLimitsXY`, `OI_InitilizeXY`

---

OI_HaltXY

Syntax

```c
OI_API OI_HaltXY(void)
```

Description

Stops any motion of the X and Y axes.

Parameters

None.

Return Value

- OI_OK if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

Comments

The `OI_Halt` function uses the currently defined deceleration ramp to achieve an accurate halt of any motion. This preserves positional accuracy.

To immediately stop all axes from moving, use the `OI_EmergencyStopAll` function.
See Also  OI_EmergencyStopAll, OI_HaltAxis, OI_HaltZ, OI_HaltF, OI_DriveContinuousXY

### OI_InitializeXY

**Syntax**

OI_API OI_InitializeXY(void)

**Description**

Initialises the stage by automatically finding limit switches and positioning the stage to the centre of the available range of travel in X and Y.

**Parameters**

None.

**Return Value**

OI_OK if successful.

OI_ABORT will be returned if the user aborts the initialisation by pressing the ESC key or CTRL-C during the procedure.

**Comments**

Stage initialisation is necessary for determining the precise range of travel available in the X and Y directions of travel.

To prevent over-travel into the mechanical ends of the lead screws, stages are fitted with limit switches that give a signal to the controller that the end of travel has been reached. The OASIS controller senses these signals, and these can be used to allow the controller to automatically determine the available range of travel on a microscope.

The initialisation procedure is defined as:

1. Move towards the negative direction until the negative physical limit switches for the X and Y axes are found;
2. Move towards the positive direction until the positive physical limit switches for the X and Y axes are found;
3. Move to the centre of the stage, defined as the midpoint between the found limit positions.

This function automatically sets the user limits to be just inside the physical limit switch positions and the stage $[X,Y] = [0,0]$ to be at the negative user limit.

Note that some stages are fitted with adjustable limit switches, and any change to the position of these switches will require another stage initialisation.

**See Also**  OI_InitializeZ, OI_InitializeF

### OI_LookupSpeedXY

**Syntax**

OI_API OI_LookupSpeedXY( int nCruiseX, int nCruiseY, double*
pdSpeedX, double* pdSpeedY )

**Description**
Retrieves the speeds, in mm per second, corresponding to a given cruise value for the Z axis.

**Parameters**
- `nCruiseX`, `nCruiseY` The cruise speeds of the X and Y axes for which the actual speed is desired.
- `pdSpeedX`, `pdSpeedY` The returned X and Y axes speeds, in mm/s, for the given cruise speeds.

**Return Value**
- OI_OK if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**
The `OI_LookupSpeedXY` returns the actual drive speeds of the X and Y axes corresponding to a given cruise speed. The speed is derived from calibration value (i.e., the leadscrew pitch values), the current cruise speed values, and is returned in mm per second.

Unlike the `OI_GetSpeedXY` function, which returns the speed corresponding to the currently selected cruise speeds, the `OI_LookupSpeedXY` function returns the speeds for a specified cruise value.

**See Also**
- `OI_GetSpeedXY`, `OI_SelectSpeedXY`, `OI_GetCruiseXY`

### OI_MoveToXY

**Syntax**
```c
OI_API OI_MoveToXY(double dX, double dY, int nWait)
```

**Description**
Moves to the specified X-Y position.

**Parameters**
- `dX` The desired X-axis position, in microns.
- `dY` The desired Y-axis position, in microns.
- `nWait` A flag indicating whether the function waits for the move to be completed before returning.

**Return Value**
- OI_OK if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**
The coordinate values are specified in microns and are relative to the axis origin, which is normally set during the initialisation process for each axis.

**See Also**
- `OI_ReadXY`, `OI_StepXY`, `OI_StepX`, `OI_StepY`, `OI_WaitForStoppedXYZ`
**OI_MoveToXY_Abs**

**Syntax**

OI_API OI_MoveToXY_Abs(long lX, long lY, int nWait)

**Description**

Moves to the specified X-Y position, in microsteps.

**Parameters**

- **lX**: The desired X-axis position, in microsteps.
- **lY**: The desired Y-axis position, in microsteps.
- **nWait**: A flag indicating whether the function waits for the move to be completed before returning.

**Return Value**

OI_OK if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

The coordinate values are specified in microsteps and are relative to the axis origin, which is normally set during the initialisation process for each axis.

**See Also**

OI_ReadXY_Abs, OI_MoveToXY, OI_ReadXY, OI_StepXY, OI_StepX, OI_StepY, OI_WaitForStoppedXYZ

---

**OI_MoveToXY_Auto**

**Syntax**

OI_API OI_MoveToXY_Auto(double dX, double dY, int nWait)

**Description**

Moves to the specified X-Y position, followed by an automatic focus.

**Parameters**

- **dX**: The desired X-axis position, in microns.
- **dY**: The desired Y-axis position, in microns.
- **nWait**: A flag indicating whether the function waits for the move to be completed before returning.

**Return Value**

OI_OK if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

The OI_MoveToXY_Auto function performs a 2-axis simultaneous move to a given X-Y position. Once the position is reached, an automatic focus is applied using the current automatic focus settings.

To test whether the move is complete, use the OI_WaitForStoppedXYZ OI_WaitForAutoFocus functions.

NOTE: An OASIS-AF hardware module is required for automatic focus.
operation.

See Also  OI_MoveToXY, OI_MoveToXYZ_Auto, OI_WaitForStoppedXYZ, OI_WaitForAutoFocus

OI_ReadLimitAlarmsXY

Syntax  OI_API OI_ReadLimitAlarmsXY(int* pnXNeg, int* pnXPos, int* pnYNeg, int* pnYPos)

Description  Reads the current status of the X and Y axes limit alarms.

Parameters  

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pnXNeg</td>
<td>Status of the negative limit for the X axis.</td>
</tr>
<tr>
<td>pnXPos</td>
<td>Status of the positive limit for the X axis.</td>
</tr>
<tr>
<td>pnYNeg</td>
<td>Status of the negative limit for the Y axis.</td>
</tr>
<tr>
<td>pnYPos</td>
<td>Status of the positive limit for the Y axis.</td>
</tr>
</tbody>
</table>

Return Value  

- OI_OK if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

Comments  
The OI_ReadLimitAlarmsXY functions tells you whether the X or Y axis is currently at a user (software) or hardware limit.

The returned status values in the arguments can be the following:

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The axis is not at the limit</td>
</tr>
<tr>
<td>1</td>
<td>The axis is at a user limit</td>
</tr>
<tr>
<td>2</td>
<td>The axis is at a hardware limit</td>
</tr>
<tr>
<td>3</td>
<td>The axis is at both a user and a hardware limit</td>
</tr>
</tbody>
</table>

See Also  OI_ReadStatusXY, OI_ReadLimitAlarmsZ, OI_ReadLimitAlarmsF

OI_ReadStatusXY

Syntax  OI_API OI_ReadStatusXY(LPWORD lpwXStatus, LPWORD lpwYStatus)
Description
Reads the current status of the X and Y axes.

Parameters
- lpwXStatus
  Returns the X axis status value.
- lpwYStatus
  Returns the Y axis status value.

Return Value
OI_OK if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

Comments
The returned status value can be a combination of the following values:

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_LIMIT_PHY_NEG</td>
<td>The axis is at the negative physical limit</td>
</tr>
<tr>
<td>S_LIMIT_USR_NEG</td>
<td>The axis is at the negative user limit</td>
</tr>
<tr>
<td>S_LIMIT_PHY_POS</td>
<td>The axis is at the positive physical limit</td>
</tr>
<tr>
<td>S_LIMIT_USR_POS</td>
<td>The axis is at the negative user limit</td>
</tr>
<tr>
<td>S_LIMIT_USR_NEG_SET</td>
<td>The user negative limit has been set</td>
</tr>
<tr>
<td>S_LIMIT_USR_POS_SET</td>
<td>The user positive limit has been set</td>
</tr>
<tr>
<td>S_INITIALIZED</td>
<td>The axis has been initialised</td>
</tr>
<tr>
<td>S_DIRECTION</td>
<td>If set, the direction of travel is negative</td>
</tr>
<tr>
<td>S_MOVING</td>
<td>The axis is moving</td>
</tr>
</tbody>
</table>

See Also
OI_ReadStatusZ, OI_ReadStatusF

**OI_ReadXY**

Syntax
OI_API OI_ReadXY(double* pdX, double* pdY)

Description
Reads the current X-Y position of the stage.

Parameters
- pdX
  The current X-axis position, in microns.
- pdY
  The current Y-axis position, in microns.

Return Value
OI_OK if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.
Comments
The coordinate values are returned in microns and are relative to the axis origin, which is normally set during the initialisation process for each axis.

See Also
OI_MoveToXY, OI_ReadZ, OI_ReadF, OI_ReadAxis

---

### OI_ReadXY_Abs

**Syntax**

OI_API OI_ReadXY_Abs(long* plX, long* plY)

**Description**

Reads the current X-Y position of the stage, in microsteps.

**Parameters**

- `plX`: The current X-axis position, in microsteps.
- `plY`: The current Y-axis position, in microsteps.

**Return Value**

- OI_OK if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

The coordinate values are returned in microsteps and are relative to the axis origin, which is normally set during the initialisation process for each axis.

See Also
OI_ReadXY, OI_MoveToXY, OI_ReadZ, OI_ReadF, OI_ReadAxis

---

### OI_SelectSpeedXY

**Syntax**

OI_API OI_SelectSpeedXY(double dXmmPerSec, double dYmmPerSec, int nFlags)

**Description**

Automatically selects the cruise speed corresponding to a desired speed in mm per second.

**Parameters**

- `dXmmPerSec`: The desired speeds for the X and Y axis, in mm per second.
- `dYmmPerSec`: The desired speeds for the X and Y axis, in mm per second.
- `nFlags`: Specifies how the search is performed, as described in the Comments below.

**Return Value**

- OI_OK if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

The OI_SelectSpeedXY function is used to automatically set the stage cruise speeds to specified actual speed targets, in mm per second.
The \textit{nFlags} parameter specifies how the search is to be carried out:

<table>
<thead>
<tr>
<th>\textit{nFlags} value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A cruise value is found that gives an actual speed as close to, but not exceeding, the desired speed.</td>
</tr>
<tr>
<td>1</td>
<td>A cruise value is found that gives the closest actual speed to the desired speed, including speeds that are greater than the desired speed.</td>
</tr>
</tbody>
</table>

The net effect of the \texttt{OI\_SelectSpeedXY} function is equivalent to a call to \texttt{OI\_SetCruiseXY} with parameters that give the best match to the desired actual speeds.

Note that you may use the \texttt{OI\_GetSpeedXY} and the \texttt{OI\_GetCruiseXY} to read the actual speeds and cruise values that have been selected.

\textbf{See Also} \texttt{OI\_GetSpeedXY}, \texttt{OI\_GetCruiseXY}, \texttt{OI\_SetCruiseXY}

\textbf{\texttt{OI\_SetCruiseXY}}

\textbf{Syntax}

\texttt{OI\_API OI\_SetCruiseXY(int nXCruise, int nYCruise)}

\textbf{Description}

Sets the cruising speed for the X and Y axes.

\textbf{Parameters}

- \texttt{nXCruise} The X axis cruise speed index.
- \texttt{nYCruise} The Y axis cruise speed index.

\textbf{Return Value}

- \texttt{OI\_OK} if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

\textbf{Comments}

The cruise speed is specified via the maximum index to be used in the currently defined acceleration / deceleration ramp for a given axis.

Each axis is assigned an associated ramp table in the OASIS hardware. This ramp table determines how acceleration and deceleration are accomplished, and also specifies the actual speeds to be used.

The ramp table has 512 entries, indexed from 0 to 511. The \texttt{OI\_SetCruiseXY} function specifies which index in the table will be used as the maximum speed at which the X and Y axes are moved.
See Also  
OI_GetCruiseXY, OI_SetAxisCruise, OI_SetCruiseZ, OI_SetCruiseF,  
OI_SetRampXY

OI_SetDriveSenseXY

Syntax  
OI_API OI_SetDriveSenseXY(int nXDir, int nYDir)

Description  
Sets the direction of rotation for movements of the X and Y axes.

Parameters  
nXDir  
The X axis drive sense.

nYDir  
The Y axis drive sense.

Return Value  
OI_OK if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

Comments  
The motor driving a given axis can be driven in either a clockwise or counter-clockwise motion. The drive sense parameter sets which direction of rotation is associated with positive valued movements.

A value of zero (0) indicates standard movement.

A non-zero value indicates reversed movement.

See Also  
OI_GetDriveSenseXY, OI_SetAxisSense, OI_GetAxisSense

OI_SetOriginXY

Syntax  
OI_API OI_SetOriginXY(void)

Description  
Sets the current XY position to be the origin (e.g., 0,0).

Parameters  
None.

Return Value  
OI_OK if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

Comments  
The OI_SetOriginXY function is used to establish the origin of the overall, micron-based coordinate system of the XY stage.

The origin is defined to be position [X=0,Y=0], and all stage positions are made relative to this origin.

By default, when the OI_InitializeXY function is used to initialise the range of
travel available to the stage, the origin is set just inside of the XY negative limit
switches, at the negative user limits automatically set by the **OI_InitializeXY**
function.

The **OI_SetOriginXY** function may be used to set the stage origin to another
user-defined position.

**Warning**
The **OI_SetOriginXY** function re-sets the entire coordinate system for the
stage. After a call to this function, previously stored position values may no
longer correspond to their associated physical stage positions.

**See Also**
**OI_InitializeXY**

---

**OI_SetPitchXY**

**Syntax**

```c
OI_API OI_SetPitchXY(double dXPitch, double dYPitch)
```

**Description**
Sets the pitch of the lead screws for the X and Y axes.

**Parameters**

- `dXPitch` The X axis lead screw pitch, in millimetres.
- `dYPitch` The Y axis lead screw pitch, in millimetres.

**Return Value**

- **OI_OK** if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the
  reason for failure.

**Comments**
The **OI_SetPitchXY** function is used to internally calculate the actual size of
each micro-step. Typically, there are 12,800 micro-steps per revolution of the
lead screw. From the supplied lead screw pitches, the **OI_SetPitchXY** function
will automatically calculate this minimum step size for you.

**NOTE:** All micron to micro-step conversions use these values for their
 calibration. It is critical that these values be correctly supplied in order to ensure
 accurate stage movement.

To retrieve the current step size value, you may use the **OI_GetAxisStepSize**
function.

Consult the specifications for your specific stage to determine the actual lead
screw pitches.

**See Also**
**OI_GetPitchXY**, **OI_SetAxisStepSize**, **OI_GetAxisStepSize**

---

**OI_SetPositionXY**

**Syntax**

```c
OI_API OI_SetPositionXY(double dX, double dY)
```
Description
Sets the current position for the X and Y axes.

Parameters
- \( dX \)  
  The desired X-axis position, in microns.
- \( dY \)  
  The desired Y-axis position, in microns.

Return Value
- OI_OK if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

Comments
The OI_SetPositionXY function resets the coordinate system for each of the X and Y axes. The current X and Y positions are redefined to be the values passed in \( dX \) and \( dY \).

Note that the physical positions of the limits for each axis are retained by this function. That is, the OI_SetPositionXY function maintains the same relation between the current position and the position of the negative and positive soft limits for each axis. Therefore, the coordinates values associated for these limits will be changed if a new position value is specified for the axis.

See Also
- OI_SetPositionXYZ, OI_SetPositionZ, OI_SetPositionF, OI_SetOriginXY, OI_SetOriginZ, OI_SetOriginF, OI_InitializeXY, OI_InitializeZ, OI_InitializeF

---

**OL_SetRampXY**

Syntax
OI_API OI_SetRampXY(int nXRamp, int nYRamp)

Description
Sets which pre-defined acceleration / deceleration ramp is used for the X and Y axes.

Parameters
- \( nXRamp \)  
  The X-axis ramp code.
- \( nYRamp \)  
  The Y-axis ramp code.

Return Value
- OI_OK if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

Comments
Three pre-defined tables may be selected for a given axis, as indicated by the \( nRamp \) parameter:

<table>
<thead>
<tr>
<th>( nRamp ) value</th>
<th>Acceleration</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Slow</td>
</tr>
<tr>
<td>1</td>
<td>Medium</td>
</tr>
<tr>
<td>2</td>
<td>Fast</td>
</tr>
</tbody>
</table>
See Also OI_GetRampXY, OI_SetAxisRamp, OI_GetAxisRamp

OI_SetUserLimitGuardDistanceXY

Syntax

OI_API OI_SetUserLimitGuardDistanceXY( double dXDistanceMicrons,
                                    double dYDistanceMicrons )

Description

Sets the.

Parameters

- dXDistanceMicrons: The value for the desired distance between the X-axis physical and soft limits, in microns, to be used when initialising the XY stage. Set to a negative value for automatic setting based on the current cruise speed and ramp.

- dYDistanceMicrons: The value for the desired distance between the Y-axis physical and soft limits, in microns, to be used when initialising the XY stage. Set to a negative value for automatic setting based on the current cruise speed and ramp.

Return Value

- OI_OK if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

Comments

When initialising the XY stage, the OASIS DLL uses the detected physical limit switch positions at each end of travel to set up the distance between the detected physical limit position and the software limits. This distance provides a “guard” buffer region allowing the OASIS controller to properly ramp to a stop once the software limit it detected before actually encountering the physical limit.

By default, the guard distance between the software and physical limits is set automatically using the current cruise speed and ramp values for the X and Y axes to ensure a proper buffer for ramping down. However, the OI_SetUserLimitGuardDistanceXY function may be used to define your own specific guard region values for X and Y.

To enable automatic detection of the guard region, use a negative value for the distance for the desired axis.

See Also OI_GetUserLimitGuardDistanceXY, OI_SetAxisInitMethod, OI_InitializeXY, OI_GetAxisTravel

OI_SetUserLimitsXY

Syntax

OI_API OI_SetUserLimitsXY(double dXMin, double dXMax, double dYMin,
**double dYMax)**

**Description**  
Sets user-defined limits of travel along the X and Y axes.

**Parameters**  
- **dXMin**  
The minimum coordinate for the X axis, in microns.
- **dXMax**  
The maximum coordinate for the X axis, in microns.
- **dYMin**  
The minimum coordinate for the Y axis, in microns.
- **dYMax**  
The maximum coordinate for the Y axis, in microns.

**Return Value**  
- **OI_OK** if successful.
  
If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**  
The `OI_SetUserLimitsXY` functions allows “soft” limits to be set at any point along the X and Y axes. Once the soft limits are set, the OASIS controller will not allow any movement outside of these limit values.

Note that the soft limits are distinct from the physical limit switches of the stage. The “hard” physical limit switches provide direct electronic feedback to the OASIS controller indicating the physical limits of travel available for the stage.

When using the `OI_InitializeXY` function to initialise the range of travel and position of the stage, the OASIS controller automatically sets the X and Y user limits to positions a short distance inside the actual physical limits. This distance is matched to the current deceleration ramp to prevent driving of the stage into the physical limit switches during normal operation.

**See Also**  
- `OI_GetUserLimitsXY`
- `OI_InitializeXY`
- `OI_SetRampXY`

---

**OI_StepX**

**Syntax**  
`OI_API OI_StepX(double dXDistance, int nWait)`

**Description**  
Moves a relative distance along the X axis.

**Parameters**  
- **dXDistance**  
The desired distance to move, in microns.
- **nWait**  
A flag indicating whether the function waits for the move to be completed before returning.

**Return Value**  
- **OI_OK** if successful.
  
If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**  
The `dXDistance` parameter should be negative for moves in the negative direction.
See Also  
OI_StepY, OI_StepXY, OI_MoveToXY, OI_StepAxis, OI_WaitForStoppedXYZ

**OI_StepXY**

**Syntax**

OI_API OI_StepX(double dXDistance, double dYDistance, int nWait)

**Description**

Moves a relative distance along the X and Y axes.

**Parameters**

- **dXDistance**  
The desired distance to move the X axis, in microns.
- **dYDistance**  
The desired distance to move the Y axis, in microns.
- **nWait**  
A flag indicating whether the function waits for the move to be completed before returning.

**Return Value**

- **OI_OK** if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

The `dXDistance` or `dYDistance` parameter should be negative for moves in the negative direction.

**See Also**

- OI_StepX, OI_StepY, OI_MoveToXY, OI_StepAxis, OI_WaitForStoppedXYZ

**OI_StepY**

**Syntax**

OI_API OI_StepY(double dYDistance, int nWait)

**Description**

Moves a relative distance along the Y axis.

**Parameters**

- **dYDistance**  
The desired distance to move, in microns.
- **nWait**  
A flag indicating whether the function waits for the move to be completed before returning.

**Return Value**

- **OI_OK** if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

The `dYDistance` parameter should be negative for moves in the negative direction.

**See Also**

- OI_StepX, OI_StepXY, OI_MoveToXY, OI_StepAxis, OI_WaitForStoppedXYZ
### OI_WaitForStoppedXY

<table>
<thead>
<tr>
<th>Syntax</th>
<th>OI_API OI_WaitForStoppedXY()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Waits for the X and Y axes to stop moving.</td>
</tr>
<tr>
<td>Parameters</td>
<td>None.</td>
</tr>
<tr>
<td>Return Value</td>
<td>OI_OK if successful.</td>
</tr>
</tbody>
</table>

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**  
The `OI_WaitForStoppedXY` function is useful after any XY stage move functions that are called with zero wait parameters. `OI_WaitForStoppedXY` will not return until the indicated axes have completed their moves or have timed out waiting.

**See Also**  
`OI_WaitForStoppedXYZ`, `OI_WaitForStoppedZ`, `OI_WaitForStoppedF`, `OI_WaitForAutoFocus`, `OI_SetDefaultAbortKeys`

---

### Z / Focus Control

---

### OI_ClearUserLimitsZ

<table>
<thead>
<tr>
<th>Syntax</th>
<th>OI_API OI_ClearUserLimitsZ(void)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Clears the user limits for the Z axis.</td>
</tr>
<tr>
<td>Parameters</td>
<td>None.</td>
</tr>
<tr>
<td>Return Value</td>
<td>OI_OK if successful.</td>
</tr>
</tbody>
</table>

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**  
The positive and negative software limits for the Z axis will be cleared by this function. Only a physical limit will restrict the range of travel.

**See Also**  
`OI_SetUserLimitsZ`, `OI_InitializeZ`
### OI_CloseMouseWheelForFocus

**Syntax**

```c
OI_API OI_CloseMouseWheelForFocus(void)
```

**Description**

Disables mouse wheel control of the focus.

**Parameters**

None.

**Return Value**

- **OI_OK** if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

See the `OI_OpenMouseWheelForFocus` function for more information about mouse wheel control of the focus.

**See Also**

- `OI_OpenMouseWheelForFocus`

---

### OI_DriveContinuousZ

**Syntax**

```c
OI_API OI_DriveContinuousZ(int nSpeed)
```

**Description**

Drives the Z axis at a continuous speed.

**Parameters**

- **nSpeed**
  
  A signed integer indicating the direction and speed at which to drive the Z axis.

**Return Value**

- **OI_OK** if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

- The `nSpeed` parameter specifies the desired speed of movement in half-steps per second.
- The speed values are signed to indicate the direction of travel, i.e., a negative speed causes a continuous drive in the negative direction, and may be any integer in the range of −4096 to +4096.
- To stop the continuous movement, use a corresponding call to either the `OI_HaltZ` or the `OI_EmergencyStopAll` function.

**Warning**

- If the Z axis limits are not appropriately set for a microscope focus mechanism, continuous movement in Z could drive the specimen up into the objective lens or down into the condenser optics.

Caution should be taken by an application to ensure that appropriate safety mechanisms are in place to prevent damage to the optical system. Usually this means that safe user limits for the Z axis have been set so as to prevent Z movements that would result in collision of the stage and/or specimen with other components of the optical system.
The OASIS DLL provides a safety function, `OI_EmergencyStopAll`, to immediately stop all axes from being driven. An application should provide facilities allowing the user to effectively access this function in all appropriate situations in order to ensure hardware damage is prevented.

See Also `OI_HaltZ, OI_EmergencyStopAll`

### OI_GetBacklashZ

**Syntax**

```
OI_API OI_GetBacklashZ(double* pdZ,)
```

**Description**

Reads the current Z backlash correction information of the focus.

**Parameters**

- `pdZ` The current Z-axis backlash correction, in microns.

**Return Value**

- `OI_OK` if successful.

- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

Backlash correction can be employed by the controller to help reduce mechanical inaccuracies due to direction changes.

The values are the current settings from the OASIS card’s flash memory, calibrated to microns.

See Also `OI_GetBacklashZ`

### OI_GetCruiseZ

**Syntax**

```
OI_API OI_GetCruiseZ(int* pnZCruise)
```

**Description**

Retrieves the current Z axis cruise speed index.

**Parameters**

- `pnZCruise` The returned Z axis cruise speed index.

**Return Value**

- `OI_OK` if successful.

- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

See the Comments for the `OI_SetCruiseZ` function for more information about cruise speeds.

See Also `OI_SetCruiseZ`
**OI_GetDriveSenseZ**

**Syntax**

```c
OI_API OI_GetDriveSenseZ(int* pnZDir)
```

**Description**

Retrieves the current direction of rotation setting for the Z axis.

**Parameters**

- `pnZDir`:
  The returned Z axis drive sense.

**Return Value**

- **OI_OK** if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

See the Comments for `OI_SetDriveSenseZ` for more information about the values for drive sense.

**See Also**

`OI_SetDriveSenseZ`, `OI_GetAxisSense`

**OI_GetMouseWheelPars**

**Syntax**

```c
OI_API OI_GetMouseWheelPars(double *pdStepSize, int *pnSpeed)
```

**Description**

Retrieves the current settings for control of the Z axis using the mouse wheel.

**Parameters**

- `pdStepSize`:
  The step size, in microns, for each step of the mouse wheel.

- `pnSpeed`:
  The desired maximum speed of movement to continuous driving when the mouse wheel is pressed.

**Return Value**

- **OI_OK** if successful.

**Comments**

See the `OI_SetMouseWheelPars` function for more information about the mouse wheel parameters.

**See Also**

`OI_SetMouseWheelPars`, `OI_OpenMouseWheelForFocus`, `OI_CloseMouseWheelForFocus`

**OI_GetMouseWheelZ**

**Syntax**

```c
OI_API OI_GetMouseWheelZ(BOOL *pbEnabled)
```

**Description**

Retrieves whether mouse wheel control of the Z axis is enabled.
Parameters  $pbEnabled$  Returns TRUE if enable, FALSE if disabled.

Return Value  OI_OK if successful.

Comments  See the OI_SetMouseWheelZ function for more information about enabling/disabling mouse wheel control of the Z axis.

See Also  OI_GetMouseWheelPars, OI_OpenMouseWheelForFocus, OI_CloseMouseWheelForFocus

---

**OI_GetRampZ**

Syntax  

$$\text{OI\_API OI\_GetRampZ(int* pnZRamp)}$$

Description  Retrieves which pre-defined acceleration / deceleration ramp is in use for the Z axis.

Parameters  $pnZRamp$  Indicates which pre-defined ramp is currently in use for the Z axis.

Return Value  OI_OK if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

Comments  Three pre-defined tables may be selected for a given axis, as indicated by the $nRamp$ parameter:

<table>
<thead>
<tr>
<th>$nRamp$ value</th>
<th>Acceleration</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Slow</td>
</tr>
<tr>
<td>1</td>
<td>Medium</td>
</tr>
<tr>
<td>2</td>
<td>Fast</td>
</tr>
</tbody>
</table>

See Also  OI_SetRampZ, OI_SetAxisRamp

---

**OI_GetSpeedZ**

Syntax  

$$\text{OI\_API OI\_GetSpeedZ( double* pdSpeed )}$$

Description  Retrieves the current speeds, in mm per second, in use by the Z axis.

Parameters  $pdSpeed$  The returned current Z-axis speed, in mm/s.
Return Value
OI_OK if successful.
If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

Comments
The OI_GetSpeedZ returns the actual drive speed of the focus control corresponding to the cruise speed values. The speed is derived from calibration value (i.e., the focus pitch or microns per step size), the current cruise speed value, and is returned in mm per second.

See Also OI_SelectSpeedZ, OI_LookupSpeedZ

---

### OI_GetUserLimitsZ

**Syntax**

```c
OI_API OI_GetUserLimitsZ(double* pdZMin, double* pdZMax)
```

**Description**

Retrieves the current user limit settings for the Z axis.

**Parameters**

- `pdZMin` The minimum coordinate for the Z axis, in microns.
- `pdZMax` The maximum coordinate for the Z axis, in microns.

**Return Value**

- OI_OK if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

See the Comments for the OI_SetUserLimitsZ for more information about user limits.

**See Also** OI_SetUserLimitsZ, OI_InitializeZ

---

### OI_HaltZ

**Syntax**

```c
OI_API OI_HaltZ(void)
```

**Description**

Stops any motion of the Z axis.

**Parameters**

None.

**Return Value**

- OI_OK if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

The OI_HaltZ functions uses the Z axis’s deceleration ramp to stop the stage smoothly.
To immediately stop the stage without using the deceleration ramp, use the `OI_EmergencyStopAll` function.

**See Also**  
`OI_EmergencyStopAll`

---

### OI_InitiateZ

**Syntax**  
`OI_API OI_InitializeZ(double dZRangeAbove, double dZRangeBelow)`

**Description**  
Initialises the Z axis for focus control.

**Parameters**  
- `dZRangeAbove`  
  The desired allowable distance above (e.g., in the positive direction) the current position, in microns.
- `dZRangeBelow`  
  The desired allowable distance above (e.g., in the negative direction) the current position, in microns.

**Return Value**  
`OI_OK` if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**  
The current position is set to zero, and the Z user limits are set to the given ranges above and below the current position.

**See Also**  
`OI_InitializeXY, OI_ReadRangeZ`

---

### OI_InitiateZLimits

**Syntax**  
`OI_API OI_InitializeZ()`

**Description**  
Initialises the Z axis for focus control using physical limit switches.

**Parameters**  
None.

**Return Value**  
`OI_OK` if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

**See Also**  
`OI_InitializeZ, OI_InitializeXY, OI_ReadRangeZ`
**OI_LookupSpeedZ**

**Syntax**

```
OI_API OI_LookupSpeedZ( int nCruise, double* pdSpeed )
```

**Description**

Retrieves the speeds, in mm per second, corresponding to a given cruise value for the Z axis.

**Parameters**

- `nCruise`
  
  The cruise speed for which the actual speed is desired.

- `pdSpeed`
  
  The returned Z-axis speed, in mm/s, for the given cruise speed.

**Return Value**

- `OI_OK` if successful.

- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

The `OI_LookupSpeedZ` function returns the actual drive speed of the Z axis control corresponding to a given cruise speed. The speed is derived from calibration value (i.e., microns per step size), the current cruise speed value, and is returned in mm per second.

Unlike the `OI_GetSpeedZ` function, which returns the speed corresponding to the currently selected cruise, the `OI_LookupSpeedZ` function returns the speed for a given cruise value.

**See Also**

- `OI_GetSpeedZ`
- `OI_SelectSpeedZ`
- `OI_GetCruiseZ`

---

**OI_MoveToZ**

**Syntax**

```
OI_API OI_MoveToZ(double dZ, int nWait)
```

**Description**

Moves to a given Z position.

**Parameters**

- `dZ`
  
  The desired Z position, in microns.

- `nWait`
  
  A flag indicating whether the function waits for the move to be completed before returning.

**Return Value**

- `OI_OK` if successful.

- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

If the `nWait` parameter is set to 0 for the call, the function returns immediately, i.e., it does not wait for the move to complete. You can use the `OI_WaitForStoppedXYZ` function also to delay execution until a move is complete.

**See Also**

- `OI_WaitForStoppedXYZ`
- `OI_StepZ`
OI_MoveToZ_Abs

Syntax

OI_API OI_MoveToZ_Abs(long lZ, int nWait)

Description
Moves to a given Z position, in microsteps.

Parameters

lZ The desired Z position, in microsteps.

nWait A flag indicating whether the function waits for the move to be completed before returning.

Return Value

OI_OK if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

Comments

The position is specified in microsteps from the axis origin, where Z = 0. If the nWait parameter is set to 0 for the call, the function returns immediately, i.e., it does not wait for the move to complete. You can use the OI_WaitForStoppedZ function also to delay execution until a move is complete.

See Also

OI_MoveToZ, OI_WaitForStoppedZ, OI_StepZ

OI_OpenMouseWheelForFocus

Syntax

OI_API OpenMouseWheelForFocus(HINSTANCE hinstModule, DWORD dwThreadId)

Description
Enables control of the Z focus by the mouse wheel on the primary pointing device.

Parameters

hinstModule The calling application’s module instance handle.

dwThreadId The ID of the calling application’s primary thread.

Return Value

OI_OK if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

Comments

The OASIS DLL can enable control of the Z focus using the mouse wheel, if fitted. When enabled, mouse wheel movements are translated into focus movements.

This may be used for convenient manual focusing operations without the need for the user to adjust the trackball, joystick, or focus drive on the microscope.
Note this function is not supported under Windows NT.

See Also OI_CloseMouseWheelForFocus

**OI_ReadLimitAlarmsZ**

**Syntax**

```c
OI_API OI_ReadLimitAlarmsZ(int* pnZNeg, int* pnZPos)
```

**Description**

Reads the current status of the Z axis limit alarms.

**Parameters**

- `pnZNeg`: Status of the negative limit for the Z axis.
- `pnZPos`: Status of the positive limit for the Z axis.

**Return Value**

- OI_OK if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

The `OI_ReadLimitAlarmsZ` function tells you whether the Z axis is currently at a user (software) or hardware limit.

The returned status values in the arguments can be the following:

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The axis is not at the limit</td>
</tr>
<tr>
<td>1</td>
<td>The axis is at a user limit</td>
</tr>
<tr>
<td>2</td>
<td>The axis is at a hardware limit</td>
</tr>
<tr>
<td>3</td>
<td>The axis is at both a user and a hardware limit</td>
</tr>
</tbody>
</table>

See Also OI_ReadStatusZ, OI_ReadLimitAlarmsXY, OI_ReadLimitAlarmsF

**OI_ReadRangeZ**

**Syntax**

```c
OI_API OI_ReadRangeZ(double* pZMin, double* pZMax)
```

**Description**

Reads the current range of Z travel.

**Parameters**

- `pZMin`: The lower limit for the Z axis range, in microns.
- `pZMax`: The upper limit for the Z axis range, in microns.
Return Value

OI_OK if successful.
If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

Comments

The Z axis range is set using OI_InitializeZ function.

See Also

OI_InitializeZ

---

**OI_ReadStatusZ**

Syntax

Description

Reads the current status of the Z axis axis.

Parameters

`lpwStatus`  
Returns the Z axis status value.

Return Value

OI_OK if successful.
If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

Comments

The returned status value can be a bit wise combination of the following values:

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_LIMIT_PHY_NEG</td>
<td>The axis is at the negative physical limit</td>
</tr>
<tr>
<td>S_LIMIT_USR_NEG</td>
<td>The axis is at the negative user limit</td>
</tr>
<tr>
<td>S_LIMIT_PHY_POS</td>
<td>The axis is at the positive physical limit</td>
</tr>
<tr>
<td>S_LIMIT_USR_POS</td>
<td>The axis is at the negative user limit</td>
</tr>
<tr>
<td>S_LIMIT_USR_NEG_SET</td>
<td>The user negative limit has been set</td>
</tr>
<tr>
<td>S_LIMIT_USR_POS_SET</td>
<td>The user positive limit has been set</td>
</tr>
<tr>
<td>S_INITIALIZED</td>
<td>The axis has been initialised</td>
</tr>
<tr>
<td>S_DIRECTION</td>
<td>If set, the direction of travel is negative</td>
</tr>
<tr>
<td>S_MOVING</td>
<td>The axis is moving</td>
</tr>
<tr>
<td>S_MOTOR_DETECTED</td>
<td>A motor was detected on the axis on startup.</td>
</tr>
</tbody>
</table>

See Also

OI_ReadStatusXY, OI_ReadStatusF, OI_ReadAxisStatus
**OI_ReadSyncZ**

**Syntax**

```c
OI_API OI_ReadSyncZ(LPWORD pwFieldNum, double *pdZ)
```

**Description**

Reads the Z-axis position synchronized with the most recent camera frame.

**Parameters**

- `pwFieldNum` - The camera field index counter value.
- `pdZ` - Returns the Z axis position value associated with the field index counter.

**Return Value**

- `OI_OK` if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

The OASIS-AF and OASIS-DC1 modules provide synchronization to video and digital cameras, respectively. When the camera synchronization has been enabled, by using a call to the `OI_SetCameraSyncMode` function, each camera frame detected by the module increments the camera frame index counter and causes the current position to be read and stored.

The `OI_ReadSyncZ` function returns the last synchronized Z-axis position, as well as the frame number associated with that value.

**See Also**

- `OI_SetCameraSyncMode`, `OI_SetDC1Registers`

**OI_ReadZ**

**Syntax**

```c
OI_API OI_ReadZ(double *pZ)
```

**Description**

Reads the current Z axis position.

**Parameters**

- `pZ` - The current Z axis position, in microns.

**Return Value**

- `OI_OK` if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

The Z position is given in microns, from the zero point set by a previous call to `OI_InitializeZ`.

**See Also**

- `OI_InitializeZ`, `OI_MoveToZ`
**OI_ReadZ_Abs**

**Syntax**

```c
OI_API OI_ReadZ_Abs(long* plZ)
```

**Description**

Reads the current Z axis position, in microsteps.

**Parameters**

- `plZ`: The current Z axis position, in microsteps.

**Return Value**

- `OI_OK` if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

The Z position is given in microsteps, from the zero point set by a previous call to `OI_InitializeZ`.

**See Also**

- `OI_InitializeZ`, `OI_MoveToZ`

---

**OI_RockZ**

**Syntax**

```c
OI_API OI_RockZ(BOOL bOn, double dZRange, int nSpeed)
```

**Description**

Enables / disables continuous movement of the focus over a range of travel.

**Parameters**

- `bOn`: TRUE starts the movement; FALSE will terminate the movement.
- `dZRange`: The desired range of travel, in microns, above and below the current Z position.
- `nSpeed`: The cruise speed of travel.

**Return Value**

- `OI_OK` if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

The `OI_RockZ` function continuously drives the focus over a given range about the current position. The range of travel may be limited due to the user limits for the Z axis.

**See Also**

- `OI_CloseMouseWheelForFocus`

---

**OI_SelectSpeedZ**

**Syntax**

```c
OI_API OI_SelectSpeedZ( double dMmPerSec, int nFlags )
```
Description
Automatically selects the cruise speed corresponding to a desired speed in mm per second.

Parameters
- **dMmPerSec**
The desired speeds for the Z axis, in mm per second.

- **nFlags**
Specifies how the search is performed, as described in the Comments below.

Return Value
- **OI_OK** if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

Comments
The **OI_SelectSpeedZ** function is used to automatically set the focus cruise speed to a specified actual speed target, in mm per second.

The **nFlags** parameter specifies how the search is to be carried out:

<table>
<thead>
<tr>
<th><strong>nFlags value</strong></th>
<th><strong>Meaning</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A cruise value is found that gives an actual speed as close to, but not exceeding, the desired speed.</td>
</tr>
<tr>
<td>1</td>
<td>A cruise value is found that gives the closest actual speed to the desired speed, including speeds that are greater than the desired speed.</td>
</tr>
</tbody>
</table>

The net effect of the **OI_SelectSpeedZ** function is equivalent to a call to **OI_SetCruiseZ** with parameters that give the best match to the desired actual speed.

Note that you may use the **OI_GetSpeedZ** and **OI_GetCruiseZ** functions to read the actual speed and cruise values that have been selected.

See Also
- **OI_GetSpeedZ**, **OI_GetCruiseZ**, **OI_SetCruiseZ**

---

**OI_SetCruiseZ**

Syntax
- **OI_API OI_SetCruiseZ(int nZCruise)**

Description
Specifies the Z cruise speed, defined as the maximum index used in the Z acceleration ramp table.

Parameters
- **nZCruise**
The Z axis cruise speed index.
Return Value
OI_OK if successful.
If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

Comments
The cruise speed is specified via the maximum index to be used in the currently defined acceleration / deceleration ramp for a given axis.

Each axis is assigned an associated ramp table in the OASIS hardware. This ramp table determines how acceleration and deceleration are accomplished, and also specifies the actual speeds to be used.

The ramp table has 512 entries, indexed from 0 to 511. The OI_SetCruiseZ function specifies which index in the table will be used as the maximum speed at which the Z axis is moved.

See Also OI_GetCruiseZ, OI_SetRampZ

---

**OI_SetDriveSenseZ**

**Syntax**

OI_API OI_SetDriveSenseZ(int nZDir)

**Description**

Specifies the physical direction of travel for positive and negative movements.

**Parameters**

- **nZDir**
  The Z axis drive sense.

**Return Value**

OI_OK if successful.
If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

The motor driving a given axis can be driven in either a clockwise or counterclockwise motion. The drive sense parameter sets which direction of rotation is associated with positive valued movements.

A value of zero (0) indicates standard movement.
A non-zero value indicates reversed movement.

See Also OI_GetDriveSenseZ, OI_SetAxisSense

---

**OI_SetMouseWheelPars**

**Syntax**

OI_API OI_SetMouseWheelPars(double dStepSize, int nSpeed)

**Description**

Sets the current settings for control of the Z axis using the mouse wheel.
Parameters

- **dStepSize**: The step size, in microns, for each step of the mouse wheel.
- **nSpeed**: The desired maximum speed of movement to continuous driving when the mouse wheel is pressed.

Return Value

- **OI_OK** if successful.

Comments

When mouse wheel control of the focus is enabled, the user may rotate the mouse wheel to step the focus up or down, or may hold down the mouse wheel and use the wheel rotation to increase the speed of a continuous drive in the positive or negative direction.

The **OI_SetMouseWheelPars** function defines the parameters used for stepping and driving the focus using the mouse wheel.

See Also **OI_GetMouseWheelPars**, **OI_OpenMouseWheelForFocus**, **OI_CloseMouseWheelForFocus**

---------

**OI_SetMouseWheelZ**

Syntax

```plaintext
OI_API OI_SetMouseWheelZ(BOOL bEnabled)
```

Description

Sets whether mouse wheel control of the Z axis is enabled.

Parameters

- **bEnabled**: Set to TRUE to enable, FALSE to disable.

Return Value

- **OI_OK** if successful.

Comments

Once the use of the mouse wheel for focus control has been established using the **OI_OpenMouseWheelForFocus** function, it may be enabled and disabled temporarily using the **OI_SetMouseWheelZ** function.

To terminate the hook function used to trap the mouse wheel events, use the **OI_CloseMouseWheelForFocus** function.

See Also **OI_GetMouseWheelPars**, **OI_OpenMouseWheelForFocus**, **OI_CloseMouseWheelForFocus**

---------

**OI_SetOriginZ**

Syntax

```plaintext
OI_API OI_SetOriginZ(void)
```

Description

Sets the current Z position to be the origin (e.g., 0).
Parameters: None.

Return Value:

- **OI_OK** if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

Comments:

The **OI_SetOriginZ** function is used to establish the origin of the overall, micron-based coordinate system of the Z focus.

The origin is defined to be position \([Z=0]\), and all positions are made relative to this origin.

By default, when the **OI_InitializeZ** function is used to initialise the range of travel available to the focus, the origin is set to the current position.

The **OI_SetOriginZ** function may be used to set the focus origin to another user-defined position. Note that the physical positions of the software limits are unchanged by this function.

**Warning**:

The **OI_SetOriginZ** function re-sets the entire coordinate system for the focus. After a call to this function, previously stored position values may no longer correspond to their associated physical focus positions.

**See Also**:

- **OI_InitializeZ**

---

### **OI_SetPitchZ**

**Syntax**

```c
OI_API OI_SetPitchZ(double dZPitch)
```

**Description**

Sets the current position for the Z axis.

**Parameters**

- `dZPitch` The pitch, in millimetres, of the Z axis.

**Return Value**

- **OI_OK** if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

The **OI_SetPitchZ** function is used to internally calculate the actual size of each micro-step for the Z axis. Typically, there are 12,800 micro-steps per revolution of the lead screw. From the supplied lead screw pitch, the **OI_SetPitchZ** function will automatically calculate this minimum step size for you.

**NOTE**: All micron to micro-step conversions use these values for their calibration. It is critical that these values be correctly supplied in order to ensure accurate movement.

To retrieve the current step size value, you may use the **OI_GetAxisStepSize** function.
Consult the specifications for your specific Z axis to determine the actual lead screw pitch. In many cases, the fine focus of a microscope gives 100 microns per revolution, or an effective pitch of 0.1 mm.

See Also  
OI_SetAxisStepSize, OI_GetAxisStepSize, OI_SetPitchXY, OI_SetPitchF

---

### OI_SetPositionZ

**Syntax**

```
OI_API OI_SetPositionZ(double dZ)
```

**Description**

Sets the current position for the Z axis.

**Parameters**

- `dZ`: The desired Z-axis position, in microns.

**Return Value**

- `OI_OK` if successful.

- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

The `OI_SetPositionZ` function resets the coordinate system for the Z axis. The current Z position is redefined to be the values passed in `dZ`. Note that the physical positions of the limits for the axis are retained by this function. That is, the `OI_SetPositionZ` function maintains the same relation between the current position and the position of the negative and positive soft limits. Therefore, the coordinates values associated for these limits will be changed if a new position value is specified for the axis.

See Also  
OI_SetPositionXYZ, OI_SetPositionXY, OI_SetPositionF, OI_SetOriginXY, OI_SetOriginZ, OI_SetOriginF, OI_InitializeXY, OI_InitializeZ, OI_InitializeF

---

### OI_SetRampZ

**Syntax**

```
OI_API OI_SetRampZ(int nZRamp)
```

**Description**

Specifies which pre-defined acceleration / deceleration ramp is in use for the Z axis.

**Parameters**

- `nZRamp`: Identifies the pre-defined ramp to be used.

**Return Value**

- `OI_OK` if successful.

- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

Three pre-defined tables may be selected for a given axis, as indicated by the `nRamp` parameter:
### OI_SetUserLimitsZ

**Syntax**

```c
OI_API OI_SetUserLimitsZ(double dZMin, double dZMax)
```

**Description**
Sets user-defined limits of travel along the Z axis.

**Parameters**
- `dZMin` - The minimum coordinate for the Z axis, in microns.
- `dZMax` - The maximum coordinate for the Z axis, in microns.

**Return Value**
- `OI_OK` if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**
- The `OI_SetUserLimitsZ` function allows “soft” limits to be set at any point along the Z axis. Once the soft limits are set, the OASIS controller will not allow any movement outside of these limit values.
- Note that the soft limits are distinct from the physical limit switches of the stage. The “hard” physical limit switches provide direct electronic feedback to the OASIS controller indicating the physical limits of travel available for the stage.
- When using the `OI_InitializeZ` function to initialise the range of travel and position of the stage, the OASIS controller automatically sets the current position to 0 and also sets the Z user limits to positions based on the ranges passed to the function.

**See Also**
- `OI_GetUserLimitsZ`, `OI_InitializeZ`

### OI_StepZ

**Syntax**

```c
OI_API OI_StepZ(double dZDistance, int nWait)
```

**Description**
Moves a relative distance from the current Z position.

**Parameters**
- `dZDistance` - The desired distance for the Z move, in microns.
nWait

A flag indicating whether the function waits for the move to be completed before returning.

Return Value

OI_OK if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

Comments

If the nWait parameter is set to 0 for the call, the function returns immediately, i.e., it does not wait for the move to complete. You can use the OI_WaitForStoppedXYZ function also to delay execution until a move is complete.

See Also

OI_WaitForStoppedXYZ, OI_MoveToZ

OI_WaitForStoppedZ

Syntax

OI_API OI_WaitForStoppedZ (void)

Description

Waits for the Z axis to stop moving.

Parameters

None.

Return Value

OI_OK if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

Comments

The OI_WaitForStoppedZ function is useful after any Z axis move functions are called with zero wait parameters. OI_WaitForStoppedZ will not return until the F axis has completed its movement.

See Also

OI_WaitForStoppedXYZ, OI_MoveToZ

F-Axis (4th axis) Control

OI_ClearUserLimitsF

Syntax

OI_API OI_ClearUserLimitsF(void)

Description

Clears the user limits for the F axis.
Parameters
None.

Return Value
OI_OK if successful.
If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

Comments
The positive and negative software limits for the F axis will be cleared by this function. Only a physical limit will restrict the range of travel.

See Also
OI_SetUserLimitsF, OI_InitializeF

OI_GetCruiseF

Syntax
OI_API OI_GetCruiseF(int* pnFCruise)

Description
Retrieves the current F axis cruise speed index.

Parameters
pnFCruise
The returned F axis cruise speed index.

Return Value
OI_OK if successful.
If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

Comments
See the Comments for the OI_SetCruiseF function for more information about cruise speeds.

See Also
OI_SetCruiseF

OI_GetDriveSenseF

Syntax
OI_API OI_GetDriveSenseF(int* pnFDir)

Description
Retrieves the current direction of rotation setting for the F axis.

Parameters
pnFDir
The returned F axis drive sense.

Return Value
OI_OK if successful.
If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

Comments
See the Comments for OI_SetDriveSenseF for more information about the values for drive sense.

See Also
OI_SetDriveSenseF, OI_GetAxisSense
**OI_GetRampF**

**Syntax**

```c
OI_API OI_GetRampF(int* pnFRamp)
```

**Description**

Retrieves which pre-defined acceleration / deceleration ramp is in use for the F axis.

**Parameters**

- `pnFRamp` Indicates which pre-defined ramp is currently in use for the F axis.

**Return Value**

- `OI_OK` if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

Three pre-defined tables may be selected for a given axis, as indicated by the `nRamp` parameter:

<table>
<thead>
<tr>
<th>nRamp value</th>
<th>Acceleration</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Slow</td>
</tr>
<tr>
<td>1</td>
<td>Medium</td>
</tr>
<tr>
<td>2</td>
<td>Fast</td>
</tr>
</tbody>
</table>

See Also: OI_SetRampF, OI_SetAxisRamp

---

**OI_GetSpeedF**

**Syntax**

```c
OI_API OI_GetSpeedF( double* pdSpeed )
```

**Description**

Retrieves the current speeds, in mm per second, in use by the F axis.

**Parameters**

- `pdSpeed` The returned current F-axis speed, in mm/s.

**Return Value**

- `OI_OK` if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

The `OI_GetSpeedF` returns the actual drive speed of the 4th axis control corresponding to the cruise speed values. The speed is derived from calibration value (i.e., microns per step size), the current cruise speed value, and is returned in mm per second.

See Also: OI_SelectSpeedF, OI_LookupSpeedF
**OI_GetUserLimitsF**

**Syntax**

```
OI_API OI_GetUserLimitsF(double* pdFMin, double* pdFMax)
```

**Description**

Retrieves the current user limit settings for the F axis.

**Parameters**

- `pdFMin`: The minimum coordinate for the F axis, in microns.
- `pdFMax`: The maximum coordinate for the F axis, in microns.

**Return Value**

- `OI_OK` if successful.

- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

See the Comments for the `OI_SetUserLimitsF` for more information about user limits.

**See Also**

- `OI_SetUserLimitsF`
- `OI.InitializeF`

---

**OI_HaltF**

**Syntax**

```
OI_API OI_HaltF(void)
```

**Description**

Stops any motion of the F axis.

**Parameters**

None.

**Return Value**

- `OI_OK` if successful.

- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

The `OI_HaltF` functions uses the F axis’s deceleration ramp to stop the stage smoothly.

To immediately stop the stage without using the deceleration ramp, use the `OI_EmergencyStopAll` function.

**See Also**

- `OI_EmergencyStopAll`

---

**OI_InitializeF**

**Syntax**

```
OI_API OI_InitializeF(void)
```

**Description**
Initialises the F axis by searching for negative and positive physical limit switches.

**Parameters**
None.

**Return Value**
OI_OK if successful.

OI_ABORT if the user aborts the initialisation process using either the ESC or CTRL-C key press.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**
The current position is set to zero, and the F user limits are set to the given ranges above and below the current position.

**See Also**
OI_InitializeFRange, OI_ReadRangeF

---

**OI_InitializeFRange**

**Syntax**
OI_API OI_InitializeFRange(double dFNegLimit, double dFPosLimit)

**Description**
Initialises the F axis using a specified distance on each side of the current position.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dFNegLimit</td>
<td>The desired allowable distance in the negative direction from the current position, in microns.</td>
</tr>
<tr>
<td>dFPosLimit</td>
<td>The desired allowable distance in the positive direction from the current position, in microns.</td>
</tr>
</tbody>
</table>

**Return Value**
OI_OK if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**
In situation where the F axis is fitted to a device that does not provide physical limit switches, use the OI_InitializeFRange function to initialise the axis to a desired range of travel about the current position.

The current position is set to zero, and the F user limits are set to the given ranges above and below the current position.

**See Also**
OI_InitializeF, OI_ReadRangeF
**OI_LookupSpeedF**

**Syntax**

OI_API OI_LookupSpeedF( int nCruise, double* pdSpeed )

**Description**

Retrieves the speeds, in mm per second, corresponding to a given cruise value for the F axis.

**Parameters**

- **nCruise**
  The cruise speed for which the actual speed is desired.

- **pdSpeed**
  The returned current F-axis speed, in mm/s, for the given cruise speed.

**Return Value**

OI_OK if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

The OI_LookupSpeedF returns the actual drive speed of the 4th axis control corresponding to a given cruise speed. The speed is derived from calibration value (i.e., microns per step size), the current cruise speed value, and is returned in mm per second.

Unlike the OI_GetSpeedF function, which returns the speed corresponding to the currently selected cruise, the OI_LookupSpeedF function returns the speed for a given cruise value.

**See Also**

OI_GetSpeedF, OI_SelectSpeedF, OI_GetCruiseF

---

**OI_MoveToF**

**Syntax**

OI_API OI_MoveToF(double dF, int nWait)

**Description**

Moves to a given F position.

**Parameters**

- **dF**
  The desired F position, in microns.

- **nWait**
  A flag indicating whether the function waits for the move to be completed before returning.

**Return Value**

OI_OK if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

If the nWait parameter is set to 0 for the call, the function returns immediately, i.e., it does not wait for the move to complete. You can use the OI_WaitForStoppedXYZ function also to delay execution until a move is complete.
See Also  
OL_WaitForStoppedXYZ, OL_StepZ

**OI_ReadF**

**Syntax**  
OI_API OI_ReadF(double *pF)

**Description**  
Reads the current F axis position.

**Parameters**  
pF  
The current F axis position, in microns.

**Return Value**  
OI_OK if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**  
The F position is given in microns, from the zero point set by a previous call to OI_InitializeF.

See Also  
OI_InitializeF, OL_MoveToF

**OI_ReadLimitAlarmsF**

**Syntax**  
OI_API OI_ReadLimitAlarmsF(int* pnFNeg, int* pnFPos)

**Description**  
Reads the current status of the F axis limit alarms.

**Parameters**  
pnFNeg  
Status of the negative limit for the F axis.

pnFPos  
Status of the positive limit for the F axis.

**Return Value**  
OI_OK if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**  
The OI_ReadLimitAlarmsF functions tells you whether the F axis is currently at a user (software) or hardware limit.

The returned status values in the arguments can be the following:

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The axis is not at the limit</td>
</tr>
<tr>
<td>1</td>
<td>The axis is at a user limit</td>
</tr>
<tr>
<td>2</td>
<td>The axis is at a hardware limit</td>
</tr>
</tbody>
</table>
The axis is at both a user and a hardware limit

See Also OI_ReadStatusF, OI_ReadLimitAlarmsXY, OI_ReadLimitAlarmsZ

---

**OI_ReadRangeF**

**Syntax**

```
OI_API OI_ReadRangeF(double* pFMin, double* pFMax)
```

**Description**

Reads the current range of F travel.

**Parameters**

- `pFMin`:
  - The lower limit for the F axis range, in microns.
- `pFMax`:
  - The upper limit for the F axis range, in microns.

**Return Value**

- `OI_OK` if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

The F axis range is set using `OI_InitializeF` function.

**See Also**

- OI_InitializeF

---

**OI_ReadStatusF**

**Syntax**

```
OI_API OI_ReadStatusF(LPWORD lpwStatus)
```

**Description**

Reads the current status of the F axis.

**Parameters**

- `lpwStatus`:
  - Returns the F axis status value.

**Return Value**

- `OI_OK` if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

The returned status value can be a bit wise combination of the following values:

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_LIMIT_PHY_NEG</td>
<td>The axis is at the negative physical limit</td>
</tr>
<tr>
<td>S_LIMIT_USR_NEG</td>
<td>The axis is at the negative user limit</td>
</tr>
<tr>
<td>S_LIMIT_PHY_POS</td>
<td>The axis is at the positive physical limit</td>
</tr>
</tbody>
</table>
**S_LIMIT_USR_POS**  The axis is at the negative user limit

**S_LIMIT_USR_NEG_SET**  The user negative limit has been set

**S_LIMIT_USR_POS_SET**  The user positive limit has been set

**S_INITIALIZED**  The axis has been initialised

**S_DIRECTION**  If set, the direction of travel is negative

**S_MOVING**  The axis is moving

**See Also**  OI_ReadStatusXY, OI_ReadStatusZ, OI_ReadAxisStatus

---

**OI_SelectSpeedF**

**Syntax**  

```c
OI_API OI_SelectSpeedF( double dMmPerSec, int nFlags )
```

**Description**  
Automatically selects the cruise speed corresponding to a desired speed in mm per second.

**Parameters**

- **dMmPerSec**  The desired speeds for the F axis, in mm per second.
- **nFlags**  Specifies how the search is performed, as described in the Comments below.

**Return Value**

- **OI_OK** if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**  
The `OI_SelectSpeedF` function is used to automatically set the 4th axis cruise speed to a specified actual speed target, in mm per second.

The `nFlags` parameter specifies how the search is to be carried out:

<table>
<thead>
<tr>
<th>nFlags value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A cruise value is found that gives an actual speed as close to, but not exceeding, the desired speed.</td>
</tr>
<tr>
<td>1</td>
<td>A cruise value is found that gives the closest actual speed to the desired speed, including speeds that are greater than the desired speed.</td>
</tr>
</tbody>
</table>
The net effect of the `OI_SelectSpeedF` function is equivalent to a call to `OI_SetCruiseF` with parameters that give the best match to the desired actual speed.

Note that you may use the `OI_GetSpeedF` and `OI_GetCruiseF` functions to read the actual speed and cruise values that have been selected.

**See Also**  
`OI_GetSpeedF`, `OI_GetCruiseF`, `OI_SetCruiseF`

---

**OI_SetCruiseF**

**Syntax**  
`OI_API OI_SetCruiseF(int nFCruise)`

**Description**  
Specifies the F cruise speed, defined as the maximum index used in the F acceleration ramp table.

**Parameters**  
- `nFCruise`  
The F axis cruise speed index.

**Return Value**  
- `OI_OK` if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**  
The cruise speed is specified via the maximum index to be used in the currently defined acceleration / deceleration ramp for a given axis.

Each axis is assigned an associated ramp table in the OASIS hardware. This ramp table determines how acceleration and deceleration are accomplished, and also specifies the actual speeds to be used.

The ramp table has 512 entries, indexed from 0 to 511. The `OI_SetCruiseF` function specifies which index in the table will be used as the maximum speed at which the F axis is moved.

**See Also**  
`OI_GetCruiseF`, `OI_SetRampF`

---

**OI_SetDriveSenseF**

**Syntax**  
`OI_API OI_SetDriveSenseF(int nFDir)`

**Description**  
Specifies the physical direction of travel for positive and negative movements.

**Parameters**  
- `nFDir`  
The F axis drive sense.

**Return Value**  
- `OI_OK` if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.
reason for failure.

Comments  The motor driving a given axis can be driven in either a clockwise or counter-clockwise motion. The drive sense parameter sets which direction of rotation is associated with positive valued movements.

A value of zero (0) indicates standard movement.

A non-zero value indicates reversed movement.

See Also  OI_GetDriveSenseF, OI_SetAxisSense

---

**OI_SetOriginF**

**Syntax**

```c
OI_API OI_SetOriginF(void)
```

**Description**

Sets the current F position to be the origin (e.g., 0).

**Parameters**

None.

**Return Value**

OI_OK if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

The `OI_SetOriginF` function is used to establish the origin of the overall, micron-based coordinate system of the F axis.

The origin is defined to be position \[F=0\], and all positions are made relative to this origin.

By default, when the `OI_InitializeFRange` function is used to initialise the range of travel available to the F axis, the origin is set to the current position

The `OI_SetOriginF` function may be used to set the F axis origin to another user-defined position. Note that the physical positions of the software limits are unchanged by this function.

**Warning**

The `OI_SetOriginF` function re-sets the entire coordinate system for the F axis. After a call to this function, previously stored position values may no longer correspond to their associated physical focus positions.

**See Also**  OI_InitializeFRange

---

**OI_SetPitchF**

**Syntax**

```c
OI_API OI_SetPitchF(double dFPitch)
```

---

See Also  OI_GetDriveSenseF, OI_SetAxisSense
**Description**

Sets the current position for the F axis.

**Parameters**

- `dFPitch`  
The pitch, in millimetres, of the F axis.

**Return Value**

- `OI_OK` if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

The `OI_SetPitchF` function is used to internally calculate the actual size of each micro-step for the F axis. Typically, there are 12,800 micro-steps per revolution of the lead screw. From the supplied lead screw pitch, the `OI_SetPitchF` function will automatically calculate this minimum step size for you.

NOTE: All micron to micro-step conversions use these values for their calibration. It is critical that these values be correctly supplied in order to ensure accurate movement.

To retrieve the current step size value, you may use the `OI_GetAxisStepSize` function.

Consult the specifications for your specific F axis to determine the actual lead screw pitch.

**See Also**

- `OI_SetAxisStepSize`, `OI_GetAxisStepSize`, `OI_SetPitchXY`, `OI_SetPitchZ`

---

**OI_SetPositionF**

**Syntax**

```c
OI_API OI_SetPositionF(double dF)
```

**Description**

Sets the current position for the F axis.

**Parameters**

- `dF`  
The desired F-axis position, in microns.

**Return Value**

- `OI_OK` if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

The `OI_SetPositionF` function resets the coordinate system for the F axis. The current F position is redefined to be the values passed in `dF`.

Note that the physical positions of the limits for the axis are retained by this function. That is, the `OI_SetPositionF` function maintains the same relation between the current position and the position of the negative and positive soft limits. Therefore, the coordinates values associated for these limits will be changed if a new position value is specified for the axis.

**See Also**

- `OI_SetPositionXYZ`, `OI_SetPositionXY`, `OI_SetPositionZ`, `OI_SetOriginXY`, `OI_SetOriginZ`, `OI_SetOriginF`, `OI_InitializeXY`, `OI_InitializeZ`, `OI_InitializeF`
**OI_SetRampF**

**Syntax**

```
OI_API OI_SetRampF(int nFRamp)
```

**Description**

Specifies which pre-defined acceleration / deceleration ramp is in use for the F axis.

**Parameters**

- `nFRamp`
  Identifies the pre-defined ramp to be used.

**Return Value**

- `OI_OK` if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

Three pre-defined tables may be selected for a given axis, as indicated by the `nRamp` parameter:

<table>
<thead>
<tr>
<th><code>nRamp</code> value</th>
<th>Acceleration</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Slow</td>
</tr>
<tr>
<td>1</td>
<td>Medium</td>
</tr>
<tr>
<td>2</td>
<td>Fast</td>
</tr>
</tbody>
</table>

**See Also**

- `OI_GetRampF`, `OI_SetRampXY`, `OI_SetRampF`, `OI_SetAxisRamp`

---

**OI_SetUserLimitsF**

**Syntax**

```
OI_API OI_SetUserLimitsF(double dFMin, double dFMax)
```

**Description**

Sets user-defined limits of travel along the F axis.

**Parameters**

- `dFMin`
  The minimum coordinate for the F axis, in microns.
- `dFMax`
  The maximum coordinate for the F axis, in microns.

**Return Value**

- `OI_OK` if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

The `OI_SetUserLimitsF` functions allows “soft” limits to be set at any point along the F axis. Once the soft limits are set, the OASIS controller will not allow any movement outside of these limit values.

Note that the soft limits are distinct from the physical limit switches of the stage. The “hard” physical limit switch switches provide direct electronic feedback to the
OASIS controller indicating the physical limits of travel available for the stage.

When using the `OI_InitializeFRange` function to initialise the range of travel and position of the stage, the OASIS controller automatically sets the current position to 0 and also sets the Z user limits to positions based on the ranges passed to the function.

See Also  `OI_GetUserLimitsZ, OI_InitializeFRange`

---

### OI_StepF

**Syntax**  
`OIAPI OI_StepF(double dFDistance, int nWait)`

**Description**  
Moves a relative distance from the current F position.

**Parameters**  
- `dFDistance`  
  The desired distance for the F move, in microns.
- `nWait`  
  A flag indicating whether the function waits for the move to be completed before returning.

**Return Value**  
`OI_OK` if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**  
If the `nWait` parameter is set to 0 for the call, the function returns immediately, i.e., it does not wait for the move to complete. You can use the `OI_WaitForStoppedXYZ` function also to delay execution until a move is complete.

See Also  `OI_WaitForStoppedXYZ, OI_MoveToF`

---

### OI_WaitForStoppedF

**Syntax**  
`OIAPI OI_WaitForStoppedF (void)`

**Description**  
Waits for the F axis to stop moving.

**Parameters**  
None.

**Return Value**  
`OI_OK` if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**  
The `OI_WaitForStoppedF` function is useful after any F axis move functions
are called with zero wait parameters. OI_WaitForStoppedF will not return until the F axis has completed its movement.

See Also OI_WaitForStoppedXYZ, OI_MoveToF

---

**T-Axis (5th axis) and S-Axis (6th axis)**

**Control**

When using the OASIS-blue controller, the BLUE-DAC plug-in daugther module plus the BLUE-CONNECT output sister card provides an additional 2 axes of stepper control. When using the OASIS-4i controller, the OASIS-XA1 module is a plug-in daughter board for the OASIS controller that provides a single additional axis of movement.

This 5th axis is designated “T”, and the 6th axis is designated “S”. The OASIS DLL includes a number of functions to manage the setup and control of the T- and S- axes via the optional hardware modules.

The following API’s are listed for the T-axis but apply equally for the S-axis using the variants shown under Syntax.

---

**OI_ClearUserLimitsT**

| Syntax | OI_API OI_ClearUserLimitsT(void)  
| OI_API OI_ClearUserLimitsS(void) |

**Description**
Clears the user limits for the axis.

**Parameters**
None.

**Return Value**
OI_OK if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**
The positive and negative software limits for the T axis will be cleared by this function. Only a physical limit will restrict the range of travel.

**See Also**
OI_SetUserLimitsT, OI_InitializeT
**OI_DriveContinuousT**

**Syntax**

```
OI_API OI_DriveContinuousT(int nSpeed)

OI_API OI_DriveContinuousS(int nSpeed)
```

**Description**

Drives the axis at a continuous speed.

**Parameters**

- **nSpeed**
  A signed integer indicating the direction and speed at which to drive the T axis.

**Return Value**

- **OI_OK** if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

The `nSpeed` parameter specifies the desired speed of movement in half-steps per second.

The speed values are signed to indicate the direction of travel, i.e., a negative speed causes a continuous drive in the negative direction, and may be any integer in the range of –4096 to +4096.

To stop the continuous movement, use a corresponding call to either the `OI_HaltT` or the `OI_EmergencyStopAll` function.

**See Also**

- OI_HaltT, OI_EmergencyStopAll

---

**OI_GetCruiseT**

**Syntax**

```
OI_API OI_GetCruiseT(int* pnCruise)

OI_API OI_GetCruiseS(int* pnCruise)
```

**Description**

Retrieves the current T axis cruise speed index.

**Parameters**

- **pnCruise**
  The returned axis cruise speed index.

**Return Value**

- **OI_OK** if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

See the Comments for the `OI_SetCruiseT` function for more information about cruise speeds.

**See Also**

- OI_SetCruiseT
### OI_GetDriveSenseT

**Syntax**

```c
OI_API OI_GetDriveSenseT(int* pnDir)
OI_API OI_GetDriveSenseS(int* pnDir)
```

**Description**

Retrieves the current direction of rotation setting for the axis.

**Parameters**

- `pnDir` The returned axis drive sense.

**Return Value**

- `OI_OK` if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

See the Comments for `OI_SetDriveSenseT` for more information about the values for drive sense.

**See Also**

`OI_SetDriveSenseT`, `OI_GetAxisSense`

### OI_GetRampT

**Syntax**

```c
OI_API OI_GetRampT(int* pnRamp)
OI_API OI_GetRampS(int* pnRamp)
```

**Description**

Retrieves which pre-defined acceleration / deceleration ramp is in use for the axis.

**Parameters**

- `pnRamp` Indicates which pre-defined ramp is currently in use for the T axis.

**Return Value**

- `OI_OK` if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

Three pre-defined tables may be selected for a given axis, as indicated by the `pnRamp` parameter:

<table>
<thead>
<tr>
<th><code>pnRamp</code> value</th>
<th>Acceleration</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Slow</td>
</tr>
<tr>
<td>1</td>
<td>Medium</td>
</tr>
<tr>
<td>2</td>
<td>Fast</td>
</tr>
</tbody>
</table>

**See Also**

`OI_SetRampT`, `OI_SetAxisRamp`
### OI_GetSpeedT

**Syntax**

OI_API OI_GetSpeedT( double* pdSpeed )  
OI_API OI_GetSpeedS( double* pdSpeed )

**Description**

Retrieves the current speeds, in mm per second, in use by the axis.

**Parameters**

- **pdSpeed**
  - The returned current axis speed, in mm/s.

**Return Value**

- OI_OK if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

The `OI_GetSpeedT` returns the actual drive speed of the 5th axis control corresponding to the cruise speed values. The speed is derived from calibration value (i.e., microns per step size), the current cruise speed value, and is returned in mm per second.

**See Also**

- OI_SelectSpeedT, OI_LookupSpeedT

### OI_GetUserLimitsT

**Syntax**

OI_API OI_GetUserLimitsT(double* pdMin, double* pdMax)  
OI_API OI_GetUserLimitsS(double* pdMin, double* pdMax)

**Description**

Retrieves the current user limit settings for the S axis.

**Parameters**

- **pdMin**
  - The minimum coordinate for the axis, in microns.
- **pdMax**
  - The maximum coordinate for the axis, in microns.

**Return Value**

- OI_OK if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

See the Comments for the `OI_SetUserLimitsT` for more information about user limits.

**See Also**

- OI_SetUserLimitsT, OI_InitializeT

### OI_HaltT

**Syntax**

OI_API OI_HaltT(void)

OI_API OI_HaltS(void)

Description  Stops any motion of the axis.

Parameters  None.

Return Value  OI_OK if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

Comments  The OI_HaltT function uses the T axis’s deceleration ramp to stop the stage smoothly.

To immediately stop the stage without using the deceleration ramp, use the OI_EmergencyStopAll function.

See Also  OI_EmergencyStopAll

---

**OI_HaltT**

| Syntax | OI_API OI_HaltT(void)  
|        | OI_API OI_HaltS(void) |

| Description | Initialises the axis by searching for negative and positive physical limit switches. |

| Parameters | None. |

| Return Value | OI_OK if successful. |

|            | OI_ABORT if the user aborts the initialisation process using either the ESC or CTRL-C key press. |

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

| Comments | The current position is set to zero, and the T user limits are set to the given ranges above and below the current position. |

| See Also | OI_HaltTRange, OI_ReadRangeT |

---

**OI_HaltTRange**

| Syntax | OI_API OI_HaltTRange(double dNegLimit, double dPosLimit)  
|        | OI_API OI_HaltSRange(double dNegLimit, double dPosLimit) |


**Description**
Initialises the axis using a specified distance on each side of the current position.

**Parameters**
- \textit{dNegLimit} The desired allowable distance in the negative direction from the current position, in microns.
- \textit{dPosLimit} The desired allowable distance in the positive direction from the current position, in microns.

**Return Value**
- \texttt{OI\_OK} if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**
In situation where the T axis is fitted to a device that does not provide physical limit switches, use the \texttt{OI\_InitializeTRange} function to initialise the axis to a desired range of travel about the current position.

The current position is set to zero, and the T user limits are set to the given ranges above and below the current position.

**See Also** \texttt{OI\_InitializeT, OI\_ReadRangeT}

---

**\texttt{OI\_LookupSpeedT}\texttt{**}

**Syntax**

\texttt{OI\_API OI\_LookupSpeedT( int nCruise, double* pdSpeed )}

\texttt{OI\_API OI\_LookupSpeedS( int nCruise, double* pdSpeed )}

**Description**
Retrieves the speeds, in mm per second, corresponding to a given cruise value for the axis.

**Parameters**
- \textit{nCruise} The cruise speed for which the actual speed is desired.
- \textit{pdSpeed} The returned current axis speed, in mm/s, for the given cruise speed.

**Return Value**
- \texttt{OI\_OK} if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**
The \texttt{OI\_LookupSpeedT} returns the actual drive speed of the 5th axis control corresponding to a given cruise speed. The speed is derived from calibration value (i.e., microns per step size), the current cruise speed value, and is returned in mm per second.

Unlike the \texttt{OI\_GetSpeedT} function, which returns the speed corresponding to the currently selected cruise, the \texttt{OI\_LookupSpeedT} function returns the speed for a given cruise value.
See Also  OI_GetSpeedT, OI_SelectSpeedT, OI_GetCruiseT

### OI_MoveToT

**Syntax**

```c
OI_API OI_MoveToT(double dPos, int nWait)
OI_API OI_MoveToS(double dPos, int nWait)
```

**Description**

Moves to a given position.

**Parameters**

- `dPos`  
  The desired position, in microns.
- `nWait`  
  A flag indicating whether the function waits for the move to be completed before returning.

**Return Value**

- `OI_OK` if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

If the `nWait` parameter is set to 0 for the call, the function returns immediately, i.e., it does not wait for the move to complete. You can use the `OI_WaitForStoppedT` function also to delay execution until a move is complete.

**See Also**  OI_WaitForStoppedT, OI_StepT

### OI_ReadLimitAlarmsT

**Syntax**

```c
OI_API OI_ReadLimitAlarmsT(int* pnNeg, int* pnPos)
OI_API OI_ReadLimitAlarmsS(int* pnNeg, int* pnPos)
```

**Description**

Reads the current status of the axis limit alarms.

**Parameters**

- `pnNeg`  
  Status of the negative limit for the axis.
- `pnPos`  
  Status of the positive limit for the axis.

**Return Value**

- `OI_OK` if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

The `OI_ReadLimitAlarmsT` function tells you whether the T axis is currently at a user (software) or hardware limit.

The returned status values in the arguments can be the following:
### Status Code

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The axis is not at the limit</td>
</tr>
<tr>
<td>1</td>
<td>The axis is at a user limit</td>
</tr>
<tr>
<td>2</td>
<td>The axis is at a hardware limit</td>
</tr>
<tr>
<td>3</td>
<td>The axis is at both a user and a hardware limit</td>
</tr>
</tbody>
</table>

**See Also**  
`OI_ReadStatusT`, `OI_ReadLimitAlarmsXY`, `OI_ReadLimitAlarmsZ`, `OI_ReadLimitAlarmsF`  

---

### OI_ReadRangeT

**Syntax**

```c
OI_API OI_ReadRangeT(double* pdMin, double* pdMax)
Oi_API OI_ReadRangeS(double* pdMin, double* pdMax)
```

**Description**  
Reads the current range of travel.

**Parameters**

- `pdMin`  
  The lower limit for the axis range, in microns.
- `pdMax`  
  The upper limit for the axis range, in microns.

**Return Value**

- OI_OK if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**  
The T axis range is set using `OI_InitializeT` function.

**See Also**  
`OI_InitializeT`

---

### OI_ReadStatusT

**Syntax**

```c
Oi_API OI_ReadStatusT(LPWORD lpwStatus)
Oi_API OI_ReadStatusS(LPWORD lpwStatus)
```

**Description**  
Reads the current status of the axis.

**Parameters**

- `lpwStatus`  
  Returns the axis status value.

**Return Value**

- OI_OK if successful.
If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

Comments

The returned status value can be a bit wise combination of the following values:

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_LIMIT_PHY_NEG</td>
<td>The axis is at the negative physical limit</td>
</tr>
<tr>
<td>S_LIMIT_USR_NEG</td>
<td>The axis is at the negative user limit</td>
</tr>
<tr>
<td>S_LIMIT_PHY_POS</td>
<td>The axis is at the positive physical limit</td>
</tr>
<tr>
<td>S_LIMIT_USR_POS</td>
<td>The axis is at the negative user limit</td>
</tr>
<tr>
<td>S_LIMIT_USR_NEG_SET</td>
<td>The user negative limit has been set</td>
</tr>
<tr>
<td>S_LIMIT_USR_POS_SET</td>
<td>The user positive limit has been set</td>
</tr>
<tr>
<td>S_INITIALIZED</td>
<td>The axis has been initialised</td>
</tr>
<tr>
<td>S_DIRECTION</td>
<td>If set, the direction of travel is negative</td>
</tr>
<tr>
<td>S_MOVING</td>
<td>The axis is moving</td>
</tr>
</tbody>
</table>

See Also OI_ReadStatusXY, OI_ReadStatusZ, OI_ReadStatusF, OI_ReadAxisStatus

---

**OI_ReadT**

Syntax

\[
\text{OI\_API } \text{OI\_ReadT}(\text{double } ^*\text{pPos})
\]

\[
\text{OI\_API } \text{OI\_ReadS}(\text{double } ^*\text{pPos})
\]

Description

Reads the current axis position.

Parameters

- \( ^*\text{pPos} \)
  - The current axis position, in microns.

Return Value

- OI_OK if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

Comments

The T position is given in microns, from the zero point set by a previous call to OI.InitializeT.

See Also OI.InitializeT, OI_MoveToT
**OI_SelectSpeedT**

**Syntax**

```c
OI_API OI_SelectSpeedT( double dMmPerSec, int nFlags )
OI_API OI_SelectSpeedS( double dMmPerSec, int nFlags )
```

**Description**

Automatically selects the cruise speed corresponding to a desired speed in mm per second.

**Parameters**

- `dMmPerSec` The desired speeds for the axis, in mm per second.
- `nFlags` Specifies how the search is performed, as described in the Comments below.

**Return Value**

- OI_OK if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

The `OI_SelectSpeedT` function is used to automatically set the 5th axis cruise speed to a specified actual speed target, in mm per second.

The `nFlags` parameter specifies how the search is to be carried out:

<table>
<thead>
<tr>
<th>nFlags value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A cruise value is found that gives an actual speed as close to, but not exceeding, the desired speed.</td>
</tr>
<tr>
<td>1</td>
<td>A cruise value is found that gives the closest actual speed to the desired speed, including speeds that are greater than the desired speed.</td>
</tr>
</tbody>
</table>

The net effect of the `OI_SelectSpeedT` function is equivalent to a call to `OI_SetCruiseT` with parameters that give the best match to the desired actual speed.

Note that you may use the `OI_GetSpeedT` and `OI_GetCruiseT` functions to read the actual speed and cruise values that have been selected.

**See Also**

- `OI_GetSpeedT`, `OI_GetCruiseT`, `OI_SetCruiseT`

---

**OI_SetCruiseT**

**Syntax**

```c
OI_API OI_SetCruiseT(int nCruise)
```
OI_API OI_SetCruiseS(int nCruise)

Description
Specifies the cruise speed index. For the OASIS-4i, this is defined as the maximum index used in the F acceleration ramp table. For the OASIS-blue, the T and S cruise are independently set.

Parameters
- nCruise: The axis cruise speed index.

Return Value
- OI_OK if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

Comments
The cruise speed is specified via the maximum index to be used in the currently defined acceleration / deceleration ramp for a given axis.

Each axis is assigned an associated ramp table in the OASIS hardware. This ramp table determines how acceleration and deceleration are accomplished, and also specifies the actual speeds to be used.

The ramp table has 512 entries, indexed from 0 to 511. The OI_SetCruiseT function specifies which index in the table will be used as the maximum speed at which the T axis is moved.

See Also OI_GetCruiseT, OI_SetRampT

---

OI_SetDriveSenseT

Syntax
- OI_API OI_SetDriveSenseT(int nDir)
- OI_API OI_SetDriveSenseS(int nDir)

Description
Specifies the physical direction of travel for positive and negative movements.

Parameters
- nDir: The axis drive sense.

Return Value
- OI_OK if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

Comments
The motor driving a given axis can be driven in either a clockwise or counter-clockwise motion. The drive sense parameter sets which direction of rotation is associated with positive valued movements.

A value of zero (0) indicates standard movement.

A non-zero value indicates reversed movement.

See Also OI_GetDriveSenseT, OI_SetAxisSense
**OI_SetOriginT**

**Syntax**

```
OI_API OI_SetOriginT(void)
OI_API OI_SetOriginS(void)
```

**Description**

Sets the current position to be the origin (i.e., 0.0). The current location will be defined as position 0.0.

**Parameters**

None.

**Return Value**

OI_OK if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

The `OI_SetOriginF` function is used to establish the origin of the overall, micron-based coordinate system of the F axis.

The origin is defined to be position [F=0], and all positions are made relative to this origin.

By default, when the `OI_InitializeFRange` function is used to initialise the range of travel available to the F axis, the origin is set to the current position.

The `OI_SetOriginF` function may be used to set the F axis origin to another user-defined position. Note that the physical positions of the software limits are unchanged by this function.

**Warning**

The `OI_SetOriginF` function re-sets the entire coordinate system for the F axis. After a call to this function, previously stored position values may no longer correspond to their associated physical focus positions.

**See Also**

`OI_InitializeFRange`

---

**OI_SetPitchT**

**Syntax**

```
OI_API OI_SetPitchT(double dPitch)
OI_API OI_SetPitchS(double dPitch)
```

**Description**

Sets the current position for the axis.

**Parameters**

- `dPitch`  
  The pitch, in millimetres, of the axis.

**Return Value**

OI_OK if successful.

If unsuccessful, a combination of error codes may be returned to indicate the
reason for failure.

**Comments**
The `OI_SetPitchT` function is used to internally calculate the actual size of each micro-step for the T axis. Typically, there are 12,800 micro-steps per revolution of the lead screw. From the supplied lead screw pitch, the `OI_SetPitchT` function will automatically calculate this minimum step size for you.

NOTE: All micron to micro-step conversions use these values for their calibration. It is critical that these values be correctly supplied in order to ensure accurate movement.

To retrieve the current step size value, you may use the `OI_GetAxisStepSize` function.

Consult the specifications for your specific F axis to determine the actual lead screw pitch.

**See Also**
`OI_SetAxisStepSize`, `OI_GetAxisStepSize`, `OI_SetPitchXY`, `OI_SetPitchZ`, `OI_SetPitchF`

---

### `OI_SetPositionT`

**Syntax**

```c
OI_API OI_SetPositionT(double dNewPos)
```

```c
OI_API OI_SetPositionS(double dNewPos)
```

**Description**
Sets the current position for the axis.

**Parameters**

- `dNewPos` The desired axis position, in microns. The current location will be defined to be this position.

**Return Value**

- `OI_OK` if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**
The `OI_SetPositionT` function resets the coordinate system for the T axis. The current T position is redefined to be the values passed in \(dT\).

Note that the physical positions of the limits for the axis are retained by this function. That is, the `OI_SetPositionT` function maintains the same relation between the current position and the position of the negative and positive soft limits. Therefore, the coordinates values associated for these limits will be changed if a new position value is specified for the axis.

**See Also**
`OI_SetPositionXYZ`, `OI_SetPositionXY`, `OI_SetPositionZ`, `OI_SetOriginXY`, `OI_SetOriginZ`, `OI_SetOriginF`, `OI_InitializeXY`, `OI_InitializeZ`, `OI_InitializeF`
**OI_SetRampT**

Syntax

```c
OI_API OI_SetRampT(int nRamp)

OI_API OI_SetRampS(int nRamp)
```

Description

Specifies which pre-defined acceleration / deceleration ramp is in use for the axis.

Parameters

- `nRamp` Identifies the pre-defined ramp to be used.

Return Value

- `OI_OK` if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

Comments

Three pre-defined tables may be selected for a given axis, as indicated by the `nRamp` parameter:

<table>
<thead>
<tr>
<th>nRamp value</th>
<th>Acceleration</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Slow</td>
</tr>
<tr>
<td>1</td>
<td>Medium</td>
</tr>
<tr>
<td>2</td>
<td>Fast</td>
</tr>
</tbody>
</table>

See Also

- `OI_GetRampF`, `OI_SetRampXY`, `OI_SetRampF`, `OI_SetAxisRamp`

---

**OI_SetUserLimitsT**

Syntax

```c
OI_API OI_SetUserLimitsT(double dMin, double dMax)

OI_API OI_SetUserLimitsS(double dMin, double dMax)
```

Description

Sets user-defined limits of travel along the axis.

Parameters

- `dMin` The minimum coordinate for the axis, in microns.
- `dMax` The maximum coordinate for the axis, in microns.

Return Value

- `OI_OK` if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

Comments

The `OI_SetUserLimitsT` functions allows “soft” limits to be set at any point along the T axis. Once the soft limits are set, the OASIS controller will not allow any movement outside of these limit values.

Note that the soft limits are distinct from the physical limit switches of the stage.
The “hard” physical limit switches provide direct electronic feedback to the OASIS controller indicating the physical limits of travel available for the stage.

When using the `OI_InitializeTRange` function to initialise the range of travel and position of the stage, the OASIS controller automatically sets the current position to 0 and also sets the T user limits to positions based on the ranges passed to the function.

See Also `OI_GetUserLimitsT`, `OI_InitializeTRange`

---

**OI_StepT**

**Syntax**

```
OI_API OI_StepT(double dDistance, int nWait)

OI_API OI_StepS(double dDistance, int nWait)
```

**Description**

Moves a relative distance from the current position.

**Parameters**

- `dDistance`
  The desired distance for the move, in microns.

- `nWait`
  A flag indicating whether the function waits for the move to be completed before returning.

**Return Value**

- `OI_OK` if successful.

- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

If the `nWait` parameter is set to 0 for the call, the function returns immediately, i.e., it does not wait for the move to complete. You can use the `OI_WaitForStoppedT` function also to delay execution until a move is complete.

See Also `OI_WaitForStoppedT`, `OI_MoveToT`

---

**OI_WaitForStoppedT**

**Syntax**

```
OI_API OI_WaitForStoppedT (void)

OI_API OI_WaitForStoppedS (void)
```

**Description**

Waits for the axis to stop moving.

**Parameters**

- None.

**Return**

- `OI_OK` if successful.
Value
If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

Comments
The `OI_WaitForStoppedT` function is useful after any T axis move functions are called with zero wait parameters. `OI_WaitForStoppedT` will not return until the T axis has completed its movement.

See Also
`OI_StepT`, `OI_MoveToT`

---

## Encoders and Closed-loop Operation

The OASIS controller supports encoder signals inputs for the X, Y, Z and F axes. Encoders provide positional feedback from rotary, linear, or grid encoder devices, which can be used by the OASIS to provide accurate position information as well as for closed-loop operations to improve movement precision.

The presence and specifications of encoders are configured within the OASIS flash memory, and must be set using the OASIS Flash Memory Configuration utility application. The OASIS software library includes facilities for enquiring about the encoder setup, enabling whether encoder feedback is to be used for position information, and defining closed-loop operation.

### OI_GetAxisEncoderEnabled

**Syntax**

```
OI_API OI_GetAxisEncoderEnabled( int AxisID, LPBOOL lpbEnabled,
                              LPBOOL lpbAutoCorrect )
```

**Description**

Retrieves the status of the encoder counter enabling.

**Parameters**

- `AxisID` The desired axis (see the introduction of this section for the appropriate constants).
- `lpbEnabled` Returns whether the encoder counter is enabled.
- `lpbAutoCorrect` Returns whether moves are automatically corrected to the nearest encoder position.

**Return Value**

- `OI_OK` if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

If encoders are fitted, their use by the OASIS controller may be enabled or disabled via software. The `OI_GetAxisEncoderEnabled` and `OI_SetAxisEncoderEnabled` functions deal with these settings.
See Also  
OI_SetAxisEncoderEnabled, OI_GetAxisEncoderFitted,  
OI_GetAxisEncoderStepSize, OI_SetEncoderEnabledXY,  
OI_GetEncoderEnabledXY, OI_SetEncoderEnabledZ,  
OI_GetEncoderEnabledZ

### OI_GetAxisEncoderFitted

**Syntax**

```c
OI_API OI_GetAxisEncoderFitted( int AxisID, LPBOOL lpbFitted )
```

**Description**

Retrieves whether an encoder has been configured as fitted for a given axis.

**Parameters**

- **AxisID**  
  The desired axis (see the introduction of this section for the appropriate constants).

- **lpbFitted**  
  Returns whether the encoder counter is enabled.

**Return Value**

- **OI_OK** if successful.

- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

The Flash memory of the controller is used to indicate that an encoder has been fitted as well as the encoder to microstep ratio. The `OI_GetAxisEncoderFitted` function returns whether the configuration of encoders for a given axis is found in the Flash memory.

See Also  
OI_GetAxisEncoderStepSize, OI_GetAxisEncoderEnabled,  
OI_SetAxisEncoderEnabled, OI_SetEncoderEnabledXY,  
OI_GetEncoderEnabledXY, OI_SetEncoderEnabledZ,  
OI_GetEncoderEnabledZ

### OI_GetAxisEncoderStepSize

**Syntax**

```c
OI_API OI_GetAxisEncoderStepSize( int AxisID, double *pdMicrons )
```

**Description**

Retrieves the step size in microns for a given axis’ encoder, if fitted.

**Parameters**

- **AxisID**  
  The desired axis (see the introduction of this section for the appropriate constants).

- **pdMicrons**  
  Returns the encoder steps size, in microns.

**Return Value**

- **OI_OK** if successful.

- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.
Comments
The encoder factor, i.e., the number of microsteps per encoder step, found in
the Flash memory is used in combination with the axis calibration to return the
encoder step size in microns.

See Also
OI_GetAxisEncoderFitted, OI_GetAxisEncoderEnabled,
OI_SetAxisEncoderEnabled, OI_SetEncoderEnabledXY,
OI_GetEncoderEnabledXY, OI_SetEncoderEnabledZ,
OI_GetEncoderEnabledZ

OI_GetEncoderClosedLoopResponseXYZ

Syntax
OI_API OI_GetEncoderClosedLoopResponseXYZ ( int* pnResponseX, int* pnResponseY, int* pnResponseZ )

Description
Returns the closed loop response rate for XYZ.

Parameters

<table>
<thead>
<tr>
<th>pnResponseX</th>
<th>Returns the closed loop response rate for the specified axis, as follows:</th>
</tr>
</thead>
<tbody>
<tr>
<td>pnResponseY</td>
<td>0 = slow</td>
</tr>
<tr>
<td>pnResponseZ</td>
<td>1 = fast</td>
</tr>
</tbody>
</table>

Return Value
OI_OK if successful.
If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

Comments
When using closed loop operations, the controller will compare the actual position with the desired position at the end of each move. The OI_GetEncoderClosedLoopResponseXYZ function returns the response rate per axis (XYZ).

See Also
OI_SetEncoderClosedLoopResponseXYZ, OI_SetEncoderEnabledZ,
OI_SetEncoderEnabledXY, OI_GetEncoderEnabledXY

OI_GetEncoderEnabledXY

Syntax
OI_API OI_GetEncoderEnabledXY( LPBOOL pbEnabledX, int* pnToiX,
LPBOOL pbEnabledY, int* pnToiY )

Description
Defines the encoder setup for the X and Y axes.

Parameters

| pbEnabledX | Returns the enabled state of the X axis encoder. |
| pnToiX     | Returns the tolerance of the closed-loop mode for |
the X axis encoder.

\[ pbEnabledY \]
Returns the enabled state of the Y axis encoder.

\[ pnTolY \]
Returns the tolerance of the closed-loop mode for the Y axis encoder.

Return Value
OI_OK if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

Comments
See the OI_SetEncoderEnabledXY function for further details regarding the setup of the encoders and closed-loop operation.

See Also
OI_SetEncoderEnabledXY, OI_SetEncoderEnabledZ, OI_GetEncoderEnabledZ

---

**OI_GetEncoderEnabledZ**

Syntax

\[ OI_API OI_GetEncoderEnabledZ( BOOL bEnabledZ, int nTolZ ) \]

Description
Reads the encoder setup for the Z axis.

Parameters

\[ pbEnabledZ \]
Returns the enabled state of the Z axis encoder.

\[ pnTolZ \]
Returns the tolerance of the closed-loop mode for the Z axis encoder.

Return Value
OI_OK if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

Comments
See the OI_SetEncoderEnabledZ function for further details regarding the setup of the encoders and closed-loop operation.

See Also
OI_SetEncoderEnabledZ, OI_SetEncoderEnabledXY, OI_GetEncoderEnabledZ

---

**OI_ReadEncoderModule**

Syntax

\[ OI_API OI_ReadEncoderModule( LPWORD pwInfo, LPWORD pwOptions, LPWORD pwOutputPulseWidth_usec, LPDWORD pdwComparitor1, LPDWORD pdwComparitor2, LPDWORD pdwReserved, LPDWORD pdwReserved2 ) \]

Description
Retrieves the information for the BLUE-EXPIO triggering.
Parameters

- **pwInfo**
  BLUE-EXPIO module ID Register, 8 bits, provides Module/Altera version information - 0xFF = Module not fitted.

- **pwOptions**
  BLUE-EXPIO control register value.

- **pwOutputPulseWidth_usec**
  Output pulse width, in microseconds.

- **pdwComparitor1**
  Comparator 1 interval, in encoder counts.

- **pdwComparitor2**
  Comparator 2 interval, in encoder counts.

- **pdwReserved, pdwReserved2**
  Unused.

Return Value

- **OI_OK** if successful.

  If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

Comments

See [OI_SetEncoderModule](#) for details regarding the BLUE-EXPIO operation.

See Also

- [OI_SetEncoderModule](#)
- [OI_SetEncoderClosedLoopResponseXYZ](#)

---

**OI_SetEncoderClosedLoopResponseXYZ**

**Syntax**

```
OI_API OI_SetEncoderClosedLoopResponseXYZ ( int nResponseX, int nResponseY, int nResponseZ )
```

**Description**

Specifies the closed loop response rate for XYZ.

**Parameters**

- **nResponseX**
  The closed loop response rate for the specified axis, as follows:

  - 0 = slow
  - 1 = fast

- **nResponseY**

- **nResponseZ**

**Return Value**

- **OI_OK** if successful.

  If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

When using closed loop operations, the controller will compare the actual position with the desired position at the end of each move. The final correction is by default at a slow rate, but the [OI_SetEncoderClosedLoopResponseXYZ](#) function allows you to specify either fast or slow response per axis (XYZ).

**See Also**

- [OI_GetEncoderClosedLoopResponseXYZ](#)
- [OI_SetEncoderEnabledZ](#)
- [OI_SetEncoderEnabledXY](#)
- [OI_GetEncoderEnabledXY](#)
## OI_SetEncoderEnabledXY

### Syntax

```
OI_API OI_SetEncoderEnabledXY( BOOL bEnabledX, int nTolX, BOOL bEnabledY, int nTolY )
```

### Description

Defines the encoder setup for the X and Y axes.

### Parameters

- **bEnabledX**: Enables or disables use of encoder feedback on the X axis. When enabled, all read operations for the X axis will use encoder counter data.

- **nTolX**: Sets the tolerance for X axis closed-loop operation, as follows:
  
  - 0 = Closed-loop disabled, encoders only used for position information.
  
  - >0 = Closed-loop enabled, encoder information will be used during move operations to ensure position is maintained to within the given counter resolution.

- **bEnabledY**: Enables or disables use of encoder feedback on the Y axis. When enabled, all read operations for the Y axis will use encoder counter data.

- **nTolY**: Sets the tolerance for Y axis closed-loop operation, as follows:
  
  - 0 = Closed-loop disabled, encoders only used for position information.
  
  - >0 = Closed-loop enabled, encoder information will be used during move operations to ensure position is maintained to within the given counter resolution.

### Return Value

- **OI_OK** if successful.

  If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

### Comments

The encoder signals may be used simply as a more accurate means of determining the current position, or they can be used to actively monitor that movements to a given position are ensured using the encoder feedback.

When in closed-loop mode, the OASIS controller will continuously monitor the current encoder information to ensure that the desired current position is maintained from external forces, such as hand movements of the stage independent of the controller facilities, etc.

To configure the controller to use encoders just for position readout, but not for closed-loop operation, set the enabled flag for the axis to TRUE and the associated tolerance to zero.
When using the closed-loop mode, you may need to adjust the tolerance value to minimize oscillation of the motor as the controller continuously applies correctional movements. For instance, for high resolution encoders, small vibration may cause slight chances in the encoder values. In these situations the tolerance should be set sufficiently high so as to be larger than the vibration variations seen on the system.

See Also
OI_GetEncoderEnabledXY, OI_SetEncoderEnabledZ, OI_GetEncoderEnabledZ

**OI_SetEncoderEnabledZ**

**Syntax**

```c
OI_API OI_SetEncoderEnabledZ( BOOL bEnabledZ, int nTolZ )
```

**Description**

Defines the encoder setup for the Z axis.

**Parameters**

- `bEnabledZ` Enables or disables use of encoder feedback on the Z axis. When enabled, all read operations for the Z axis will use encoder counter data.

- `nTolZ` Sets the tolerance for Z axis closed-loop operation, as follows:
  - `0` = Closed-loop disabled, encoders only used for position information.
  - `>0` = Closed-loop enabled, encoder information will be used during move operations to ensure position is maintained to within the given counter resolution.

**Return Value**

- `OI_OK` if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

The encoder signals may be used simply as a more accurate means of determining the current position, or they can be used to actively monitor that movements to a given position are ensured using the encoder feedback.

When in closed-loop mode, the OASIS controller will continuously monitor the current encoder information to ensure that the desired current position is maintained from external forces, such as hand movements of the stage independent of the controller facilities, etc.

To configure the controller to use encoders just for position readout, but not for closed-loop operation, set the enabled flag for the axis to TRUE and the associated tolerance to zero.

When using the closed-loop mode, you may need to adjust the tolerance value to minimize oscillation of the motor as the controller continuously applies correctional movements. For instance, for high resolution encoders, small vibration may cause slight chances in the encoder values. In these situations
the tolerance should be set sufficiently high so as to be larger than the vibration variations seen on the system.

See Also  
OI_GetEncoderEnabledZ, OI_SetEncoderEnabledXY, OI_GetEncoderEnabledXY

---

### OI_SetEncoderModule

**Syntax**

```c
OI_API OI_SetEncoderModule (WORD wOutputPulseWidth_usec,
                            WORD wComparitor1On, DWORD dwComparitor1Value,
                            WORD wComparitor2On, DWORD dwComparitor2Value,
                            WORD wReserved, DWORD dwReserved, WORD wReserved2,
                            DWORD dwReserved2, WORD wOptions)
```

**Description**

Defines the operation of the BLUE-EXPIO comparitors.

**Parameters**

- `wOutputPulseWidth`  
The output signal pulse width, in microseconds.
- `wComparitor1On`  
Enables Comparator 1 output. Set to 1 to enable, 0 (zero) to disable.
- `dwComparitor1Value`  
The encoder count period for the output of Comparator 1.
- `wComparitor2On`  
Enables Comparator 2 output. Set to 1 to enable, 0 (zero) to disable.
- `dwComparitor2Value`  
The encoder count period for the output of Comparator 2.
- `wReserved, dwReserved, wReserved2, dwReserved2`  
Unused.
- `wOptions`  
BLUE-EXPIO control register parameters.

**Return Value**

- `OI_OK` if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

The OASIS-blue controller, when fitted with the BLUE-EXPIO encoder and trigger module, provides advanced functions for synchronizing motion control with external device triggers.

The `OI_SetEncoderModule` function allows the BLUE-EXPIO module's trigger output logic to be configured. The BLUE-EXPIO has two general purpose comparators that may be used to accumulate encoder input signals and send output trigger pulses at user-specified intervals.

Comparator 1 may be assigned to either X or F axis encoder inputs, while
Comparator 2 may be assigned to either Y or F axis encoder inputs, as specified by \textit{wOptions} bits 8 and 9, respectively.

The encoder signals may be either RS422 or TTL, as set by the \textit{wOptions} bits 4-7 (see table below). The output pulse width is specified in microseconds.

The ultimate aim is to provide hardware-based synchronization of output trigger signals to positions on any of the axes of the OASIS-blue controller. One application for instance is to enhance image acquisition by allowing continuous movement with trigger synchronization of a digital camera while moving X or Y, for mosaic image acquisition, or Z, for Z-stack acquisition.

The BLUE-EXPIO control register defines the behaviour of the trigger output signals. The control WORD bits are defined as follows:

<table>
<thead>
<tr>
<th>\textit{wOptions Bit}</th>
<th>\textit{Meaning}</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>Comparator 2 input select.</td>
</tr>
<tr>
<td>0</td>
<td>Comparator 2 generated from Y input</td>
</tr>
<tr>
<td>1</td>
<td>Comparator 2 generated from Z input</td>
</tr>
<tr>
<td>8</td>
<td>Comparator 1 input select.</td>
</tr>
<tr>
<td>0</td>
<td>Comparator 1 generated from X input</td>
</tr>
<tr>
<td>1</td>
<td>Comparator 1 generated from F input</td>
</tr>
<tr>
<td>7</td>
<td>F encoder input type.</td>
</tr>
<tr>
<td>0</td>
<td>RS422</td>
</tr>
<tr>
<td>1</td>
<td>TTL</td>
</tr>
<tr>
<td>6</td>
<td>Z encoder input type.</td>
</tr>
<tr>
<td>0</td>
<td>RS422</td>
</tr>
<tr>
<td>1</td>
<td>TTL</td>
</tr>
<tr>
<td>5</td>
<td>Y encoder input type.</td>
</tr>
<tr>
<td>0</td>
<td>RS422</td>
</tr>
<tr>
<td>1</td>
<td>TTL</td>
</tr>
<tr>
<td>4</td>
<td>X encoder input type.</td>
</tr>
<tr>
<td>0</td>
<td>RS422</td>
</tr>
<tr>
<td>1</td>
<td>TTL</td>
</tr>
<tr>
<td>[3..2]</td>
<td>Position latch function.</td>
</tr>
<tr>
<td>0</td>
<td>XYZF position latch via DSP</td>
</tr>
<tr>
<td>1</td>
<td>XYZF position latch via trigB input on BLUE-CONNECT PCB.</td>
</tr>
<tr>
<td>2</td>
<td>XYZF position latch via trigA input on BLUE-CONNECT PCB.</td>
</tr>
<tr>
<td>[1..0]</td>
<td>General-purpose comparator function select.</td>
</tr>
<tr>
<td>0</td>
<td>Comparator trigger on X axis.</td>
</tr>
<tr>
<td>1</td>
<td>Comparator trigger on Y axis.</td>
</tr>
<tr>
<td>2</td>
<td>Comparator trigger on Z axis.</td>
</tr>
</tbody>
</table>
Automatic focus can be performed automatically by the OASIS controller providing that the OASIS-AF option is installed and a suitable video camera is fitted to the system.

### OI_AutoFocus

**Syntax**

OI_API OI_AutoFocus(void)

**Description**

Performs an automatic focus using the current autofocus parameters.

**Parameters**

None.

**Return Value**

OI_OK if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

The OI_AutoFocus function returns immediately. To wait until the autofocus operation is complete, follow the call to OI_AutoFocus with a call to OI_WaitForAutoFocus.

**See Also**

OI_SetAutoFocus, OI_AutoFocusEx, OI_WaitForAutoFocus

### OI_AutoFocus_FINE

**Syntax**

OI_API OI_AutoFocus_FINE( double dRange, int nSamples, double dStepSize )

**Description**

Performs a fine focus, by searching for the optimal focus from the current position.

**Parameters**

- **dRange**
  
  The range of positions over which to search for focus.

- **nSamples**
  
  The number of focus threshold measurements to average at each position.

- **dStepSize**
  
  The Z step distance between discrete moves during
the autofocus operation.

Return Value

OI_OK if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

Comments

The `OI_AutoFocus_Fine` function differs from the `OI_AutoFocusEx` function. Instead of the continuous sweep through the specified autofocus search range, the `OI_AutoFocus_Fine` function uses a searching algorithm from the current Z position to fine the best focus. The algorithm may be summarised as follows:

1. Measure the current focus score;
2. Step in a given direction, and continue while the focus score improves;
3. If no improvement was seen in the direction of Step 2, step in the opposite direction while the focus score improves.

The `dStepSize` parameter specifies the distance to move between measurements of the focus score. The `nSamples` parameter allows focus score values to be averaged at each measurement, reducing the effects of noise.

The searching is limited by the `dRange` parameter, preventing the focus operation from moving farther than a given distance.

In some situations, this may produce more accurate results, although the overall time taken to achieve focus is usually increased.

See Also OI_WaitForAutoFocus, OI_AutoFocusEx, OI_AutoFocus

---

**OI_AutoFocus_Step**

Syntax

```c
OI_API OI_AutoFocus_Step( double dRange, int nSpeed, int nTolerance, double dStepSize )
```

Description

Performs a stepwise automatic focus.

Parameters

- `dRange` The range of positions over which to search for focus.
- `nSpeed` The cruise speed at which the Z axis is moved during the focus search.
- `nTolerance` The peak-definition tolerance value.
- `dStepSize` The Z step distance between discrete moves during the autofocus sweep.

Return

OI_OK if successful.
If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

The OI_AutoFocus_Step function differs from the OI_AutoFocusEx function. Instead of the continuous sweep through the specified autofocus search range, the OI_AutoFocus_Step function moves through the range in discrete steps, as given by the dStepSize parameter.

In some situations, this may produce more accurate results, although the overall time taken to achieve focus is usually increased.

See Also OI_WaitForAutoFocus, OI_AutoFocusEx, OI_AutoFocus

OI_AutoFocusEx

Syntax

OI_API OI_AutoFocusEx( double dRange, int nSpeed, int nTolerance )

Description

Performs an automatic focus using the specified parameters.

Parameters

- **dRange**: The range of positions over which to search for focus.
- **nSpeed**: The cruise speed at which the Z axis is moved during the focus search.
- **nTolerance**: The peak-definition tolerance value.

Return Value

- **OI_OK** if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

Comments

The OI_AutoFocusEx function returns immediately. To wait until the autofocus operation is complete, follow the call to OI_AutoFocus with a call to OI_WaitForAutoFocus.

See Also OI_AutoFocus, OI_SetAutoFocus, OI_WaitForAutoFocus

OI_GetAutoFocus

Syntax

OI_API OI_GetAutoFocusThreshold(double* pdRange, int* pnSpeed, int* pnTolerance)

Description

Retrieves the current AutoFocus settings.

Parameters

- **pdRange**: The returned range of travel for the autofocus, in microns.
**pnSpeed**

The returned cruise speed at which the focus is moved.

**pnTolerance**

The returned peak-finding tolerance value.

**Return Value**

- OI_OK if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

See the Comments for the **OI_AutoFocusEx** for more information about the AutoFocus settings.

**See Also**

- **OI_AutoFocusEx**, **OI_GetAutoFocusThreshold**

---

### OI_GetAutoFocusEx

**Syntax**

```c
OI_API OI_GetAutoFocusEx( double* pdRange, int* pnSpeed, int* pnTolerance, double *pdStepSize )
```

**Description**

Retrieves the current AutoFocus extended settings.

**Parameters**

- **pdRange**
  
  The range of positions over which to search for focus.

- **pnSpeed**
  
  The cruise speed at which the Z axis is moved during the focus search.

- **pnTolerance**
  
  The peak-definition tolerance value.

- **pdStepSize**
  
  The step size in microns between samples.

**Return Value**

- OI_OK if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

The **OI_GetAutoFocusEx** function extends the **OI_GetAutoFocus** function by also returning the fine focus step size.

**See Also**

- **OI_AutoFocusEx**, **OI_AutoFocus_Fine**, **OI_AutoFocus_Step**, **OI_AutoFocus**, **OI_SetAutoFocusEx**, **OI_SetAutoFocus**, **OI_WaitForAutoFocus**

---

### OI_GetAutoFocusThreshold

**Syntax**

```c
OI_API OI_GetAutoFocusThreshold(int* pnThresh)
```
### OI_GetFineFocusSamples

<table>
<thead>
<tr>
<th>Syntax</th>
<th>OI_API OI_GetFineFocusSamples(int* pnSamples)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Retrieves the number of focus score samples to be taken at each step in the fine focus operation.</td>
</tr>
<tr>
<td>Parameters</td>
<td>nSamples The number of focus score samples.</td>
</tr>
<tr>
<td>Return Value</td>
<td>OI_OK if successful. If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.</td>
</tr>
<tr>
<td>Comments</td>
<td>See the OI_SetFineFocusSamples function for a description of the focus samples.</td>
</tr>
<tr>
<td>See Also</td>
<td>OI_SetFineFocusSamples, OI_AutoFocus_Fine</td>
</tr>
</tbody>
</table>

### OI_ReadFocusProfile

<table>
<thead>
<tr>
<th>Syntax</th>
<th>OI_API OI_ReadFocusProfile(double* pdScores, double* pdZPos, int nSize, int* pnSamples)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Reads the AutoFocus profile from the last automatic focus operation.</td>
</tr>
<tr>
<td>Parameters</td>
<td>pdScores Pointer to an array, of length nSize, to receive the focus score values.</td>
</tr>
<tr>
<td></td>
<td>pdZPos Pointer to an array, of length nSize, to receive the Z position values.</td>
</tr>
<tr>
<td></td>
<td>nSize The length of the pdScores and pdZPos arrays.</td>
</tr>
</tbody>
</table>
The returned actual number of samples.  

**Return Value**  
OI_OK if successful.  
If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.  

**Comments**  
The automatic focus works by sweeping over a given Z range whilst continuously sampling the focus score. This process generates a list of Score vs. Z-Position value pairs.  

The **OI_ReadFocusProfile** function will return these pairs of values in two arrays, one for the focus scores (returned in the *pdScores* array) and another for the corresponding Z positions (returned in the *pdZPos* array).  

The actual number of samples achieved during the automatic focus process is dictated by the video rate, the size of the Z range used, and the speed at which the Z axis was driven. A larger Z range and/or a slower speed will allow more samples to be taken.  

The actual number of samples taken during the last AutoFocus operation is returned in the *pnSamples* value. The *nSize* parameter is passed into the function by the caller to indicate the size of the destination arrays. If the total number of samples taken during the AutoFocus was greater than the *nSize* value, only the first *nSize* values are returned in the arrays. If the total number of samples is less than the *nSize* value, only the first *pnSamples* values contain valid data.  

**See Also**  
**OI_AutoFocus, OI_AutoFocusEx, OI_AutoFocus_Step, OI_SetAutoFocus**

---

**OI_ReadFocusScore**

**Syntax**  
OI_API OI_ReadFocusScore(double* pdScore)

**Description**  
Read the current focus score calculation from the incoming video source.

**Parameters**  
*pdScore*  
The returned focus score.

**Return Value**  
OI_OK if successful.  
If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.  

**Comments**  
Higher focus score values indicate greater sharpness of focus.

**See Also**  
**OI_ReadFocusScoreEx, OI_SetAutoFocusThreshold**
**OI_ReadFocusScoreEx**

Syntax

OI_API OI_ReadFocusScoreEx(double* pdScore, double *pdZPos)

Description
Read the current focus score calculation from the incoming video source and the Z axis position corresponding to that reading.

Parameters

- *pdScore*: The returned focus score.
- *pdZPos*: The returned Z position.

Return Value

- OI_OK if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

Comments
Higher focus score values indicate greater sharpness of focus.

See Also
  - OI_ReadFocusScore, OI_SetAutoFocusThreshold

---

**OI_RequestAutoFocusStatus**

Syntax

OI_API OI_RequestAutoFocusStatus(LPWORD pwFinished, LPWORD pwSuccess, LPWORD pwNumSamples)

Description
Read the status of the last automatic focus operation.

Parameters

- *pwFinished*: The returned status indicating whether the previous autofocus is complete.
- *pwSuccess*: The returned success status of the last autofocus.
- *pwNumSamples*: The returned number of video samples taken during the autofocus.

Return Value

- OI_OK if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

Comments
The **OI_RequestAutoFocusStatus** function returns the details regarding a previously executed autofocus operation.

See Also
  - OI_AutoFocus, OI_AutoFocusEx, OI_AutoFocus_Step
**OI_SetAutoFocus**

**Syntax**

```c
OI_API OI_SetAutoFocus( double dRange, int nSpeed, int nTolerance )
```

**Description**

Sets the default focus parameters.

**Parameters**

- `dRange` The range of positions over which to search for focus.
- `nSpeed` The cruise speed at which the Z axis is moved during the focus search.
- `nTolerance` The peak-definition tolerance value.

**Return Value**

- `OI_OK` if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

The focus search will be performed using a continuous sweep over the given micron range at the indicated cruise speed. The tolerance value may be adjusted for improved peak-finding.

**See Also**

`OI_AutoFocus`, `OI_AutoFocusEx`

**OI_SetAutoFocusEx**

**Syntax**

```c
OI_API OI_SetAutoFocusEx( double dRange, int nSpeed, int nTolerance,
                          double dStepSize )
```

**Description**

Sets the default focus parameters, including step size.

**Parameters**

- `dRange` The range of positions over which to search for focus.
- `nSpeed` The cruise speed at which the Z axis is moved during the focus search.
- `nTolerance` The peak-definition tolerance value.
- `dStepSize` The step size in microns between samples.

**Return Value**

- `OI_OK` if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

This function extends the `OI_SetAutoFocus` function by adding the step size parameter that is used with the `OI_AutoFocus_Step` and `OI_AutoFocus_Fine` functions.
### OI_SetAutoFocusThreshold

**Syntax**  
OI_API OI_SetAutoFocusThreshold(int nThresh)

**Description**  
Sets the threshold value used when generating the focus score from the incoming video signal.

**Parameters**  
- nThresh: The threshold value, ranging from 0 to 255.

**Return Value**  
OI_OK if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**  
The AutoFocus threshold may be used to reduce the effects of noise present in the image when calculating the focus score.

The focus score is generated by measuring the overall sharpness of edges in the image. Noise in the image may introduce false peaks in the AutoFocus curve measured during automatic focus operation.

Increasing the AutoFocus threshold may reduce the effect of noise, and therefore improve the performance of automatic focus.

**See Also**  
OI_GetAutoFocusThreshold, OI_AutoFocus, OI_AutoFocusEx, OI_AutoFocus_Step

### OI_SetFineFocusSamples

**Syntax**  
OI_API OI_SetFineFocusSamples(int nSamples)

**Description**  
Sets the number of focus score samples to be taken at each step in the fine focus operation.

**Parameters**  
- nSamples: The number of focus score samples.

**Return Value**  
OI_OK if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**  
The OI_AutoFocus_Fine function will perform an autofocus by searching from the current position to determine if the focus improves. This is made in discrete steps of a given size. You can use the OI_SetFineFocusSamples function to set the default number of focus score readings to be made at each step. These
readings will be averaged to produce the final score. Higher samples may reduce the effects of noise in the video signal.

See Also  
OI_GetFineFocusSamples, OI_AutoFocus_Fine

---

**OI_WaitForAutoFocus**

**Syntax**  
OI_API OI_WaitForAutoFocus(void)

**Description**  
Waits for any automatic focus operations to complete before returning.

**Parameters**  
None.

**Return Value**  
OI_OK if successful.

OI_ABORT if aborted by the user.

OI_TIMEOUT if the operation times out before a focus is found.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**  
In order to facilitate multitasking, all automatic focus functions return immediately. This, for instance, allows an AutoFocus operation to continue while the PC software is able to perform other, non-automation critical, functions such as the analysis of a previously acquired image or writing of data to disk.

However, at some point the application will need to ensure the automatic focus is complete. The OI_WaitForAutoFocus accomplishes this, as the function will not return until either the autofocus completes normally, is terminated by the user using the Escape key, or times out.

See Also  
OI_AutoFocus, OI_AutoFocusEx, OI_AutoFocus_Step

---

**Predictive Focus Functions**

The OASIS controller includes facilities to maintain a predicted focus position based on a first-order fit of focus Z-position as a function of X- and Y-positions. By supplying the OASIS library with three XYZ position values defining the in-focus Z-position at each XY location, the predicted focus plane can be calculated and used to determine the predicted, or expected, Z position as a function of X and Y. The OI_SetPredictiveZ function is used to define the three unique XYZ measurements defining the plane.
Once defined, the OI_GetPredictiveZ function will return the expected Z position for a given XY location.

Additionally, the OASIS controller provides an automatic predictive focus tracking facility, where the controller continuously monitors the XY and Z position, ensuring that the focus position is maintained in the expected predictive focus plane. The OI_SetAutoPredictiveZ function is used to enable and disable automatic predictive focus tracking.

More than 3 predictive focus points may also be used. The OI_SetMultiPredictiveZ function allows up to 256 points to be defined. The OASIS library divides the set of points into triangular regions, each region defining a focus plane within its 3 vertices. The OI_UpdatePredictiveZ function may be used to ensure the currently active plane corresponds to the current XY location.

Note that predictive focus is only available when the focus drive is being controlled directly by the OASIS. Predictive focus is not available for other focus drive controllers supported by the OASIS DLL, such as the Leica Microsystems DM microscope, Olympus BX-61, or the Leica Microsystems MZ motorfocus unit.

### OI_GetAutoPredictiveZ

**Syntax**

```c
OI_API OI_GetAutoPredictiveZ (LPBOOL pbFlag, LPBOOL pbValid, LPDOUBLE pdZ)
```

**Description**

Returns the status of the automatic predictive focus tracking.

**Parameters**

- `pbFlag` Flag indicating whether automatic predictive focus tracking is enabled or disabled.
- `pbValid` Flag indicating whether the predictive focus setup is valid.
- `pdZ` The predicted focus position for the current XY location.

**Return Value**

- OI_OK if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

The OI_GetAutoPredictiveZ function is used to determine whether the automatic predictive focus tracking is currently enabled and whether the predictive focus setup is valid. The function also returns the predicted focus value for the current XY position.

See the OI_SetAutoPredictiveZ for a full description of automatic focus tracking facilities.

**See Also**

- OI_SetAutoPredictiveZ
- OI_SetPredictiveZ
**OI_GetCoincDomain**

**Syntax**

```c
OI_API OI_GetCoincDomain( double X, double Y, int* pnDomainIndex )
```

**Description**

Returns the predicted Z domain for the given X and Y position values.

**Parameters**

- **X**
  - The X position, in microns.

- **Y**
  - The Y position, in microns.

- **pnDomainIndex**
  - Returns the predicted focus domain index.

**Return Value**

- **OI_OK** if successful.

- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

The `OI_GetCoincDomain` function returns the index of the predictive focus domain that coincides with the given X and Y stage location. The index corresponds to the domain list as returned by `OI_GetPredictiveZDomains`.

**See Also**

- `OI_GetPredictiveZDomains`

---

**OI_GetMultiPredictiveZ**

**Syntax**

```c
OI_API OI_GetMultiPredictiveZ( int* pnMethod, int* pnPoints,
                                 double *xvals, double *yvals, double *zvals )
```

**Description**

Returns the predictive focus information.

**Parameters**

- **pnMethod**
  - Returns the method to use to divide the sample area into regions.

- **pnPoints**
  - Returns the number of XYZ points used to define the predictive focus map. The maximum number of points supported is 256.

- **xvals**
  - Pointer to an array to receive the X-axis positions, in microns, representing the measurement X positions.

- **yvals**
  - Pointer to an array to receive the Y-axis positions, in microns, representing the measurement Y positions.

- **zvals**
  - Pointer to an array to receive the Z-axis positions, in microns, representing the measurement Z positions.

**Return Value**

- **OI_OK** if successful.
If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

Comments
See the `OI_SetMultiPredictiveZ` for more information regarding multi-point predictive focus.

See Also `OI_SetMultiPredictiveZ`, `OI_UpdatePredictiveZ`, `OI_InitializeZ`, `OI_MoveToZ`

---

### `OI_GetPredictiveFlag`

**Syntax**

```c
OI_API OI_GetPredictiveFlag (LPBOOL pbFlag)
```

**Description**

Returns whether the predictive focus has been previously defined.

**Parameters**

- `pbFlag`
  
  Returns TRUE if the predictive focus has been previously setup.

**Return Value**

- `OI_OK` if successful.

- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

The `OI_GetPredictiveFlag` function returns whether the predictive focus has been previously set up.

**See Also**

`OI_SetPredictiveZ`, `OI_SetPredictiveFlag`

---

### `OI_GetPredictiveZ`

**Syntax**

```c
OI_API OI_GetPredictiveZ (double X, double Y, double *pZ, int *pnStatus)
```

**Description**

Returns the predicted Z position for the given X and Y position values.

**Parameters**

- `X`
  
  The X position, in microns.

- `Y`
  
  The Y position, in microns.

- `pZ`
  
  Returns the predicted focus position value.

- `pnStatus`
  
  Returns the predictive focus status, as define in the Comments below.

**Return Value**

- `OI_OK` if successful.

- If unsuccessful, a combination of error codes may be returned to indicate the
The `OI_GetPredictiveZ` function returns the predicted focus position for a given X and Y stage location, as well as the current status of the predictive focus setup.

<table>
<thead>
<tr>
<th>nStatus value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The predictive focus has been defined and is valid.</td>
</tr>
<tr>
<td>1</td>
<td>Invalid predictive focus coefficients. This typically indicates the input positions set via the <code>OI_SetPredictiveZ</code> function did not contain three unique XYZ locations.</td>
</tr>
<tr>
<td>2</td>
<td>The predictive focus has not been defined.</td>
</tr>
</tbody>
</table>

See Also  
`OI_SetPredictiveZ`

---

**OI_GetPredictiveZDomains**

**Syntax**

```c
OI_API OI_GetPredictiveZDomains (int* pnDomains, double* ax, double* ay, double* az, double* bx, double* by, double* bz, double* cx, double* cy, double* cz )
```

**Description**

Returns the predictive focus domains.

**Parameters**

- `pnDomains`
  Returns the method to use to divide the sample area into regions.

- `ax, ay, az`
  Pointers to arrays to receive the first corner data, i.e., vertex `a`.

- `bx, by, bz`
  Pointers to arrays to receive the second corner data, i.e., vertex `b`.

- `cx, cy, cz`
  Pointers to arrays to receive the third corner data, i.e., vertex `c`.

**Return Value**

`OI_OK` if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

When using multi-point predictive focus, the OASIS library divides the sample area into triangular regions. This triangulation approximates the surface of the sample, with each triangle defining a facet where a linear interpolation of the plane is used to track focus.

Use the `OI_GetPredictiveZDomains` function to retrieve the
triangulation data. Each domain is defined by three sets of XYZ positions, i.e., the vertices of the triangle as defined by the OI_SetMultiPredictiveZ locations. Since vertices are shared by adjacent triangles, the number of domains is greater than the number of predictive focus points.

As shown in the diagram above, a given triangular domain has three vertices \( a, b, c \), each of which being defined by an XY stage location and the Z focus at that point. These vertices correspond to predictive focus points, but in this case are associated with the domains found by the triangulation of those points.

See Also OI_SetMultiPredictiveZ, OI_UpdatePredictiveZ, OI_GetCoincDomain

---

**OI_GetPredictiveZOffset**

**Syntax**

```
OI_API OI_GetPredictiveZOffset ( double* pdOffset )
```

**Description**

Returns the fixed offset for predictive focus operations.

**Parameters**

- `dOffset` The current offset, in microns.

**Return Value**

- `OI_OK` if successful.

- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

You can apply a fixed offset to predictive focus operations, if desired, using OI_SetPredictiveZOffset. Use OI_GetPredictiveZOffset to get the current value for the offset. The default is zero.

**See Also**

- OI_SetPredictiveZOffset, OI_SetPredictiveZ, OI_SetPredictiveFlag
**OI_InvalidatePredictiveZ**

**Syntax**

OI_API OI_InvalidatePredictiveZ()

**Description**

Forces the current predictive focus setup to be invalid.

**Parameters**

None.

**Return Value**

OI_OK if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

Automatic predictive focus tracking requires first that the predictive focus setup be defined using three sets of XYZ positions. The `OI_InvalidatePredictiveZ` function sets the predictive focus in an invalid, i.e., not set up, state.

**See Also**

OI_SetPredictiveZ, OI_GetAutoPredictiveZ

---

**OI_SetAutoPredictiveZ**

**Syntax**

OI_API OI_SetAutoPredictiveZ (BOOL bFlag)

**Description**

Turns on automatic predictive focus tracking.

**Parameters**

bFlag

Flag that enables and disables automatic predictive focus tracking.

**Return Value**

OI_OK if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

In particular, the OI_INVALIDCONFIG bit will be set if the Z-axis software limits are not set.

**Comments**

The OASIS controller can automatically perform movement of the Z-axis to ensure the Z-position is maintained to the predicted focus position based on the current XY position. When enabled, this predictive focus tracking will work during software commanded moves as well as during joystick operations, for high-speed maintenance of the predicted focus position.

The `OI_SetAutoPredictiveZ` function enables and disabled automatic predictive focus tracking.

Note that any commanded movement of the Z-axis, either by software or joystick/digiknob, will cause the automatic predictive focus tracking to be disabled.
Also, the Z-axis software limits must be set before attempting to enable automatic predictive focus.

See Also  
OI_SetPredictiveZ, OI_GetAutoPredictiveZ, OI_SetUserLimitsZ, OI_InitializeZ

OI_SetMultiPredictiveZ

Syntax

OI_API OI_SetMultiPredictiveZ (int nMethod, int nPoints, double *xvals, double *yvals, double *zvals)

Description

Sets the predictive focus locations using more than three points.

Parameters

nMethod  
The method to use to divide the sample area into regions. Use OI_PF_MULTI_PLANE.

nPoints  
The number of XYZ points used to define the predictive focus map, i.e., the length of arrays xvals, yvals, and zvals. The maximum number of points supported is 256.

xvals  
Pointer to array of X-axis positions, in microns, representing the measurement X positions.

yvals  
Pointer to array of Y-axis positions, in microns, representing the measurement Y positions.

zvals  
Pointer to array of Z-axis positions, in microns, representing the measurement Z positions.

Return Value

OI_OK if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

Comments

The OASIS controller’s Z-axis drive includes the capability of fitting the plane of focus as a function of XY location. This is based on measurements of Z-axis in-focus positions take at three XY locations. Given these three sets of XYZ positions, this plane can be calculated, enabling the OASIS controller’s DSP to maintain the resulting Z position continuously during XY movements.

The OASIS library supports more than just three points using the OI_SetMultiPredictiveZ function. The function accepts up to 256 points defining an in-focus Z position for each XY location. The library then uses these points to define regions, each a facet defined by a plane.

Use method OI_PF_MULTI_PLANE to perform this sub-division.

When more than 3 XYZ locations are used, calls to OI_GetPredictiveZ result in resolution of the sub-region corresponding to the given XY
position, with the returned predicted Z being the interpolated result in that region.

When automatic tracking of focus is desired (e.g., using `OI_SetAutoPredictiveZ`), calls to `OI_UpdatePredictiveZ` are required to resolve the current region and updated the OASIS controller to that region's plane.

See Also `OI_UpdatePredictiveZ`, `OI.InitializeZ`, `OI.MoveToZ`

---

**OI_SetPredictiveFlag**

**Syntax**

```c
OI_API OI_SetPredictiveFlag (BOOL bFlag)
```

**Description**
Sets the flag indicating whether the predictive focus has been previously defined.

**Parameters**

- `bFlag`
  Sets the status of the predictive focus flag. Set to FALSE to indicate that the predictive focus is no longer valid.

**Return Value**

- OI_OK if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

The `OI_SetPredictiveFlag` function may be used to programmatically set the status of the predictive focus flag.

**See Also** `OI_SetPredictiveZ`, `OI_SetPredictiveFlag`

---

**OI_SetPredictiveZ**

**Syntax**

```c
OI_API OI_SetPredictiveZ (double *pX, double *pY, double *pZ)
```

**Description**
Sets the predictive focus locations.

**Parameters**

- `pX`
  Pointer to array of three X-axis positions, in microns, representing the measurement locations X positions.
- `pY`
  Pointer to array of three Y-axis positions, in microns, representing the measurement locations Y positions.
- `pZ`
  Pointer to array of three Z-axis positions, in microns, representing the measurement locations Z positions.
locations Z positions.

**Return Value**

OI_OK if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

The OASIS controller’s Z-axis drive includes the capability of fitting the plane of focus as a function of XY location. This is based on measurements of Z-axis in-focus positions taken at three XY locations. Given these three sets of XYZ positions, this plane can be calculated, enabling the OASIS controller’s DSP to maintain the resulting Z position continuously during XY movements.

The **OI_SetPredictiveZ** function defines the predictive focus input locations, allowing the OASIS library to calculate the plane of focus as a function of XY stage. Three arrays of positions, one each for X, Y, and Z axes provide the input locations, i.e., pX[0] gives the X position of the first sampling location, pX[1] for the second sampling, and pX[2] for the third sampling.

Thus the coordinate \{pX[0], pY[0], pZ[0]\} represents the first XYZ location, \{pX[1], pY[1], pZ[1]\} is the second XYZ location, and \{pX[2], pY[2], pZ[2]\} is the third XYZ location.

The effect of the **OI_SetPredictiveZ** function is to define the coefficients of predictive focus for the current specimen. Calls to **OI_GetPredictiveZ** will return the predicted Z value for the given XY location. To enable automatic predictive focus tracking by the OASIS, call **OI_SetAutoPredictiveZ**.

**See Also**

OI_InitializeZ, OI_MoveToZ

---

**OI_SetPredictiveZOffset**

**Syntax**

```
OI_API OI_SetPredictiveZOffset ( double dOffset )
```

**Description**

Sets the fixed offset for predictive focus operations.

**Parameters**

- **dOffset**
  
  The desired offset, in microns.

**Return Value**

OI_OK if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

You can apply a fixed offset to predictive focus operations, if desired using **OI_SetPredictiveZOffset**.
See Also: OI_GetPredictiveZOffset, OI_SetPredictiveZ, OI_SetPredictiveFlag

**OI_UpdatePredictiveZ**

**Syntax**  
OI_API OI_UpdatePredictiveZ ( int nOption )

**Description**  
Forces the predictive focus region to be resolved for the current XY position.

**Parameters**  
nOption  
When set to 1, nOption causes the function to call OI_SetAutoPredictiveZ when finished updating to turn on predictive focus tracking.

**Return Value**  
OI_OK if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**  
In a multi-plane situation where more than 3 predictive focus points are used, the OI_UpdatePredictiveZ function may be used to ensure the current region is kept current. The OASIS controller can automatically track the focus given a plane, so when more than one plane is used, the OI_UpdatePredictiveZ function should be called when necessary to ensure the correct plane is currently active.

When the OI_SetAutoPredictiveZ function has been previously called to enable tracking, the nOption parameter may be zero, as the new plane will automatically become active. If predictive focus tracking is not enabled, you can set the nOption parameter to 1 to enable it within the OI_UpdatePredictiveZ call.

See Also: OI_SetMultiPredictiveZ, OI_SetAutoPredictiveZ, OI_SetPredictiveFlag

---

**Video Camera Interface Functions**

In addition to automatic focus facilities, OASIS-AF module also provides real-time measurements on detected features in the video field of view. A detection setting may be defined on the intensity of the video signal, allowing features of interested to be identified in the signal. These detected features may then be measured to provide total area, maximum chord length, and maximum gradient values at video rates.

This high-speed measurement provides substantial flexibility, as it allows specialized calculations to be made for customized autofocus operations. Furthermore, the measurement results can be used during the course of specimen scanning to identify very quickly if a given field has any information. This allows an application to rapidly skip blank fields, without having to incur the overhead of acquiring the image and apply measurements using software.
The OASIS-AF settings that are involved in video measurements are:

- The video window size and position
- The detection thresholds for the odd and even video fields
- The detection phase—i.e. light or dark—for the odd and even video fields

A full frame of standard analog video is actually composed of two interlaced video fields. The first video field reads the odd lines of video. A second pass reads the even lines of video. The combined results of these two scans give a full video frame of information. Therefore, each PAL video field is composed of 288 video lines, while NTSC video contains 240 lines. The successive scans of these lines lead to the typical full-frame vertical resolutions of 576 lines for PAL and 480 lines for NTSC.

Note that some modes of video results functions deal only with either the odd or even lines, therefore the actual vertical resolution is half the video window’s vertical resolution (in pixels).

The OASIS-AF module allows each video field (i.e., the odd or even video scan lines) to be treated separately. This means that you can define two distinct threshold values and phase definitions (light or dark) corresponding to the two separate video fields. These two settings are applied continuously to each successive video field by the OASIS-AF hardware, and the results may be read rapidly at any time.

---

**OI_GetVideoWindow**

**Syntax**

```
OI_API OI_GetVideoWindow( int* pnXStart, int* pnXStop, int* pnYStart, int* pnYStop )
```

**Description**

Reads the current settings for the video window placement.

**Parameters**

- `pnXStart` The start X value for the video window, in pixels.
- `pnXStop` The stop X value for the video window, in pixels.
- `pnYStart` The start Y value for the video window, in video field lines.
- `pnYStop` The stop Y value for the video window, in video field lines.

**Return Value**

- `OI_OK` if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

The video window defines the region in which all video measurements are made. This includes focus score calculations as well as detected area, maximum chord length, and maximum gradient measurements.

The OASIS-AF hardware places some restrictions on the positions and sizes that are possible for these settings.
The \texttt{nXStart} and \texttt{nXStop} values must be set on 4-pixel boundaries, e.g., 4, 8, 12, etc.

The \texttt{nYStart} and \texttt{nYStop} values are specified in terms of pixels. For instance, a PAL video field has a vertical resolution of 768 pixels, and an NTSC video field has a vertical resolution of 480 pixels.

The \texttt{nYStart} video window \textit{Y} start position is restricted to values from 0 to 254, i.e. roughly the top half of the video field.

The \texttt{nYStop} video window \textit{Y} stop position may take on any values up to the size of the available video field, i.e., up to 767 for PAL video and up to 479 for NTSC video signals.

\textbf{See Also} \texttt{OI\_SetVideoWindow, OI\_GetAFFitted, OI\_GetPCBStatus}

\section*{\texttt{OI\_IsVideoDetected}}

\textbf{Syntax}

\begin{verbatim}
OI\_API OI\_IsVideoDetected(BOOL* pbFound )
\end{verbatim}

\textbf{Description}

Reads whether a valid video signal has been detected.

\textbf{Parameters}

\begin{itemize}
  \item \textit{pbFound} Pointer to BOOL result indicating if video has been detected.
\end{itemize}

\textbf{Return Value}

\texttt{OI\_OK} if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

\textbf{Comments}

The \textit{pbFound} will be set to TRUE if an OASIS-AF module has been fitted and an appropriate incoming video signal has been detected.

If no video signal is detected, or if an OASIS-AF module has not been fitted, the \textit{pbFound} parameter will be set to FALSE.

\textbf{See Also} \texttt{OI\_GetAFFitted, OI\_ReadPCBStatus}

\section*{\texttt{OI\_ReadVideoData}}

\textbf{Syntax}

\begin{verbatim}
OI\_API OI\_ReadVideoData(LPWORD lpwStatus, LPDWORD pdwArea,
                        LPWORD pwMaxChord, LPWORD pwMaxGradient)
\end{verbatim}

\textbf{Description}

Reads the current video measurement data.

\textbf{Parameters}

\begin{itemize}
  \item \textit{lpwStatus} The status of the video measurements, as
\end{itemize}
pdwArea
The total count of pixels measured in the detected phase.

pwMaxChord
The pixel length of the longest chord in the detected phase.

pwMaxGradient
Not used.

Return Value
OI_OK if successful.
If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

Comments
The pnStatus variable indicates the video field for which the current results are valid, and is one of the following values:

<table>
<thead>
<tr>
<th>pnStatus Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The video measurement data is not available.</td>
</tr>
<tr>
<td>1</td>
<td>The returned results are for the Odd video field.</td>
</tr>
<tr>
<td>2</td>
<td>The returned results are for the Even video field.</td>
</tr>
</tbody>
</table>

See Also
OI_SetVideoSettings, OI_ReadVideoResults

OI_ReadVideoResults

Syntax
OI_API OI_ReadVideoResults( int nMode, LPDWORD pdwArea, LPWORD pwMaxChord)

Description
Reads the video measurement results for the odd video field, even video field, or a combination of both.

Parameters
nMode
Specifies the desired results, as described in the Comments below.

pdwArea
The total count of pixels measured in the detected phase.

pwMaxChord
The pixel length of the longest chord in the detected phase.

Return Value
OI_OK if successful.
OI_TIMEOUT if the desired results are not available within 200 msec.
If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.
**Comments**

The `nMode` parameter indicates which video field is to be read, using the following values:

<table>
<thead>
<tr>
<th><code>nMode</code> Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The function should return the Odd video field results.</td>
</tr>
<tr>
<td>2</td>
<td>The function should return the Even video field results.</td>
</tr>
<tr>
<td>3</td>
<td>The function should return a combination of the two fields, i.e., the sum of the even and odd field area measurements and the maximum of the even and odd chord lengths are returned. This provides a result for the entire video frame.</td>
</tr>
</tbody>
</table>

**See Also**  
OI_SetVideoSettings, OI_ReadVideoData

---

**OI_ReadVideoResultsEx**

**Syntax**

```c
OI_API OI_ReadVideoResultsEx( int nMode, LPDWORD pdwArea, LPWORD pwMaxChord, LPWORD pwMaxGradient)
```

**Description**

Reads the extended video measurement results for the odd video field, even video field, or a combination of both.

**Parameters**

- **`nMode`**  
  Specifies the desired results, as described in the Comments below.

- **`pdwArea`**  
  The total count of pixels measured in the detected phase.

- **`pwMaxChord`**  
  The pixel length of the longest chord in the detected phase.

- **`pwMaxGradient`**  
  The maximum gradient value in the detected phase.

**Return Value**

- OI_OK if successful.
- OI_TIMEOUT if the desired results are not available within 200 msec.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

The `nMode` parameter indicates which video field is to be read, using the following values:

<table>
<thead>
<tr>
<th><code>nMode</code> Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>
The function should return the Odd video field results.

The function should return the Even video field results.

The function should return a combination of the two fields, i.e., the sum of the even and odd field area measurements and the maximum of the even and odd chord lengths are returned. This provides a result for the entire video frame.

See Also OI_SetVideoSettings, OI_ReadVideoData

**OI_ReadVideoResultsXY**

**Syntax**

```c
OI_API OI_ReadVideoResultsXY( int nMode, double* pdResults, double* pdXPos, double* pdYPos )
```

**Description**

Reads the stage XY position and the extended video measurement results for the odd video field, even video field, or a combination of both.

**Parameters**

- **nMode**
  Specifies the desired results, as described in the Comments below.
- **pdResults**
  An array of doubles into which the video results are to be copied.
- **pdXPos**
  The position of the X Axis, in microns.
- **pdYPos**
  The position of the Y Axis, in microns.

**Return Value**

- OI_OK if successful.
- OI_TIMEOUT if the desired results are not available within the video timeout period.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

The **nMode** parameter indicates which video field is to be read, using the following values:

<table>
<thead>
<tr>
<th><strong>nMode Value</strong></th>
<th><strong>Meaning</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The function should return the Odd video field results.</td>
</tr>
<tr>
<td>2</td>
<td>The function should return the Even video field results.</td>
</tr>
</tbody>
</table>
The function should return a combination of the two fields, i.e., the sum of the even and odd field area measurements and the maximum of the even and odd chord lengths are returned. This provides a result for the entire video frame.

The `pdResults` argument is a pointer to an array of doubles. The calling application must ensure the array contains at least four elements. The video results are returned in the array in this order:

<table>
<thead>
<tr>
<th>pdResults Array Index</th>
<th>Video Result Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Focus score (full video frame)</td>
</tr>
<tr>
<td>1</td>
<td>Detected area</td>
</tr>
<tr>
<td>2</td>
<td>Detected maximum chord length</td>
</tr>
<tr>
<td>3</td>
<td>Detected maximum gradient</td>
</tr>
</tbody>
</table>

**See Also**
OI_SetVideoSettings, OI_ReadVideoData, OI_ReadVideoResults, OI_ReadVideoResultsEx, OI_ReadVideoResultsXYZF, OI_ReadVideoResultsZ

---

### OI_ReadVideoResultsXYZF

**Syntax**

```c
OI_API OI_ReadVideoResultsXYZF( int nMode, double* pdResults, double* pdPositions )
```

**Description**

Reads the stage XY position, focus Z position, and fourth axis position, as well as the extended video measurement results for the odd video field, even video field, or a combination of both.

**Parameters**

- **nMode**
  - Specifies the desired results, as described in the Comments below.

- **pdResults**
  - Array of doubles into which the video results are to be copied.

- **pdPositions**
  - Array of doubles into which the position data is to be copied.

**Return Value**

- OI_OK if successful.
- OI_TIMEOUT if the desired results are not available within the video timeout period.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.
### Comments

The `nMode` parameter indicates which video field is to be read, using the following values:

<table>
<thead>
<tr>
<th><code>nMode</code> Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The function should return the Odd video field results.</td>
</tr>
<tr>
<td>2</td>
<td>The function should return the Even video field results.</td>
</tr>
<tr>
<td>3</td>
<td>The function should return a combination of the two fields, i.e., the sum of the even and odd field area measurements and the maximum of the even and odd chord lengths are returned. This provides a result for the entire video frame.</td>
</tr>
</tbody>
</table>

The `pdResults` argument is a pointer to an array of doubles into which the video results are copied. The calling application must ensure the array contains at least four elements. The video results are returned in the array in this order:

<table>
<thead>
<tr>
<th><code>pdResults Array Index</code></th>
<th>Video Result Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Focus score (full video frame)</td>
</tr>
<tr>
<td>1</td>
<td>Detected area</td>
</tr>
<tr>
<td>2</td>
<td>Detected maximum chord length</td>
</tr>
<tr>
<td>3</td>
<td>Detected maximum gradient</td>
</tr>
</tbody>
</table>

The `pdPositions` argument is a pointer to an array of doubles into which the position data is copied. The calling application must ensure the array consists of at least four elements. The position data is returned in the array in this order:

<table>
<thead>
<tr>
<th><code>pdPositions Array Index</code></th>
<th>Axis Position Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X Axis position</td>
</tr>
<tr>
<td>1</td>
<td>Y Axis position</td>
</tr>
<tr>
<td>2</td>
<td>Z Axis position</td>
</tr>
<tr>
<td>3</td>
<td>F Axis position</td>
</tr>
</tbody>
</table>

### See Also

- `OI_SetVideoSettings`, `OI_ReadVideoData`, `OI_ReadVideoResults`, `OI_ReadVideoResultsEx`, `OI_ReadVideoResultsXY`, `OI_ReadVideoResultsZ`
OI_ReadVideoResultZ

Syntax

\[ \text{OI\_API OI\_ReadVideoResultZ( int nMode, double* pdResults, double* pdZPos )} \]

Description

Reads the focus Z position and the extended video measurement results for the odd video field, even video field, or a combination of both.

Parameters

- \( nMode \) Specifies the desired results, as described in the Comments below.
- \( pdResults \) An array of doubles into which the video results are to be copied.
- \( pdZPos \) The position of the Z Axis, in microns.

Return Value

- OI_OK if successful.
- OI_TIMEOUT if the desired results are not available within the video timeout period.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

Comments

The \( nMode \) parameter indicates which video field is to be read, using the following values:

<table>
<thead>
<tr>
<th>( nMode ) Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The function should return the Odd video field results.</td>
</tr>
<tr>
<td>2</td>
<td>The function should return the Even video field results.</td>
</tr>
<tr>
<td>3</td>
<td>The function should return a combination of the two fields, i.e., the sum of the even and odd field area measurements and the maximum of the even and odd chord lengths are returned. This provides a result for the entire video frame.</td>
</tr>
</tbody>
</table>

The \( pdResults \) argument is a pointer to an array of doubles. The calling application must ensure the array contains at least four elements. The video results are returned in the array in this order:

<table>
<thead>
<tr>
<th>( pdResults ) Array Index</th>
<th>Video Result Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Focus score (full video frame)</td>
</tr>
<tr>
<td>1</td>
<td>Detected area</td>
</tr>
<tr>
<td>2</td>
<td>Detected maximum chord length</td>
</tr>
<tr>
<td>3</td>
<td>Detected maximum gradient</td>
</tr>
</tbody>
</table>
**OI_SetVideoSettings**

**Syntax**

```c
OI_API OI_SetVideoSettings(int nEvenPhase, int nEvenThreshold, int nOddPhase, int nOddThreshold)
```

**Description**
Sets the video measurement parameters for the Even and Odd video fields.

**Parameters**

- `nEvenPhase`: Specifies the desired phase, either light or dark, for the Even video fields.
- `nEvenThreshold`: Specifies the desired threshold, from 0 to 255, for the Even video fields.
- `nOddPhase`: Specifies the desired phase, either light or dark, for the Odd video fields.
- `nOddThreshold`: Specifies the desired threshold, from 0 to 255, for the Odd video fields.

**Return Value**

- `OI_OK` if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

A full frame of video is composed of two video fields, one painting the odd lines, the other painting the even ones.

The OASIS-AF module assigns each video field its own Phase and Threshold. This provides for nearly simultaneous measurement of two distinct phases in the incoming video signal.

The Phase parameter (`nEvenPhase` and `nOddPhase`) may be either of two values:

<table>
<thead>
<tr>
<th>Phase Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Light objects are detected for measurement. Features in the video signal are detected from the brightest values (255) down to the value specified by the Threshold parameter.</td>
</tr>
<tr>
<td>1</td>
<td>Dark objects are detected for measurement. Features in the video signal are detected from the darkest values (0) up to the value specified by the Threshold parameter.</td>
</tr>
</tbody>
</table>

**See Also**

- `OI_ReadVideoData`, `OI_ReadVideoResults`
**OI_SetVideoWindow**

**Syntax**

```
OI_API OI_SetVideoWindow( int nXStart, int nXStop, int nYStart, int nYStop )
```

**Description**
Sets the positions defining the video window placement.

**Parameters**

- **nXStart**
  The start X value for the video window, in pixels.

- **nXStop**
  The stop X value for the video window, in pixels.

- **nYStart**
  The start Y value for the video window, in video field lines.

- **nYStop**
  The stop Y value for the video window, in video field lines.

**Return Value**

- **OI_OK** if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

The video window defines the region in which all video measurements are made. This includes focus score calculations as well as detected area, maximum chord length, and maximum gradient measurements.

The OASIS-AF hardware places some restrictions on the positions and sizes that are possible for these settings.

The **nXStart** and **nXStop** values must be set on 4-pixel boundaries, e.g., 4, 8, 12, etc.

The **nYStart** and **nYStop** values are specified in terms of pixels. For instance, a PAL video field has a vertical resolution of 768 pixels, and an NTSC video field has a vertical resolution of 480 pixels.

The **nYStart** video window Y start position is restricted to values from 0 to 254, i.e. roughly the top half of the video field.

The **nYStop** video window Y stop position may take on any values up to the size of the available video field, i.e. up to 767 for PAL video and up to 479 for NTSC video signals.

**See Also**

- `OI_SetVideoWindow`
- `OI_GetAFFitted`
- `OI_GetPCBStatus`
Filter Changer Functions

Version 2.0 of the OASIS DLL introduces a new filter changer component. This component configures the F-Axis (the 4th axis) of the controller for filter changer operation.

From Version 2.04 onwards, the optional 5th axis provided by the OASIS-XA1 module may be used as a secondary filter changer. If the OASIS-XA1 module is fitted, then filter commands such as `OI_MoveToFilter` are routed to either of the two available filter axes, F or T. The `OI_SelectFilterChanger` function is used to set the currently active filter changer.

A filter changer is essentially a number of defined positions corresponding to each filter in the filter changer. A popular configuration for a filter changer is a wheel consisting of a number of equally spaced filters. A given filter is selected by rotating through the appropriate angle between the current position and the new position. Also, the filter wheel usually has a home switch fitted, allowing the controller to automatically orient itself to a known position.

The OASIS DLL’s filter changer component manages the setup and operation of such filter changers, allowing you to specify the number of filters, automatically initialise the filter changer based on the home switch or limit limit switches (for linear rather than rotating filter changers), and move to filters according to filter indices rather than axes positions.

To define a filter changer:

1. Set the number filters in your filter changer, using a call to `OI_SetFilterCount`;
2. Set the offset giving the distance in microsteps between the filter home position and the first filter, using `OI_SetFilterHomeOffset`;
3. Initialise the filter changer, using `OI_InitFilterChanger`.

Once you have defined the filter changer, you may use the `OI_MoveToFilter` function to move to filter positions.

The OASIS DLL’s filter changer component also supports the Leica Microsysetemts’ DMR automated microscope fluorescence module. See the following table for the supported functions for the OASIS controller and the Leica Microsystems DMR microscope.

<table>
<thead>
<tr>
<th>Supported Functions</th>
<th>OASIS</th>
<th>Leica DMR</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>OI_SetFilterCount</code></td>
<td><code>OI_GetFilterCount</code></td>
<td><code>OI_GetFilterCount</code></td>
</tr>
<tr>
<td><code>OI_GetFilterCount</code></td>
<td><code>OI_DeleteFilter</code></td>
<td><code>OI_MoveToFilter</code></td>
</tr>
<tr>
<td><code>OI_MoveToFilter</code></td>
<td><code>OI_GetFilterPosition</code></td>
<td><code>OI_GetFilterPosition</code></td>
</tr>
<tr>
<td><code>OI_InitFilterChanger</code></td>
<td><code>OI_SetFilterChanger</code></td>
<td><code>OI_GetFilterName</code></td>
</tr>
<tr>
<td><code>OI_SetFilterHomeOffset</code></td>
<td><code>OI_GetFilterHomeOffset</code></td>
<td><code>OI_GetFilterDescription</code></td>
</tr>
<tr>
<td><code>OI_GetFilterHomeOffset</code></td>
<td><code>OI_WaitForStoppedFilter</code></td>
<td><code>OI_SetShutter</code></td>
</tr>
<tr>
<td><code>OI_GetFilterName</code></td>
<td></td>
<td><code>OI_GetShutter</code></td>
</tr>
<tr>
<td><code>OI_MoveToFilter</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>OI_GetFilterPosition</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>OI_SetFilterChanger</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>OI_GetFilterHomeOffset</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>OI_WaitForStoppedFilter</code></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## OI_ClearFilterHomeInfo

**Syntax**

OI_API OI_ClearFilterHomeInfo( void )

**Description**

Clears the home switch sensor values.

**Parameters**

None.

**Return Value**

OI_OK if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

The OI_ClearFilterHomeInfo function clears the internal flag indicating the home switch has been sensed.

Note the flag will be reset automatically whenever the controller senses the home switch signal.

**See Also**

OI_ReadFilterHomeInfo

---

## OI_DeleteFilter

**Syntax**

OI_API OI_DeleteFilter( int nPosition )

**Description**

Removes the specified filter from the list of filter positions.

**Parameters**

- **nPosition**: The one based index of the filter to remove.

**Return**

OI_OK if successful.
Value

If unsuccessful, a combination of error codes may be returned to indicate the
reason for failure.

Comments

The `OI_DeleteFilter` function provides a simple method to delete a given filter
from the list of currently defined filters. The indicated filter is removed, causing
any filters in the list after it to be moved up one index.

After a call to `OI_DeleteFilter`, the filter count will be reduced by one.

See Also `OI_SetJoystickEnabled`

---

### OI_GetFilterChanger

**Syntax**

```c
OI_API OI_GetFilterChanger( int* pnChanger )
```

**Description**

Retrieves the active filter changer.

**Parameters**

- `pnChanger` Pointer to receive the zero-based index of the
currently active filter changer.

**Return Value**

- `OI_OK` if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the
reason for failure.

**Comments**

If the OASIS-XA1 module is fitted, the OASIS system is considered to have 2
filter changers, one using the 4th axis and the other using the 5th axis. The
`OI_GetFilterChanger` functions returns which filter is active, i.e., which axis is
the current target for filter control commands.

See Also `OI_SetFilterChanger, OI_GetFilterChangerCount`

---

### OI_GetFilterChangerCount

**Syntax**

```c
OI_API OI_GetFilterChangerCount( int* pnChangers )
```

**Description**

Retrieves the number of filter changers available.

**Parameters**

- `pnChangers` Pointer to receive the value of the number of filter
changes fitted.

**Return Value**

- `OI_OK` if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the
reason for failure.

**Comments**

For the OASIS controller, the number of filter changes is dependant on whether
the OASIS-XA1 module is fitted. If fitted, this module allows an extra axis for filter change control, bringing the total to two. If not fitted, the OASIS card’s 4th axis is used as a single filter changer device.

See Also OI_GetFilterChanger, OI_SetFilterChanger

**OI_GetFilterCount**

**Syntax**

```
OI_API OI_GetFilterCount( int *pnFilters )
```

**Description**

Retrieves the current number of filter positions in the filter changer.

**Parameters**

- `pnFilters` The returned number of filters.

**Return Value**

- OI_OK if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

See the OI_SetFilterCount function for more information about the filter count.

See Also OI_SetFilterCount

**OI_GetFilterDescription**

**Syntax**

```
OI_API OI_GetFilterDescription( int nPosition, LPSTR szBuffer, int nBufferLen )
```

**Description**

Retrieves the current description for a given filter position.

**Parameters**

- `nPosition` The one-based filter position.
- `szBuffer` The destination text buffer.
- `nBufferLen` The length of the destination buffer specified in the `szBuffer` parameter.

**Return Value**

- OI_OK if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

See the OI_SetFilterDescription function for more information about filter descriptions.

See Also OI_SetFilterDescription, OI_SetFilterName, OI_GetFilterName
### OI_GetFilterHomeOffset

**Syntax**

```c
OI_API OI_GetFilterHomeOffset( double *pdOffset )
```

**Description**

Retrieves the current offset, in calibrated units, between the home position and the first filter.

**Parameters**

- `pdOffset` The returned joystick control status for the X axis.

**Return Value**

- **OI_OK** if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

See the `OI_SetFilterHomeOffset` function for more information about the filter home offset.

**See Also**

- `OI_SetFilterHomeOffset`

---

### OI_GetFilterName

**Syntax**

```c
OI_API OI_GetFilterName( int nPosition, LPSTR szBuffer, int nBufferLen )
```

**Description**

Retrieves the current name for a given filter position.

**Parameters**

- `nPosition` The one-based filter position.
- `szBuffer` The destination text buffer.
- `nBufferLen` The length of the destination buffer specified in the `szBuffer` parameter.

**Return Value**

- **OI_OK** if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

See the `OI_SetFilterName` function for more information about filter names and descriptions.

**See Also**

- `OI_SetFilterName`, `OI_SetFilterDescription`, `OI_GetFilterDescription`

---

### OI_GetFilterOffset

**Syntax**

```c
OI_API OI_GetFilterOffset( int nPosition, double *pdOffset )
```
Description
Retrieves the offset from the home position to a given filter's position.

Parameters
- nPosition
  The one-based filter position.
- dOffset
  The distance, in calibrated units, from the filter home position to the filter position.

Return Value
OI_OK if successful.
If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

Comments
The OI_GetFilterOffset function returns the offset, in calibrated units, from the home position to a given filter position.

Note that after the filter changer has been initialised using a call to OI_InitFilterChanger, the home position will be defined as a calibrated position of 0.

See Also
OI_SetFilterOffset, OI_InitFilterChanger, OI_SetFilterHomeOffset

---

**OI_GetFilterPosition**

Syntax
OI_API OI_GetFilterPosition( int *pnPosition )

Description
Retrieves the currently selected filter position index.

Parameters
- pnPosition
  The returned one-based filter position index.

Return Value
OI_OK if successful.
If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

Comments
See the OI_MoveToFilter function for more information about filter positions.

See Also
OI_MoveToFilter

---

**OI_GetFilterTimeout**

Syntax
OI_API OI_GetFilterTimeout(LPDWORD lpdwMSecs)

Description
Retrieves the current filter changer timeout duration.

Parameters
- lpdwMSecs
  Pointer to receive the current filter changer timeout, in milliseconds.
Return Value

OI_OK if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

Comments

When filter changer operations that use a wait loop take more time than the timeout duration, the function returns with an error which includes the OI_TIMEOUT bit being set.

The OI_GetFilterTimeout function allows you to find out how long these functions currently allow before a timeout occurs.

See Also OI_SetFilterTimeout

---

### OI_GetAvailableShutterCount

**Syntax**

```c
OI_API OI_GetAvailableShutterCount( int * pnTotalSupported)
```

**Description**

Retrieves the maximum number of available shutters for the current configuration.

**Parameters**

- `pnTotalSupported` The returned count of available shutters.

**Return Value**

OI_OK if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

The number of available shutters is based on the configuration. For instance, a single OASIS-blue controller supports 2 shutters, while an OASIS-4i in combination with the OI-SC4 shutter controller supports up to 4. This function returns the maximum number of shutters for the current configuration.

See Also OI_SetShutter, OI_GetShutter

---

### OI_GetShutter

**Syntax**

```c
OI_API OI_GetShutter( int *pnPosition, int nShutter )
```

**Description**

Retrieves the current status of a shutter.

**Parameters**

- `pnPosition` The returned position value.
- `nShutter` The shutter to be read.

**Return Value**

OI_OK if successful.

---
If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

See the `OI_SetShutter` function for more information about shutter control.

**See Also**

`OI_SetShutter`

---

### OI_GetShutterEx

**Syntax**

```c
OI_API OI_GetShutterEx(LPWORD pwRegister, LPSHUTTERINFO lpsiShutters, int* pnShutterCount )
```

**Description**

Retrieves the current status of a shutter.

**Parameters**

- `pwRegister`
  
  Returns the shutter control register value. Bit 3 will be set to 1 if the High Voltage for channel 1 is enabled.

- `lpsiShutters`
  
  Pointer to an array of SHUTTERINFO structs describing the shutter information.

- `pnShutterCount`
  
  Returns the number of available shutter channels.

**Compatibility**

Available only on **OASIS-blue** controller.

**Return Value**

- `OI_OK` if successful.

  If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

The `OI_GetShutterEx` function returns extended shutter control information for the OASIS-blue shutter controller channels.

The shutter control action consists of a high voltage pulse followed by a lower holding voltage that is sustained either indefinitely or for a timed duration.

The SHUTTERINFO structure returns information regarding the currently configured pulse duration as well as a readout of the current timer values for a timed shutter operation that may be currently underway. The following table provides the details for the SHUTTERINFO structure:

<table>
<thead>
<tr>
<th>SHUTTERINFO Member</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>wDurationHV</td>
<td>Indicates the High Voltage pulse duration (20-100 msec)</td>
</tr>
<tr>
<td>wTimerHV</td>
<td>Indicates the current High Voltage timer value (20-100 msec)</td>
</tr>
<tr>
<td>wDurationLV</td>
<td>Indicates the Low (holding) Voltage duration (0 to 65535 msec)</td>
</tr>
</tbody>
</table>
wTimerLV Indicates the current Low (holding) Voltage timer value (0 to 65535 msec)

Example:

// Simple example to see if any shutter channel is currently open
BOOL IsAnyShutterOpen()
{
    WORD wRegister;
    SHUTTERINFO si[2];
    int iShutters;

    // Read shutter status
    OI_GetShutterEx( &wRegister, si, &iShutters );

    // if either of the low voltage timers are > 0, a shutter is open
    return ( (si[0].wTimerLV>0) || (si[1].wTimerLV>0) );
}

See the OI_SetShutterEx function for more information about shutter control using the OASIS-blue controller.

See Also

OI_SetShutterEx

OI_API OI_GetShutterEx( LPWORD pwRegister, LPSHUTTERINFO lpsiShutters, int* pnShutterCount );

**OI_GetShutterMulti**

**Syntax**

OI_API OI_GetShutter( int *pnStates )

**Description**

Retrieves the current status of a shutter.

**Parameters**

- **pnStates**
  
  The returned position values.

**Return Value**

- OI_OK if successful.

- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

The OI_GetShutter function returns the status of all shutters in the pnStates parameter. Bit 0 will be set if shutter #1 is open, Bit 1 will be set if shutter #2 is open, and so on.

You may use the SHUTTER defines (i.e., SHUTTER1, SHUTTER2, SHUTTER3, SHUTTER4) to test each bit using a bitwise AND with the returned
See Also  
OI_SetShutterMulti

**OI_InitFilterChanger**

Syntax  
$$\text{OI_API OI_InitFilterChanger( int nMethod, double dUnitsPerRev )}$$

Description  
Initialises the filter wheel either by a home-switch method, a limit switch method, or a manual method.

Parameters  
- *nMethod*  
The method to use for initialisation.
- *dUnitsPerRev*  
The number of calibrated units per full filter changer revolution or travel. Note this value may be different from the pitch of the axis.

Return Value  
- OI_OK if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

Comments  
The filter changer functions require three values in order to function. These values are:

1. The number of discreet filter positions;
2. The number of steps per full filter changer revolution;
3. The offset to the first filter.

The first value is set using the **OI_SetFilterCount** function, and the third value is set using the **OI_SetFilterHomeOffset** function. The **OI_InitFilterChanger** is used to set the second value, i.e., the number of microsteps per full filter changer revolution.

This can be accomplished by one of three methods, as specified by the *nMethod* parameter.

<table>
<thead>
<tr>
<th><em>nMethod Value</em></th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>OI_FILTER_INIT_HOME</td>
<td>The filter changer device has a home switch. This is typical for rotating filter wheels. This method causes the filter wheel to be rotated through 2 home switch cycles in order to measure the microsteps per revolution value.</td>
</tr>
</tbody>
</table>
**OI_FILTER_INIT_LIMITS**  The filter changer device has limit switches. This method supports a linear filter changer with limit switches at the negative and positive limits of travel. This method causes the filter changer to travel to each end to find the limit switches and calculates the distance between the limits as the valid range of travel.

**OI_FILTER_INIT_USER**  This method allows the value of the steps per revolution to be specified by the *dUnitsPerRev* parameter.

Once the steps per revolution value is obtained, the **OI_InitFilterChanger** function automatically recalculates the offsets to each filter position based on the current values for the home switch offset to the first filter and the number of filter positions.

See Also  **OI_SetFilterHomeOffset**, **OI_SetFilterCount**

---

### OI_MoveToFilter

**Syntax**  

```
OI_API OI_MoveToFilter( int nPosition, int nWait )
```

**Description**  Moves the filter changer to the indicated filter position.

**Parameters**
- **nPosition**  The one-based index of the filter to move to.
- **nWait**  A non-zero *nWait* parameter indicates the function should wait until the position is reached prior to returning.

**Return Value**  
- **OI_OK** if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**  The **OI_MoveToFilter** command causes the filter control axis to move to the location corresponding to the indicated filter position index.

See Also  **OI_SetFilterCount**, **OI_InitFilterChanger**, **OI_SetFilterHomeOffset**

---

### OI_ReadFilterChangerInfo

**Syntax**  

```
OI_API OI_ReadFilterChangerInfo( double *pdUnitsPerRev, double *pdFilterSpacing )
```
### OI_ReadFilterChangerInfo

**Syntax**

```c
OI_API OI_ReadFilterChangerInfo( LPBOOL lpbHomeFound, double *pdHomeLeft, double *pdHomeRight )
```

**Description**
Retrieves the current information regarding the home switch detection.

**Parameters**
- `lpbHomeFound` Flag indicating if a home switch has been detected.
- `pdHomeLeft` The leading edge position of the home switch, in calibrated units.
- `pdHomeRight` The trailing edge position of the home switch, in calibrated units.

**Return Value**
- `OI_OK` if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**
The OASIS controller hardware automatically detects whenever a home switch has been detected. This information is used, for instance, during initialisation in order to determine the calibrated distance per filter wheel revolution by measuring the distance between successive home filter detections.

The **OI_ReadFilterHomeInfo** allows you to determine if a home filter has been detected.
detected as well as the positions of the leading and trailing edges of the filter switch connection.

See Also  OI_InitFilterChanger

---

### OI_SelectFilterChanger

**Syntax**

```
OI_API OI_SelectFilterChanger( int nChanger )
```

**Description**

Selects the active filter changer.

**Parameters**

- `nChanger`: The zero-based index of the filter changer to be made active.

**Return Value**

- `OI_OK` if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

When the optional OASIS-XA1 module is fitted, the OASIS system can provide two filter changers, one on the F-axis and the other on the T-axis. Filter commands are routed to either axis based on the currently active filter changer, as selected by `OI_SelectFilterChanger`.

See Also  OI_GetFilterChanger, OI_GetFilterChangerCount

---

### OI_SetFilterCount

**Syntax**

```
OI_API OI_SetFilterCount( int nFilters )
```

**Description**

Sets the number of filter positions in the filter changer.

**Parameters**

- `nFilters`: The number of filters.

**Return Value**

- `OI_OK` if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

A filter changer like a filter wheel will have a fixed number of positions, one for each filter. The `OI_SetFilterCount` function is used to set this value.

A call to `OI_SetFilterCount` will cause a re-calculation of each filter’s spacing, based on the number of microsteps required for one full revolution of the filter changer and the number of filters positions.

Note the maximum number of filters in a filter changer is 32 positions.
**See Also**   OI_GetFilterCount

---

**OI_SetFilterDescription**

**Syntax**   

```c
OI_API OI_SetFilterDescription( int nPosition, LPSTR szBuffer )
```

**Description**   

Retrieves the status of joystick control for the X, Y, and Z axes.

**Parameters**

- `nPosition`   The one-based filter position.
- `szBuffer`   The text buffer containing the name to set.

**Return Value**   

- `OI_OK` if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**   

Each filter position may be assigned a name and description. The `OI_SetFilterDescription` function allows you to specify an extended (up to 64 characters, including the terminating null character) string describing the filter position.

**See Also**   

- OI_GetFilterDescription, OI_SetFilterName, OI_GetFilterName

---

**OI_SetFilterHomeOffset**

**Syntax**   

```c
OI_API OI_SetFilterHomeOffset( double dOffset )
```

**Description**   

Sets the offset, in calibrated units, between the home positions and the first filter position.

**Parameters**

- `dOffset`   The distance, in calibrated units, from the home switch position to the first filter position.

**Return Value**   

- `OI_OK` if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**   

To automatically set the offsets to each filter position after a home-based initialisation, the offset from the home switch to the first filter is required.

Calling the `OI_SetFilterHomeOffset` causes a recalculation of the offsets to each filter.

See the `OI_InitFilterChanger` function for more information about the use of the home offset.
Oi_SetFilterLocation

Syntax

Oi_API OI_SetFilterLocation( int nPosition )

Description
Sets the specified filter to the current offset.

Parameters

nPosition
The filter position to be set.

Return Value

OI_OK if successful.
If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

Comments

The Oi_SetFilterLocation function may be used to explicitly set a given filter’s position to the current location of the filter wheel. For instance, you may adjust the filter changer’s control axis to put the desired filter into place, then call the Oi_SetFilterLocation function to “teach” the filter changer the offset to that filter.

See Also
Oi_InitFilterChanger, Oi_SetFilterOffset, Oi_GetFilterOffset

Oi_SetFilterName

Syntax

Oi_API OI_SetFilterName( int nPosition, LPSTR szBuffer )

Description
Sets the name of a given filter position as a short text string.

Parameters

nPosition
The one-based filter position.

szBuffer
The text buffer containing the name to set.

Return Value

OI_OK if successful.
If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

Comments

Each filter position may be assigned a name and description. The Oi_SetFilterName function allows you to specify a short (up to 16 characters, including the terminating null character) string naming the filter position.

See Also
Oi_GetFilterName, Oi_SetFilterDescription, Oi_GetFilterDescription
**OI_SetFilterOffset**

**Syntax**

OI_API OI_SetFilterOffset( int nPosition, double dOffset )

**Description**

Sets the offset from the home position to a given filter's position.

**Parameters**

- `nPosition` The one-based filter position.
- `dOffset` The distance, in calibrated units, from the filter home position to the filter position.

**Return Value**

OI_OK if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

Although normally the OI_InitFilterChanger and OI_SetFilterHomeOffset function is used to automatically set the offsets to each filter position, you may also use OI_SetFilterOffset to set or adjust these positions yourself.

**See Also**

OI_GetFilterOffset, OI_InitFilterChanger, OI_SetFilterHomeOffset

---

**OI_SetFilterTimeout**

**Syntax**

OI_API OI_SetFilterTimeout(DWORD dwMSecs)

**Description**

Sets the timeout value for filter change operations.

**Parameters**

- `dwMSecs` The duration for the timeout, in milliseconds.

**Return Value**

OI_OK if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

When filter changer operations that use a wait loop take more time than the timeout duration, the function returns with an error, which includes the OI_TIMEOUT, bit being set.

The OI_SetFilterTimeout function allows you to specify how long these functions should allow before a timeout occurs.

**See Also**

OI_GetFilterTimeout
**OI_SetShutter**

**Syntax**

```c
OI_API OI_SetShutter( int nPosition, int nShutter )
```

**Description**

Sets the state of a shutter.

**Parameters**

- `nPosition` The desired shutter position, as described in the comments.
- `nShutter` The shutter to be set.

**Return Value**

- `OI_OK` if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

A shutter may be used in conjunction with a filter changer in order to control when light is allowed to pass to the specimen. This is particularly important in some fluorescence applications where the specimen must be protected from excessive light exposure to prevent fading of signal.

The OI-SC4 shutter controller supports up to 4 shutters. The `nShutter` parameter specifies the one-based index of the shutter to be set. For instance, an `nShutter` value of 1 indicates that shutter #1 is to be set to the stage specified by `nPosition`.

The shutter position is specified according to the following values.

<table>
<thead>
<tr>
<th><code>nPosition</code> Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Opens the shutter, allowing light to pass.</td>
</tr>
<tr>
<td>0</td>
<td>Closes the shutter.</td>
</tr>
</tbody>
</table>

**See Also**

- `OI_SetShutterMulti`, `OI_GetShutter`, `OI_GetShutterMulti`

**OI_SetShutterEx**

**Syntax**

```c
OI_API OI_SetShutterEx( LPBOOL pbShutters, LPWORD pwValues)
```

**Description**

Sets the state of a shutter.

**Parameters**

- `pbShutters` Pointer to an array of shutters to be set.
- `pwValues` Pointer to an array of shutter values.

**Return Value**

- `OI_OK` if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Compatibility**

- Available only on OASIS-blue controller.
Comments

The `OI_SetShutterEx` provides extended shutter control of the on-board dual shutter control channels of the OASIS-blue controller.

The `pbShutters` parameter specifies whether channels 0 and 1 are to be affected by the `pwValues` parameter.

Example:

```c
void MyShutterFunc()
{
    BOOL bShutters[2];
    WORD wValues[2];

    // indicate that we want to set both channels
    bShutters[0] = TRUE;
    bShutters[1] = TRUE;

    // shutter channel one will open for 300 msec
    wValues[0] = 300;

    // shutter channel two will open indefinitely
    wValues[1] = OI_SHUTTER_OPEN;
}
```

The shutter control values are described in the following table:

<table>
<thead>
<tr>
<th>Shutter Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 or OI_SHUTTER_CLOSE</td>
<td>Closes the shutter.</td>
</tr>
<tr>
<td>1 to 65534</td>
<td>Open for specified time, in milliseconds</td>
</tr>
<tr>
<td>65535 or OI_SHUTTER_OPEN</td>
<td>Opens the shutter, leaving open indefinitely, until closed by a subsequent command.</td>
</tr>
</tbody>
</table>

See Also `OI_SetShutterMulti`, `OI_GetShutter`, `OI_GetShutterMulti`

---

**OI_SetShutterMulti**

**Syntax**

```c
OI_API OI_SetShutterMulti( int nStates, int nStateMask )
```

**Description**

Sets the state of a shutter.

**Parameters**

- `nStates`: A combination desired shutter positions, as described in the comments.

- `nStateMask`: A mask defining which bits of the `nStates` parameter are valid.
Return Value

OI_OK if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

Comments

The OI-SC4 shutter controller supports up to 4 shutters. The nStates parameter specifies the desired position of each shutter, using a bit-field. For instance, Bit 0 is used to set shutter #1, Bit 1 is used to set shutter #2 and so on.

You may use the SHUTTER #define’s to indicate the individual shutters, e.g., SHUTTER1, SHUTTER2, SHUTTER3, SHUTTER4.

The nStateMask is used to indicate which portions of the nStates bitfield are valid. For instance, setting nStateMask to a value of (SHUTTER1 | SHUTTER3) indicates that only shutters 1 and 3 are to be affected by the corresponding bits in the nStates bitfield.

The shutter positions in the nStates bitfield are specified according to the following values.

<table>
<thead>
<tr>
<th>nPosition Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Opens the shutter, allowing light to pass.</td>
</tr>
<tr>
<td>0</td>
<td>Closes the shutter.</td>
</tr>
</tbody>
</table>

See Also OI_SetShutter, OI_GetShutter, OI_GetShutterMulti

---

**OI_WaitForStoppedFilter**

**Syntax**

OI_API OI_WaitForStoppedFilter(void)

**Description**

Waits until the filter changer control axis stops moving.

**Parameters**

None.

**Return Value**

OI_OK if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

The OI_WaitForStoppedFilter function may be used to ensure the filter has reached a given position after an OI_MoveToFilter function call. Note that you may also specify a wait when calling OI_MoveToFilter.

**See Also**

OI_MoveToFilter
Hardware Joystick and Trackball Functions

Manual control of automated axes may be achieved using either an analog joystick or a trackball (or other serial device). Several functions address enabling/disabling the use of the devices via software control, as well as interrogating the status of trackball button presses.

**OI_ClearTrackballStatus**

**Syntax**

OI_API OI_ClearTrackballStatus( WORD wMask )

**Description**

Clears trackball button pressed and released flags.

**Parameters**

- *wMask*  
  A mask indicating which flags are to be cleared, as described in the comments section below.

**Return Value**

- OI_OK if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

Whenever a trackball button pressed or released flag has been set by the controller, it will remain set until cleared by the `OI_ClearTrackballStatus` function.

The flags to be cleared are controlled by the `wMask` parameter, which should be a bitwise OR of the pressed and released flag values as defined in the comments of the `OI_ReadTrackballStatus` function.

For instance, to clear only the button 1 pressed and released flags, use the following:

```
OI_ClearTrackballStatus( OI_BUTTON1_PRESSED
| OI_BUTTON1_RELEASED );
```

`OI_ClearTrackballStatus` may be used to clear all button flags by passing a value of `OI_CLEAR_ALL_BUTTONS` as the `wMask` parameter.

**See Also**

- OI_SetJoystickEnabled

**OI_GetJoystickEnabled**

**Syntax**

OI_API OI_GetJoystickEnabled(BOOL* pbXEnabled, BOOL* pbYEnabled, BOOL* pbZEnabled)

...
Description
Retrieves the status of joystick control for the X, Y, and Z axes.

Parameters
- **pbXEnabled**
  The returned joystick control status for the X axis.
- **pbYEnabled**
  The returned joystick control status for the Y axis.
- **pbZEnabled**
  The returned joystick control status for the Z axis.

Return Value
- **OI_OK** if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

Comments
See the **OI_SetJoystickEnabled** function for more information about hardware joystick control.

See Also
**OI_SetJoystickEnabled**

---

**OI_GetTrackballControl**

Syntax
```c
OI_API OI_GetTrackballControl( LPWORD pwEnable )
```

Description
Retrieves the current settings for default trackball button action enabling.

Parameters
- **pwEnable**
  The returned trackball button control WORD.

Return Value
- **OI_OK** if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

Comments
The **OI_GetTrackballControl** function retrieves the currently defined trackball button actions. The **pwEnable** word will be a bitwise OR of the button flags indicating which buttons are to perform the default action, as defined in the **OI_SetTrackballControl** function description.

See Also
**OI_SetTrackballControl**

---

**OI_GetTrackballEnabled**

Syntax
```c
OI_API OI_GetTrackballControl( BOOL* pbEnabled )
```

Description
Retrieves the current trackball enabled status.

Parameters
- **pbEnabled**
  The returned trackball control status.
Return Value
OI_OK if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

Comments
The OI_GetTrackballEnabled function retrieves the status of trackball control inputs. The pbEnabled word will be TRUE if trackball inputs are processed, FALSE if the trackball inputs are disabled.

See Also
OI_SetTrackballEnabled

---

**OI_ReadTrackballStatus**

Syntax
OI_API OI_ReadTrackballStatus( LPWORD pwStatus )

Description
Retrieves the status of the trackball input.

Parameters
pwStatus
The return status indicating the pressed and released flags for each button.

Return Value
OI_OK if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

Comments
The OASIS controller maintains a “pressed” and “released” flag for each trackball button. These flags are set whenever the corresponding trackball action takes place.

For instance, if the button has not been pressed, then both the pressed and released flags will be 0. Once the button is pressed, but before it is released, the pressed flag will be 1 while the released flag remains 0. Once the button is released, both pressed and released flags will be 1.

The pressed and released flags remain set until they cleared using OI_ClearTrackballStatus.

The test the returned status WORD, use the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>OI_BUTTON1_PRESSED</td>
<td>Button 1 (typically top-left) has been pressed.</td>
</tr>
<tr>
<td>OI_BUTTON1_RELEASED</td>
<td>Button 1 has been released.</td>
</tr>
<tr>
<td>OI_BUTTON2_PRESSED</td>
<td>Button 2 (typically top-right) has been pressed.</td>
</tr>
<tr>
<td>OI_BUTTON2_RELEASED</td>
<td>Button 2 has been released.</td>
</tr>
</tbody>
</table>
OI_BUTTON3_PRESSED Button 3 (typically bottom-left) has been pressed.

OI_BUTTON3_RELEASED Button 3 has been released.

OI_BUTTON4_PRESSED Button 4 (typically bottom-right) has been pressed.

OI_BUTTON4_RELEASED Button 4 has been released.

The following code gives an example of testing for a button 3 press:

```c
void CheckForButton3()
{
    WORD wStatus;

    // read the current status
    OI_ReadTrackballStatus( &wStatus );

    // check to see if the button 3 pressed flag is on
    if( wStatus & OI_BUTTON3_PRESSED )
    {
        // do your button 3 action here…

        // clear the button 3 flag
        OI_ClearTrackballStatus( OI_BUTTON3_PRESSED );
    }
}
```

See Also OI_ClearTrackballStatus, OI_SetTrackballControl

---

**OI_SetJoystickEnabled**

**Syntax**

```c
OI_API OI_SetJoystickEnabled(BOOL bXEnabled, BOOL bYEnabled, BOOL bZEnabled)
```

**Description**

Specifies which axes are enabled for joystick control.

**Parameters**

- `bXEnabled` Enables or disables joystick control of the X axis.
- `bYEnabled` Enables or disables joystick control of the Y axis.
- `bZEnabled` Enables or disables joystick control of the Z axis.
Return Value
OI_OK if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

Comments
A hardware joystick may be fitted allowing X, Y, and/or Z control inputs. The OI_SetJoystickEnabled function is used to set which axes are enabled for joystick control.

See Also
OI_GetJoystickEnabled

### OI_SetTrackballControl

**Syntax**

OI_API OI_SetTrackballControl( WORD wEnable )

**Description**

Enables default handling of trackball button presses.

**Parameters**

- **wEnable**
  - Enabling WORD for trackball buttons.

**Return Value**

- OI_OK if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

By default, the controller configures the trackball buttons for the following functions:

<table>
<thead>
<tr>
<th>Button</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Top-left (button 1)</td>
<td>Enable / disable trackball control of XY stage</td>
</tr>
<tr>
<td>Top-right (button 2)</td>
<td>Enable / disable trackball control of Z focus</td>
</tr>
<tr>
<td>Bottom-left (button 3)</td>
<td>Perform autofocus using current settings</td>
</tr>
<tr>
<td>Bottom-right (button 4)</td>
<td>Cycle through 3 pre-defined autofocus range and speed settings. (These settings are configured in the OASIS flash memory.)</td>
</tr>
</tbody>
</table>

An application may wish to take over the use of some or all of the buttons for custom tasks. The OI_SetTrackballControl function enables / disables the default trackball actions. A mask WORD given by the wMask parameter indicates which buttons should perform the default processing. The mask is assembled using a bitwise OR from the following button identifiers:

<table>
<thead>
<tr>
<th>Value</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>OI_BUTTON1</td>
<td>Button 1, typically top-left</td>
</tr>
</tbody>
</table>
OI_BUTTON2 Button 2, typically top-right
OI_BUTTON3 Button 3, typically bottom-left
OI_BUTTON4 Button 4, typically bottom-right

For instance, an application may wish to keep the top two buttons for the default actions of XY stage and Z focus movement enabling, while using the bottom two buttons for custom actions. In this case only buttons 1 and 2 should perform the default actions, so an application would set the trackball control to the following:

\[
\text{OI\_SetTrackballControl( OI\_BUTTON1 | OI\_BUTTON2 );}
\]

This indicates that the buttons 3 and 4 should not be processed for default actions (usually autofocus). The application may then trap the button pressed and/or released flags using the \text{OI\_ReadTrackballStatus} function.

See Also \text{OI\_GetTrackballControl}, \text{OI\_ReadTrackballStatus}

---

**\text{OI\_SetTrackballEnabled}**

**Syntax**

\[
\text{OI\_API OI\_SetTrackballEnabled( BOOL bEnabled )}
\]

**Description**

Enables and disables the function of the trackball.

**Parameters**

\[
bEnabled \quad \text{Enables or disables the trackball.}
\]

**Return Value**

\[
\text{OI\_OK if successful.}
\]

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

A trackball may be fitted allowing X, Y, and/or Z control inputs. The \text{OI\_SetTrackballEnabled} function is used to enable and disable all trackball control functions.

See Also \text{OI\_GetTrackballEnabled}

---

**Timeouts**

Several functions use loops internally to wait for acknowledgement by the OASIS hardware or to poll the system waiting for specific conditions, such as the stage to stop moving, a limit
switch to be found, or an automatic focus operation to complete. Examples of functions where this type of polling is used are:

- `OI_WaitForStoppedXYZ`
- `OI_WaitForStoppedF`
- `OI_WaitForAutoFocus`
- `OI_ReadFocusScore`
- `OI_ReadVideoResults`

In some instances, these operations may take longer than expected to finish, and in extreme cases, the automation system may be waiting for a physical situation to occur that may not be possible. For instance, an attempt to read video results from the OASIS-AF module when video is not available causes the system to poll the video input searching for video synchronization.

To protect against these operations leading to infinite polling loops, the OASIS DLL uses timeout. In essence, the logic of the polling functions is to check for the desired conditions for as long as the timeout period specifies. If the desired condition is not reached in the specified timeout period, the function returns with an `OI_TIMEOUT` error code.

### OI_GetAutoFocusTimeout

**Syntax**

```
OI_API OI_GetAutoFocusTimeout(LPDWORD lpdwMSecs)
```

**Description**

Retrieves the current timeout value for automatic focus operation.

**Parameters**

- `lpdwMSecs`: The returned AutoFocus timeout, in milliseconds.

**Return Value**

- `OI_OK` if successful.

  If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

The AutoFocus timeout is used in the `OI_WaitForAutoFocus` function.

**See Also**

- `OI_WaitForAutoFocus`

### OI_GetMoveTimeout

**Syntax**

```
OI_API OI_GetMoveTimeout(LPDWORD lpdwMSecs)
```

**Description**

Retrieves the current timeout value for X,Y,Z, and F axis movements.

**Parameters**

- `lpdwMSecs`: The returned move timeout, in milliseconds.
**Return Value**

OI_OK if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**
The move timeout is used in the OI_WaitForStoppedXYZ and OI_WaitForStoppedF functions.

**See Also**
OI_WaitForStoppedXYZ, OI_WaitForStoppedF

---

**OI_GetVideoTimeout**

**Syntax**

OI_API OI_GetVideoTimeout(LPDWORD lpdwMSecs)

**Description**

Retrieves the current timeout value used in video measurement functions such as reading the focus score and detected area and maximum chord length.

**Parameters**

*lpdwMSecs*  
The returned video timeout, in milliseconds.

**Return Value**

OI_OK if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**
The video timeout is used in the OI_ReadFocusScore and OI_ReadVideoResults functions.

**See Also**
OI_ReadFocusScore, OI_ReadVideoResults

---

**OI_SetAutoFocusTimeout**

**Syntax**

OI_API OI_SetAutoFocusTimeout(DWORD dwMSecs)

**Description**

Sets the timeout value for automatic focus operation.

**Parameters**

*dwMSecs*  
The desired AutoFocus timeout, in milliseconds.

**Return Value**

OI_OK if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**
The AutoFocus timeout is used in the OI_WaitForAutoFocus function.

**See Also**
OI_WaitForAutoFocus
OI_SetMoveTimeout

Syntax
OI_API OI_SetMoveTimeout(DWORD dwMSecs)

Description
Retrieves the current timeout value for X,Y,Z, and F axis movements.

Parameters
dwMSecs
The desired move timeout, in milliseconds.

Return Value
OI_OK if successful.
If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

Comments
The move timeout is used in the OI_WaitForStoppedXYZ and OI_WaitForStoppedF functions.

See Also
OI_WaitForStoppedXYZ, OI_WaitForStoppedF

OI_SetVideoTimeout

Syntax
OI_API OI_SetVideoTimeout(DWORD dwMSecs)

Description
Retrieves the current timeout value used in video measurement functions such as reading the focus score and detected area and maximum chord length.

Parameters
dwMSecs
The desired video timeout, in milliseconds.

Return Value
OI_OK if successful.
If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

Comments
The video timeout is used in the OI_ReadFocusScore and OI_ReadVideoResults functions.

See Also
OI_ReadFocusScore, OI_ReadVideoResults
File I/O and Settings

**OI_LoadPositions**

**Syntax**

```c
OI_API OI_LoadPositions (LPCTSTR sFile)
```

**Description**

Loads the OASIS position and limit data from the specified file.

**Parameters**

- `sFile`    The name of the file from which the positions are to be loaded.

**Return Value**

- `OI_OK` if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**

The `OI_LoadPositions` function restores the OASIS position and limit information for each axis from a text file. The file is structured like a standard Windows INI file.

If you pass an empty string in the `sFile` parameter, the settings will be loaded from the Windows Registry.

The current position of each axis is defined to be the value read from the file. For instance, if an X position value of 1234.5 is read from the specified file, then after the call to `OI_LoadPositions`, the current X axis position will be set to 1234.5.

The positions of the Negative and Positive User Limits are also read and set from the values found in the file.

To save the positions, use the `OI_SavePositions` function.

**See Also**

`OI_SavePositions`, `OI_LoadSettings`, `OI_SaveSettings`

---

**OI_LoadSettings**

**Syntax**

```c
OI_API OI_LoadSettings(LPCTSTR sFile)
```

**Description**

Loads the OASIS settings from the specified file.

**Parameters**

- `sFile`    The name of the file from which the settings are to be loaded.

**Return Value**

- `OI_OK` if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.
The OI_LoadSettings function restores the OASIS settings from a text file. The file is structured like a standard Windows INI file.

If you pass an empty string in the sFile parameter, the settings will be loaded from the Windows Registry.

To save the settings, use the OI_SaveSettings function.

See Also  OI_SaveSettings, OI_LoadPositions, OI_SavePositions

---

### OI_LoadSettingsEx

**Syntax**  

OI_API OI_LoadSettingsEx(LPCTSTR sFile, int nComponent )

**Description**  

Loads the OASIS settings for a component from the specified file.

**Parameters**  

- **sFile**  
  The name of the file from which the settings are to be loaded.

- **nComponent**  
  The component for which you want to load the settings.

**Return Value**  

OI_OK if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

**Comments**  

The OI_LoadSettingsEx function restores the OASIS settings from a text file for a specific component. The file is structured like a standard Windows INI file.

If you pass an empty string in the sFile parameter, the settings will be loaded from the Windows Registry.

To save the settings, use the OI_SaveSettings function.

See Also  OI_SaveSettings, OI_LoadPositions, OI_SavePositions

---

### OI_SavePositions

**Syntax**  

OI_API OI_SavePositions (LPCTSTR sFile)

**Description**  

Saves the current OASIS position and limit data for each axis to the specified file.

**Parameters**  

- **sFile**  
  The name of the file from which the positions are to be loaded.
Return Value
OI_OK if successful.
If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

Comments
The OI_SavePositions function stores the OASIS position and limit information for each axis to a text file. The file is structured like a standard Windows INI file.

If you pass an empty string in the sFile parameter, the settings will be stored to the Windows Registry.

The positions of the Negative and Positive User Limits are also stored in the file.

The stored position information can be restored using the OI_LoadPositions function, we redefines the position and limit values for each axis based on the information found in the file.

See Also OI_LoadPositions, OI_SaveSettings, OI_LoadSettings

OI_SaveSettings

Syntax
OI_API OI_SaveSettings(LPCTSTR sFile)

Description
Save the OASIS settings to the specified file.

Parameters
sFile The name of the file to which the settings are to be stored.

Return Value
OI_OK if successful.
If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

Comments
The OI_SaveSettings function stores the OASIS settings to a text file. The file is structured like a standard Windows INI file.

If you pass an empty string in the sFile parameter, the settings will be stored to the Windows Registry.

To load the settings saved by OI_SaveSettings, use the OI_LoadSettings function.

See Also OI_LoadSettings, OI_SavePositions, OI_LoadPositions

OI_SaveSettingsEx

Syntax
OI_API OI_SaveSettingsEx( LPCTSTR sFile, int nComponent )
Description

Save the OASIS settings for a component to the specified file.

Parameters

- *sFile*: The name of the file to which the settings are to be stored.

- *nComponent*: The component for which you want to save settings.

Return Value

- **OI_OK** if successful.

If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

Comments

The **OI_SaveSettingsEx** function stores the OASIS settings to a text file for a given component. The file is structured like a standard Windows INI file.

If you pass an empty string in the *sFile* parameter, the settings will be stored to the Windows Registry.

To load the settings saved by **OI_SaveSettingsEx**, use the **OI_LoadSettingsEx** function.

See Also

- **OI_LoadSettings**, **OI_SavePositions**, **OI_LoadPositions**

---

## Error Handling

**OI_GetLastError**

Syntax

- **OI_API OI_GetLastError( int* pnCmd, int* pnRetCode, char* szDesc, int nLen )**

Description

Retrieves information about the last error.

Parameters

- *pnCmd*: Returns the command code associated with the error.

- *pnRetCode*: Returns the return value from the function where the error occurred.

- *szDesc*: Returns an extended description of the error.

- *nLen*: Specifies the maximum length of the *szDesc* character array.

Return Value

- This function always returns **OI_OK**.
### Comments

All functions in the OASIS DLL use the return value to report basic error information. The `OI_GetLastError` function allows you to retrieve more detailed information regarding the error. The `pnRetCode` argument is the same as the error value returned from the function where the error occurred.

Note that error information is not cleared whenever a subsequent successful function is called.

The `pnCmd` argument may be useful when seeking support from Objective Imaging regarding the nature of the error. This code corresponds to the low level command associated with the error. This code may also be 0, indicating that a low level command was not involved in the error.

### See Also

- `OI_EnableMsgReportDlg`

---

### `OI_EnableMsgReportDlg`

<table>
<thead>
<tr>
<th>Syntax</th>
<th><code>OI_API OI_EnableMsgReportDlg( BOOL bEnabled )</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Enables or disables message box reporting of general exception handling.</td>
</tr>
<tr>
<td>Parameters</td>
<td><code>bEnabled</code> Flag enabling or disabling general exception messages.</td>
</tr>
<tr>
<td>Return Value</td>
<td>This function always returns <code>OI_OK</code>.</td>
</tr>
<tr>
<td>Comments</td>
<td>Whenever a general exception occurs and is handled in an OASIS DLL function, a general error message box is normally displayed. The <code>OI_EnableMsgReportDlg</code> function is used to programatically enable or disable the display of these message dialogs.</td>
</tr>
<tr>
<td>See Also</td>
<td><code>OI_GetLastError</code></td>
</tr>
</tbody>
</table>

---
Microns / Step Conversion

OI_MicronsToAbsoluteX

OI_MicronsToAbsoluteY

OI_MicronsToAbsoluteZ

OI_MicronsToAbsoluteF

Syntax

- OI_API OI_MicronsToAbsoluteX(double dMicronVal, LPDWORD lpdwSteps)
- OI_API OI_MicronsToAbsoluteY(double dMicronVal, LPDWORD lpdwSteps)
- OI_API OI_MicronsToAbsoluteZ(double dMicronVal, LPDWORD lpdwSteps)
- OI_API OI_MicronsToAbsoluteF(double dMicronVal, LPDWORD lpdwSteps)

Description

Converts a given axis position in microns into the absolute internal microstep counter value.

Parameters

- dMicronVal
  - The position value, in microns, to be converted.
- lpdwSteps
  - The returned absolute coordinate, in microsteps.

Return Value

- OI_OK if successful.
- If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

Comments

The current calibrated microstep size is used for these conversions. This calibration depends on an accurate setting of the microstep size using either the OI_SetAxisStepSize function or the OI_SetPitchXY function (for motorised stages).

See Also

OI_SetAxisStepSize, OI_SetPitchXY, OI_MicronsToStepsX, OI_MicronsToStepsY, OI_MicronsToStepsZ, OI_MicronsToStepsZ
### OI_MicronsToStepsX

### OI_MicronsToStepsY

### OI_MicronsToStepsZ

### OI_MicronsToStepsF

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
<th>Parameters</th>
<th>Return Value</th>
<th>Comments</th>
<th>See Also</th>
</tr>
</thead>
<tbody>
<tr>
<td>OI_API OI_MicronsToStepsX(double dMicronVal, long* lpdwSteps)</td>
<td>Converts a given micron distance into the corresponding microstep distance.</td>
<td>dMicronVal</td>
<td>OI_OK if successful.</td>
<td>The current calibrated microstep size is used for these conversions. This calibration depends on an accurate setting of the microstep size using either the OI_SetAxisStepSize function or the OI_SetPitchXY function (for motorised stages).</td>
<td>OI_SetAxisStepSize, OI_SetPitchXY, OI_MicronsToAbsoluteX, OI_MicronsToAbsoluteY, OI_MicronsToAbsoluteZ, OI_MicronsToAbsoluteF</td>
</tr>
</tbody>
</table>
### OI_StepsToMicronsX

Converts a given micron distance into the corresponding microstep distance.

**Syntax**

```c
OI_API OI_StepsToMicronsX( long lSteps, double* pdMicrons )
```

**Parameters**

- `lSteps`: The distance, in microsteps, to be converted.
- `pdMicrons`: The returned distance in microns.

**Return Value**

- `OI_OK` if successful.

**Comments**

The current calibrated microstep size is used for these conversions. This calibration depends on an accurate setting of the microstep size using either the `OI_SetAxisStepSize` function or the `OI_SetPitchXY` function (for motorised stages).

**See Also**

- `OI_SetAxisStepSize`
- `OI_SetPitchXY`
- `OI_MicronsToStepsX`
- `OI_MicronsToStepsY`
- `OI_MicronsToStepsZ`
- `OI_MicronsToStepsF`

---

### General Purpose I/O

The OASIS controller provides a variety of hardware for general purpose input and output functions. These are:
### Port Function Location Comments

<table>
<thead>
<tr>
<th>Port</th>
<th>Function</th>
<th>Location</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTRL1</td>
<td>TTL Input / Output</td>
<td>Main 44-way connector, pin 19</td>
<td></td>
</tr>
<tr>
<td>CTRL2</td>
<td>TLL Input / Output</td>
<td>Main 44-way connector, pin 23</td>
<td></td>
</tr>
<tr>
<td>Open Collector 1</td>
<td>General output control, up to 12V</td>
<td>Main 44-way connector, pin 37</td>
<td>Includes a 100 ohm current limiting resistor.</td>
</tr>
<tr>
<td>Open Collector 2</td>
<td>General output control, up to 12V, 100 mA max</td>
<td>Main 44-way connector, pin 41</td>
<td>No current limiting resistor.</td>
</tr>
<tr>
<td>INPUT0</td>
<td>General TTL-compatible input</td>
<td>Connector PL4, pin 1</td>
<td>3.3V and 5V compatible input</td>
</tr>
<tr>
<td>INPUT1</td>
<td>General TTL-compatible input</td>
<td>Connector PL4, pin 3</td>
<td>3.3V and 5V compatible input</td>
</tr>
<tr>
<td>INPUT2</td>
<td>General TTL-compatible input</td>
<td>Connector PL4, pin 5</td>
<td>3.3V and 5V compatible input</td>
</tr>
<tr>
<td>INPUT3</td>
<td>General TTL-compatible input</td>
<td>Connector PL4, pin 7</td>
<td>3.3V and 5V compatible input</td>
</tr>
</tbody>
</table>

---

**OI_ReadInputPorts**

**Syntax**

```c
OI_API OI_ReadInputPorts( LPBYTE lpbyVal )
```

**Description**

Reads the input ports from connector PL4 on the OASIS controller.

**Parameters**

- `lpbyVal` Pointer to a BYTE returning the input port readings.

**Return Value**

- `OI_OK` if successful.
Comments
Connector PL4 on the OASIS controller provides four input ports. Each port is compatible with 3.3V and 5V inputs.

The status of each input is encoded in the BYTE returned from the `OI_ReadInputPorts` function:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>INPUT0 – pin 1 of PL4 connector</td>
</tr>
<tr>
<td>1</td>
<td>INPUT1 – pin 3 of PL4 connector</td>
</tr>
<tr>
<td>2</td>
<td>INPUT2 – pin 5 of PL4 connector</td>
</tr>
<tr>
<td>3</td>
<td>INPUT3 – pin 7 of PL4 connector</td>
</tr>
<tr>
<td>4 – 7</td>
<td>Not used.</td>
</tr>
</tbody>
</table>

See Also `OI_ReadIO`, `OI_WriteIO`

### OI_ReadIO

**Syntax**

```
OI_API OI_ReadIO( LPBYTE lpbyVal )
```

**Description**

Reads the status of the 2 I/O and 2 Open Collector ports.

**Parameters**

- `bEnabled` Flag enabling or disabling general exception messages.

**Return Value**

This function always returns `OI_OK`.

**Comments**

The status of each port is encoded in the BYTE returned from the `OI_ReadIO` function:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Open Collector 1. A set bit means the transistor is on.</td>
</tr>
<tr>
<td>1</td>
<td>Open Collector 2. A set bit means the transistor is on.</td>
</tr>
<tr>
<td>2</td>
<td>CTRL1. A set bit means the TLL logic is high.</td>
</tr>
<tr>
<td>3</td>
<td>CTRL2. A set bit means the TTL logic is high.</td>
</tr>
<tr>
<td>4 – 7</td>
<td>Not used.</td>
</tr>
</tbody>
</table>

See Also `OI_WriteIO`, `OI_ReadInputPorts`
OI_WriteIO

Syntax

OI_API OI_WriteIO( BYTE byVal )

Description

Writes to the CTRL and Open Collector ports.

Parameters

byVal A BYTE value indicating the logic level for each output.

Return Value

This function always returns OI_OK.

Comments

The BYTE parameter uses the following bits to set the CTRL and Open Collector values:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Open Collector 1. A set bit means the transistor should be turned on.</td>
</tr>
<tr>
<td>1</td>
<td>Open Collector 2. A set bit means the transistor should be turned on.</td>
</tr>
<tr>
<td>2</td>
<td>CTRL1. A set bit means the TLL logic should be set to high.</td>
</tr>
<tr>
<td>3</td>
<td>CTRL2. A set bit means the TTL logic should be set to high.</td>
</tr>
<tr>
<td>4 – 7</td>
<td>Not used.</td>
</tr>
</tbody>
</table>

See Also

OI_ReadIO, OI_ReadInputPorts