
OASIS Automation Controller Software Library

Reference Manual

Version 3.2.1

Objective Imaging Ltd.

PLEASE NOTE: DUE TO A POLICY OF CONTINUOUS DEVELOPMENT, THE INFORMATION PRESENTED IN THIS DOCUMENT MAY BE SUBJECT TO CHANGE.

Please contact Objective Imaging Ltd. for updated information.

Objective Imaging Ltd.

www.objectiveimaging.com

email: info@objectiveimaging.com



Copyright © 2000-2010 Objective Imaging Ltd. All rights reserved.

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
<i>Positional Units and Axis Calibration</i>	14
<i>The Coordinate System</i>	14
<i>Moving a Single Axis</i>	15
<i>Waiting for Movement Completion</i>	15
<i>Moving the XY Stage and Focus</i>	16
<i>Acceleration Tables</i>	21
<i>Cruising Speed</i>	23
<i>Encoder Support</i>	24
SAVING SETTINGS	26
<i>Saving and Loading Settings</i>	26
<i>Saving and Loading Positions</i>	27
RETURN VALUES	27
ADVANCED TOPICS	29
USING MULTIPLE OASIS CONTROLLERS	29
<i>Counting the Number of Installed OASIS Controllers</i>	29
<i>Routing Commands to a Controller</i>	29
<i>Using General Axis Commands with Multiple Controllers</i>	30
<i>Special Considerations When Using Multiple Controllers</i>	31
GENERAL PURPOSE I/O	31
FUNCTION DESCRIPTIONS	33
HARDWARE CONTROL	34
<i>OI_Close</i>	35
<i>OI_CloseComponent</i>	35
<i>OI_Configure</i>	36
<i>OI_CountCards</i>	37
<i>OI_EmergencyStopAll</i>	38
<i>OI_EnableMotorPower</i>	38
<i>OI_EnableMotorPowerEx</i>	38
<i>OI_GetAFFitted</i>	39
<i>OI_GetAHMDelay</i>	39
<i>OI_GetAutoFocusHWMode</i>	40
<i>OI_GetCardAxisCount</i>	40
<i>OI_GetCardType</i>	41
<i>OI_GetComponentStatus</i>	41
<i>OI_GetConfiguration</i>	43
<i>OI_GetDefaultAbortKeys</i>	44
<i>OI_GetDefaultWaitCursorEnabled</i>	44
<i>OI_GetDriverOpen</i>	45
<i>OI_GetFlashChecksum</i>	45
<i>OI_GetHardwareMode</i>	46
<i>OI_GetMultiAxisMode</i>	46
<i>OI_GetSelectedCard</i>	47

<i>OI_GetTotalAxisCount</i>	47
<i>OI_GetUseCount</i>	47
<i>OI_IsModuleFitted</i>	48
<i>OI_Open</i>	49
<i>OI_OpenComponent</i>	49
<i>OI_ReadCardStatus</i>	50
<i>OI_ReadPCBName</i>	50
<i>OI_ReadPCBStatus</i>	51
<i>OI_ReadPCBTemperature</i>	52
<i>OI_ReadPCBType</i>	53
<i>OI_ResetHardware</i>	53
<i>OI_SelectCard</i>	54
<i>OI_SetAHMDelay</i>	54
<i>OI_SetAutoFocusHWMMode</i>	55
<i>OI_SetDefaultAbortKeys</i>	55
<i>OI_SetDefaultWaitCursorEnabled</i>	56
<i>OI_SetHardwareMode</i>	56
<i>OI_SetMultiAxisMode</i>	57
VERSION INFORMATION.....	58
<i>OI_GetDriverVersion</i>	58
<i>OI_ReadPCBID</i>	59
<i>OI_ReadPCBVersion</i>	59
<i>OI_ReadSerialNum</i>	60
GENERAL, SINGLE AXIS CONTROL.....	60
<i>OI_ClearAxisUserLimits</i>	61
<i>OI_DriveAxisContinuous</i>	61
<i>OI_FlashReadAxisPitch</i>	62
<i>OI_GetAxisBacklash</i>	62
<i>OI_GetAxisCruise</i>	63
<i>OI_GetAxisInitMethod</i>	64
<i>OI_GetAxisMaxMove</i>	64
<i>OI_GetAxisPitch</i>	65
<i>OI_GetAxisRamp</i>	65
<i>OI_GetAxisRange</i>	66
<i>OI_GetAxisSense</i>	67
<i>OI_GetAxisStepSize</i>	67
<i>OI_GetAxisStepsPerRev</i>	68
<i>OI_GetAxisTravel</i>	68
<i>OI_GetAxisUserLimits</i>	69
<i>OI_HaltAxis</i>	69
<i>OI_LookupAxisSpeed</i>	70
<i>OI_MoveAxis</i>	70
<i>OI_ReadAxis</i>	71
<i>OI_ReadAxisAtLimit</i>	72
<i>OI_ReadAxisMoving</i>	72
<i>OI_ReadAxisRampValue</i>	73
<i>OI_ReadAxisStatus</i>	73
<i>OI_SetAxisBacklash</i>	74
<i>OI_SetAxisCruise</i>	75
<i>OI_SetAxisEncoderEnabled</i>	75
<i>OI_SetAxisInitMethod</i>	76
<i>OI_SetAxisPitch</i>	77
<i>OI_SetAxisRamp</i>	77
<i>OI_SetAxisSense</i>	78
<i>OI_SetAxisStepSize</i>	79
<i>OI_SetAxisToDefaults</i>	79
<i>OI_SetAxisTravel</i>	80
<i>OI_SetAxisUserLimits</i>	81

<i>OI_StepAxis</i>	81
<i>OI_StepAxisAbs</i>	82
<i>OI_WaitForAxisStopped</i>	83
SIMULTANEOUS THREE AXIS CONTROL	83
<i>OI_DriveContinuousXYZ</i>	83
<i>OI_HaltAllAxes</i>	84
<i>OI_MoveToXYZ</i>	84
<i>OI_MoveToXYZ_Auto</i>	85
<i>OI_ReadMaxMoveXYZ</i>	86
<i>OI_ReadXYZ</i>	87
<i>OI_SetPitchFromFlashXYZ</i>	87
<i>OI_SetPositionXYZ</i>	88
<i>OI_WaitForStoppedXYZ</i>	88
XY STAGE CONTROL	89
<i>OI_ClearUserLimitsXY</i>	89
<i>OI_DriveContinuousXY</i>	90
<i>OI_GetBacklashXY</i>	91
<i>OI_GetCruiseXY</i>	91
<i>OI_GetDriveSenseXY</i>	92
<i>OI_GetFullTravelXY</i>	92
<i>OI_GetPitchXY</i>	93
<i>OI_GetRampXY</i>	93
<i>OI_GetSpeedXY</i>	94
<i>OI_GetUserLimitGuardDistanceXY</i>	94
<i>OI_GetUserLimitsXY</i>	95
<i>OI_HaltXY</i>	95
<i>OI_InitializeXY</i>	96
<i>OI_LookupSpeedXY</i>	96
<i>OI_MoveToXY</i>	97
<i>OI_MoveToXY_Abs</i>	98
<i>OI_MoveToXY_Auto</i>	98
<i>OI_ReadLimitAlarmsXY</i>	99
<i>OI_ReadStatusXY</i>	99
<i>OI_ReadXY</i>	100
<i>OI_ReadXY_Abs</i>	101
<i>OI_SelectSpeedXY</i>	101
<i>OI_SetCruiseXY</i>	102
<i>OI_SetDriveSenseXY</i>	103
<i>OI_SetOriginXY</i>	103
<i>OI_SetPitchXY</i>	104
<i>OI_SetPositionXY</i>	104
<i>OI_SetRampXY</i>	105
<i>OI_SetUserLimitGuardDistanceXY</i>	106
<i>OI_SetUserLimitsXY</i>	107
<i>OI_StepX</i>	107
<i>OI_StepXY</i>	108
<i>OI_StepY</i>	108
<i>OI_WaitForStoppedXY</i>	109
Z / FOCUS CONTROL	109
<i>OI_ClearUserLimitsZ</i>	109
<i>OI_CloseMouseWheelForFocus</i>	110
<i>OI_DriveContinuousZ</i>	110
<i>OI_GetBacklashZ</i>	111
<i>OI_GetCruiseZ</i>	111
<i>OI_GetDriveSenseZ</i>	112
<i>OI_GetMouseWheelPars</i>	112
<i>OI_GetMouseWheelZ</i>	113
<i>OI_GetRampZ</i>	113

<i>OI_GetSpeedZ</i>	114
<i>OI_GetUserLimitsZ</i>	114
<i>OI_HaltZ</i>	114
<i>OI_InitializeZ</i>	115
<i>OI_InitializeZLimits</i>	115
<i>OI_LookupSpeedZ</i>	116
<i>OI_MoveToZ</i>	116
<i>OI_MoveToZ_Abs</i>	117
<i>OI_OpenMouseWheelForFocus</i>	117
<i>OI_ReadLimitAlarmsZ</i>	118
<i>OI_ReadRangeZ</i>	119
<i>OI_ReadStatusZ</i>	119
<i>OI_ReadSyncZ</i>	120
<i>OI_ReadZ</i>	120
<i>OI_ReadZ_Abs</i>	121
<i>OI_RockZ</i>	121
<i>OI_SelectSpeedZ</i>	122
<i>OI_SetCruiseZ</i>	123
<i>OI_SetDriveSenseZ</i>	123
<i>OI_SetMouseWheelPars</i>	124
<i>OI_SetMouseWheelZ</i>	124
<i>OI_SetOriginZ</i>	125
<i>OI_SetPitchZ</i>	125
<i>OI_SetPositionZ</i>	126
<i>OI_SetRampZ</i>	126
<i>OI_SetUserLimitsZ</i>	127
<i>OI_StepZ</i>	128
<i>OI_WaitForStoppedZ</i>	128
F-AXIS (4 TH AXIS) CONTROL	129
<i>OI_ClearUserLimitsF</i>	129
<i>OI_GetCruiseF</i>	129
<i>OI_GetDriveSenseF</i>	129
<i>OI_GetRampF</i>	130
<i>OI_GetSpeedF</i>	130
<i>OI_GetUserLimitsF</i>	131
<i>OI_HaltF</i>	131
<i>OI_InitializeF</i>	132
<i>OI_InitializeFRange</i>	132
<i>OI_LookupSpeedF</i>	133
<i>OI_MoveToF</i>	133
<i>OI_ReadF</i>	134
<i>OI_ReadLimitAlarmsF</i>	134
<i>OI_ReadRangeF</i>	135
<i>OI_ReadStatusF</i>	135
<i>OI_SelectSpeedF</i>	136
<i>OI_SetCruiseF</i>	137
<i>OI_SetDriveSenseF</i>	138
<i>OI_SetOriginF</i>	138
<i>OI_SetPitchF</i>	139
<i>OI_SetPositionF</i>	139
<i>OI_SetRampF</i>	140
<i>OI_SetUserLimitsF</i>	140
<i>OI_StepF</i>	141
<i>OI_WaitForStoppedF</i>	142
T-AXIS (5 TH AXIS) AND S-AXIS (6 TH AXIS) CONTROL	142
<i>OI_ClearUserLimitsT</i>	142
<i>OI_DriveContinuousT</i>	143
<i>OI_GetCruiseT</i>	143

<i>OI_GetDriveSenseT</i>	144
<i>OI_GetRampT</i>	144
<i>OI_GetSpeedT</i>	145
<i>OI_GetUserLimitsT</i>	145
<i>OI_HaltT</i>	146
<i>OI_InitializeT</i>	146
<i>OI_InitializeTRange</i>	147
<i>OI_LookupSpeedT</i>	147
<i>OI_MoveToT</i>	148
<i>OI_ReadLimitAlarmsT</i>	149
<i>OI_ReadRangeT</i>	149
<i>OI_ReadStatusT</i>	150
<i>OI_ReadT</i>	151
<i>OI_SelectSpeedT</i>	151
<i>OI_SetCruiseT</i>	152
<i>OI_SetDriveSenseT</i>	153
<i>OI_SetOriginT</i>	153
<i>OI_SetPitchT</i>	154
<i>OI_SetPositionT</i>	154
<i>OI_SetRampT</i>	155
<i>OI_SetUserLimitsT</i>	156
<i>OI_StepT</i>	156
<i>OI_WaitForStoppedT</i>	157
ENCODERS AND CLOSED-LOOP OPERATION	157
<i>OI_GetAxisEncoderEnabled</i>	158
<i>OI_GetAxisEncoderFitted</i>	158
<i>OI_GetAxisEncoderStepSize</i>	159
<i>OI_GetEncoderClosedLoopResponseXYZ</i>	159
<i>OI_GetEncoderEnabledXY</i>	160
<i>OI_GetEncoderEnabledZ</i>	160
<i>OI_ReadEncoderModule</i>	161
<i>OI_SetEncoderClosedLoopResponseXYZ</i>	161
<i>OI_SetEncoderEnabledXY</i>	162
<i>OI_SetEncoderEnabledZ</i>	163
<i>OI_SetEncoderModule</i>	164
AUTOMATIC FOCUS	166
<i>OI_AutoFocus</i>	166
<i>OI_AutoFocus_Fine</i>	167
<i>OI_AutoFocus_Step</i>	168
<i>OI_AutoFocusEx</i>	168
<i>OI_GetAutoFocus</i>	169
<i>OI_GetAutoFocusEx</i>	169
<i>OI_GetAutoFocusThreshold</i>	170
<i>OI_GetFineFocusSamples</i>	170
<i>OI_ReadFocusProfile</i>	171
<i>OI_ReadFocusScore</i>	172
<i>OI_ReadFocusScoreEx</i>	172
<i>OI_RequestAutoFocusStatus</i>	172
<i>OI_SetAutoFocus</i>	173
<i>OI_SetAutoFocusEx</i>	174
<i>OI_SetAutoFocusThreshold</i>	174
<i>OI_SetFineFocusSamples</i>	175
<i>OI_WaitForAutoFocus</i>	175
PREDICTIVE FOCUS FUNCTIONS	176
<i>OI_GetAutoPredictiveZ</i>	176
<i>OI_GetCoincDomain</i>	177
<i>OI_GetMultiPredictiveZ</i>	178
<i>OI_GetPredictiveFlag</i>	178

<i>OI_GetPredictiveZ</i>	179
<i>OI_GetPredictiveZDomains</i>	180
<i>OI_GetPredictiveZOffset</i>	181
<i>OI_InvalidatePredictiveZ</i>	181
<i>OI_SetAutoPredictiveZ</i>	182
<i>OI_SetMultiPredictiveZ</i>	182
<i>OI_SetPredictiveFlag</i>	183
<i>OI_SetPredictiveZ</i>	184
<i>OI_SetPredictiveZOffset</i>	185
<i>OI_UpdatePredictiveZ</i>	185
VIDEO CAMERA INTERFACE FUNCTIONS	186
<i>OI_GetVideoWindow</i>	187
<i>OI_IsVideoDetected</i>	188
<i>OI_ReadVideoData</i>	188
<i>OI_ReadVideoResults</i>	189
<i>OI_ReadVideoResultsEx</i>	190
<i>OI_ReadVideoResultsXY</i>	191
<i>OI_ReadVideoResultsXYZF</i>	192
<i>OI_ReadVideoResultsZ</i>	193
<i>OI_SetVideoSettings</i>	194
<i>OI_SetVideoWindow</i>	195
FILTER CHANGER FUNCTIONS	196
<i>OI_ClearFilterHomeInfo</i>	197
<i>OI_DeleteFilter</i>	198
<i>OI_GetFilterChanger</i>	198
<i>OI_GetFilterChangerCount</i>	199
<i>OI_GetFilterCount</i>	199
<i>OI_GetFilterDescription</i>	200
<i>OI_GetFilterHomeOffset</i>	200
<i>OI_GetFilterName</i>	201
<i>OI_GetFilterOffset</i>	201
<i>OI_GetFilterPosition</i>	202
<i>OI_GetFilterTimeout</i>	202
<i>OI_GetAvailableShutterCount</i>	202
<i>OI_GetShutter</i>	203
<i>OI_GetShutterEx</i>	203
<i>OI_GetShutterMulti</i>	205
<i>OI_InitFilterChanger</i>	205
<i>OI_MoveToFilter</i>	206
<i>OI_ReadFilterChangerInfo</i>	207
<i>OI_ReadFilterHomeInfo</i>	208
<i>OI_SelectFilterChanger</i>	208
<i>OI_SetFilterCount</i>	209
<i>OI_SetFilterDescription</i>	209
<i>OI_SetFilterHomeOffset</i>	210
<i>OI_SetFilterLocation</i>	210
<i>OI_SetFilterName</i>	211
<i>OI_SetFilterOffset</i>	211
<i>OI_SetFilterTimeout</i>	211
<i>OI_SetShutter</i>	212
<i>OI_SetShutterEx</i>	213
<i>OI_SetShutterMulti</i>	214
<i>OI_WaitForStoppedFilter</i>	215
HARDWARE JOYSTICK AND TRACKBALL FUNCTIONS	215
<i>OI_ClearTrackballStatus</i>	215
<i>OI_GetJoystickEnabled</i>	216
<i>OI_GetTrackballControl</i>	216
<i>OI_GetTrackballEnabled</i>	217

<i>OI_ReadTrackballStatus</i>	217
<i>OI_SetJoystickEnabled</i>	219
<i>OI_SetTrackballControl</i>	219
<i>OI_SetTrackballEnabled</i>	220
TIMEOUTS	221
<i>OI_GetAutoFocusTimeout</i>	221
<i>OI_GetMoveTimeout</i>	222
<i>OI_GetVideoTimeout</i>	222
<i>OI_SetAutoFocusTimeout</i>	223
<i>OI_SetMoveTimeout</i>	223
<i>OI_SetVideoTimeout</i>	223
FILE I/O AND SETTINGS	224
<i>OI_LoadPositions</i>	224
<i>OI_LoadSettings</i>	225
<i>OI_LoadSettingsEx</i>	225
<i>OI_SavePositions</i>	226
<i>OI_SaveSettings</i>	226
<i>OI_SaveSettingsEx</i>	227
ERROR HANDLING	227
<i>OI_GetLastError</i>	227
<i>OI_EnableMsgReportDlg</i>	228
MICRONS / STEP CONVERSION	229
<i>OI_MicronsToAbsoluteX</i>	229
<i>OI_MicronsToAbsoluteY</i>	229
<i>OI_MicronsToAbsoluteZ</i>	229
<i>OI_MicronsToAbsoluteF</i>	229
<i>OI_MicronsToStepsX</i>	230
<i>OI_MicronsToStepsY</i>	230
<i>OI_MicronsToStepsZ</i>	230
<i>OI_MicronsToStepsF</i>	230
<i>OI_StepsToMicronsX</i>	231
<i>OI_StepsToMicronsY</i>	231
<i>OI_StepsToMicronsZ</i>	231
<i>OI_StepsToMicronsF</i>	231
GENERAL PURPOSE I/O	231
<i>OI_ReadInputPorts</i>	232
<i>OI_ReadIO</i>	233
<i>OI_WriteIO</i>	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.

<i>File</i>	<i>Description</i>
OASIS.EXE	Application to setup, test and demonstrate the facilities of the OASIS controller.
OIFLASHCFG.EXE	Application to setup the FLASH memory of the OASIS controller.
OASIS_DLL_MANUAL.PDF	The OASIS DLL Developer Kit manual, in Adobe's PDF format.
OASIS4I.H	Header file that prototypes all the OASIS DLL control functions.
OI_CONST.H	Header file defining various constant values used by the OASIS DLL.
OASIS4I.LIB	Import library file for linking your application to the

<i>File</i>	<i>Description</i>
	OASIS DLL functions.
example\OASISTST.DSP	Project file for the OASISTST MFC example application.
example\OASISTST.CPP	Source file for the OASISTST MFC example application.
example\OASISTST.H	Primary header file for the OASISTST MFC example application.
example\OASISTST.RC	Resource template for the OASISTST MFC example application.
example\RESOURCES.H	The resources header file defining the IDs used by the OASISTST MFC example application.
example\StdAfx.h	Standard system includes header for the OASISTST MFC example application.
example\StdAfx.cpp	Source file for standard precompiled headers.
example\Release\OASISTST.EXE	The OASISTST MFC example application executable.

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.

```

#include "stdafx.h"
#include "oasis4i.h"

int APIENTRY 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-world 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 call **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.

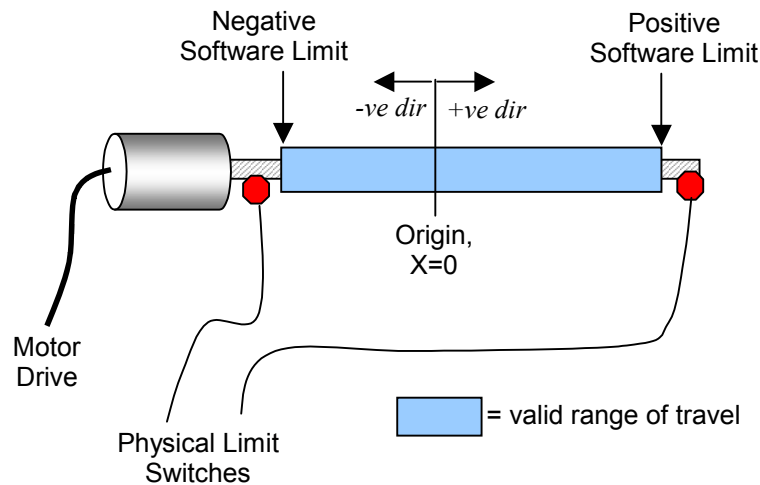


Figure 2. Physical and software limits, with range of travel.

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:

```
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:

```
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:

OI_SetOriginXY()

or

OI_SetPositionXY(double XPos, double YPos)

The **OI_SetOriginXY** function defines the current position as the origin, i.e., XY=[0,0]. The **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:

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:

OI_StepX(double dXDistance, int nWait)
OI_StepY(double dYDistance, int nWait)

or

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:

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.

```

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_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_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_InitializeZ(double ZRangeAbove, double ZRangeBelow)

The **OI_InitializeZ** function does the following:

1. Sets the current position to the origin (Z=0);
2. Sets the positive software limit a distance of *ZRangeAbove* microns above the current position;
3. Sets the negative software limit a distance of *ZRangeBelow* 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.

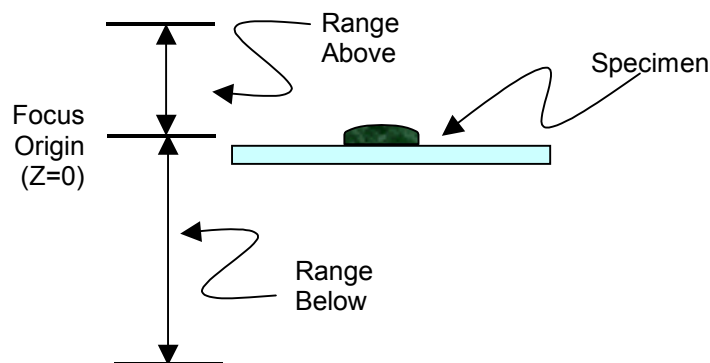


Figure 4. Focus Initialisation.

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:

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:

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:

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.

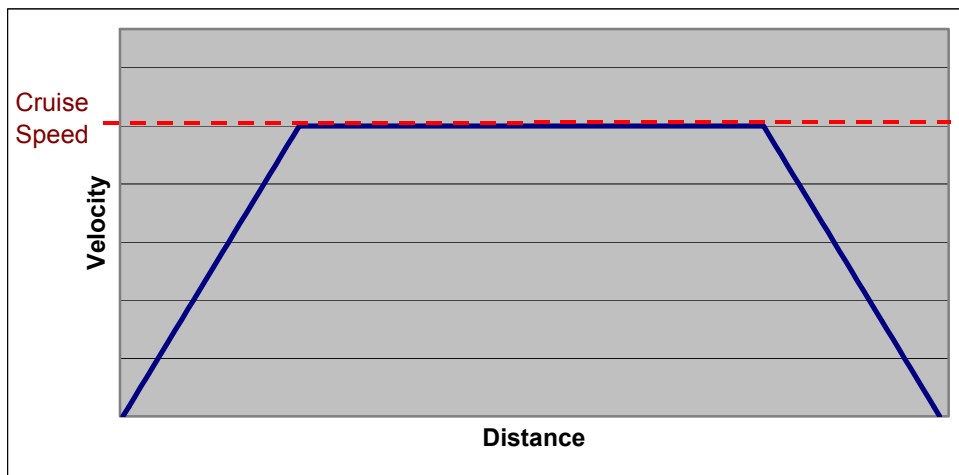


Figure 6. Linear acceleration ramp profile

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.

<i>Index</i>	<i>Timer Interval (microseconds)</i>	<i>Step Size (microsteps)</i>
0	244	1
1	227	1
2	212	1
3	397	2
4	374	2
5	354	2
6	335	2

Figure 7. Example showing the first 7 values of a linear ramp 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 you 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 axes 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 at 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 *bEnabled* parameter. The second *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:

OI_GetAxisEncoderEnabled(int AxisID, LPBOOL lpbEnabled, LPBOOL lpbAutoCorrect)

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 “servo” 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:

OI_SetEncoderEnabledXY(BOOL bEnabledX, int nTolX, BOOL bEnabledY, int nTolY)

OI_SetEncoderEnabledZ(BOOL bEnabledZ, int nTolZ)

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.

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:

OI_GetAxisStepsPerRev(int AxisID, LPDWORD lpdwStepsPerRev)

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:

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:

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:

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:

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:

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:

Return Code	Value	Meaning
OI_OK	0	The operation completed successfully.
OI_FAILED	1	The operation failed.
OI_ABORT	2	The operation failed due to a user abort.
OI_NOHARDWARE	4	The operation failed because the required hardware is not fitted.
OI_TIMEOUT	8	The operation failed due to a timeout.
OI_INVALIDARG	16	An invalid argument was passed into the function.

OI_HARDWAREBUSY	32	The command could not be executed because the hardware was busy with another request.
OI_ACCESSDENIED	64	Access to the desired functionality is not available.
OI_NOTSUPPORTED	128	The operation is not supported by the current hardware platform.
OI_ILLEGALAXIS	256	The operation attempted to use an axis that is not available.
OI_INVALIDCONFIG	512	The requested configuration is not allowed.
OI_MAXMOVEFAIL	1024	A move command failed because it exceeded the maximum allowed move for the specified axis, as defined in the flash memory.

To further investigate hardware failures, use **OI_ReadPCBStatus** to inquire about the specific devices where failure occurred.

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 **OI_CountCards** function to determine how many OASIS cards are installed in a system:

```
OI_CountCards( int* pnNumber )
```

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:

```
OI_SelectCard( int nCard )
```

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

To determine which card is currently active, call:

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

Except in the case where general axis drive commands (such as the **OI_MoveAxis** and **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

OI_SetMultiAxisMode(int nMode)

OI_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.

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

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:

<i>Port</i>	<i>Number Available</i>	<i>Description</i>
General I/O	2	TTL compatible Input and Output
Open Collector	2	Output only
PL4 Input Ports	4	Input only

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:

Group	Meaning
Hardware Control	These functions deal with the overall initialisation, communication, and status of the OASIS hardware.
Version Information	These functions return version information about the OASIS hardware and the OASIS DLL.
General, Single Axis Control	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
Simultaneous Three Axis Control	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.
XY Stage Control	These functions deal exclusively with the two axes of a motorized XY stage.
Z / Focus Control	These functions deal exclusively with the single, Z-axis of a motorized focus mechanism.
F-Axis Control	These functions deal exclusively with the single fourth, or "F", axis available in the OASIS controller.
Encoder and Closed-loop	These functions relate to the use of encoder feedback for position information and closed-loop operation.
Automatic Focus	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.
Predictive Focus	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.

Video and Digital Camera Interface	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.
Filter Changer Functions	The functions provide an interface for controlling filter changer devices, such as a rotating filter wheel.
Hardware Joystick / Trackball Functions	If hardware XY and/or Z axis joystick is fitted to the OASIS controller, these functions may be used to enable its operation.
Timeout Functions	This section describes the functions used to specify the timeout durations used for movement, automatic focus, and video functions.
File I/O Functions	These functions are used to save and restore system settings to and from file.
Error Handling	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.
Micron / Step Conversion Functions	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.
General Purpose Input / Output	Access to the general I/O hardware of the OASIS controller.

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	<i>nComponent</i> 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:

<i>nComponent value</i>	<i>Meaning</i>
OI_CFG_FOCUS	The Z-axis focus control. All functions dealing the Z-axis are routed to the specified controller.
OI_CFG_FILTER	The filter changer. All filter changer functions are routed to the specified controller.

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.

See Also **OI_Close, OI_Configure, OI_OpenComponent**

OI_Configure

Syntax **OI_API OI_Configure(int nComponent, int nControl)**

Description Configures a component for a specific controller.

Parameters	<i>nComponent</i>	The component to be configured, as defined in the comments below.
	<i>pnControl</i>	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.

<i>nComponent value</i>	<i>Meaning</i>
OI_CFG_FOCUS	The Z-axis focus control. All functions dealing the Z-axis are routed to the specified controller.
OI_CFG_FILTER	The filter changer. All filter changer functions are routed to the specified controller.

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:

<i>nControl value</i>	<i>Meaning</i>
OI_OASIS	The OASIS controller is used for the component.
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.

Note that the Leica Microsystems controller uses a RS-232 communications interface, so that a spare serial port is required. Also the RS-232 interface is many orders of magnitude slower than the OASIS's PCI interface, so the performance of Z-axis command operations will be significantly reduced.

See Also **OI_GetConfiguration, OI_OpenComponent, OI_CloseComponent**

OI_CountCards

Syntax OI_API OI_CountCards(int* pnFound)

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

Parameters *pnFound* Returns then 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	<table><tr><td><i>bXYEnabled</i></td><td>Flag indicating whether power to the X and Y axes is to be enabled.</td></tr><tr><td><i>bZFEnabled</i></td><td>Flag indicating whether power to the Z and F axes is to be enabled.</td></tr></table>	<i>bXYEnabled</i>	Flag indicating whether power to the X and Y axes is to be enabled.	<i>bZFEnabled</i>	Flag indicating whether power to the Z and F axes is to be enabled.
<i>bXYEnabled</i>	Flag indicating whether power to the X and Y axes is to be enabled.				
<i>bZFEnabled</i>	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	<table><tr><td><i>pwMotors</i></td><td>Array of motors to enable/disable.</td></tr></table>	<i>pwMotors</i>	Array of motors to enable/disable.
<i>pwMotors</i>	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:

<i>Index</i>	<i>Axis</i>
0	X
1	Y
2	Z
3	F

See Also **OI_EnableMotorPower**

OI_GetAFFitted

Syntax 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 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.

Return Value	<p>OI_OK if successful.</p> <p>If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.</p>
Comments	See OI_SetAHMDelay for further details.
See Also	OI_SetAHMDelay

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	<i>pnMode</i>	<p>Pointer to value Parameter indicating the hardware status. This will be set to either:</p> <p>OI_OASIS (a value of 1), or</p> <p>OI_SIM (a value of 0).</p>
Return Value	<p>OI_OK if successful.</p> <p>OI_NOHARDWARE if the OASIS primary board is not installed.</p>	
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	<i>nCard</i>	Zero-based index of the card in the system.
	<i>pnAxisCount</i>	The number of axes available on the specified card.
Return Value	<p>OI_OK if successful.</p> <p>If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.</p>	

Comments 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 OI_API OI_GetCardType(int nCard, int* pnType)

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

Parameters

<i>nCard</i>	Zero-based index of the card in the system.
<i>pnType</i>	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:

<i>nType value</i>	<i>Controller</i>
OASIS_4I	OASIS-4i
OASIS_BLUE	OASIS-blue

See Also **OI_CountCards**

OI_GetComponentStatus

Syntax OI_API OI_GetComponentStatus(int nComponent, LPDWORD lpdwStatus)

Description Retrieves the current status report for a given component.

Parameters

<i>nComponent</i>	The component.
<i>lpdwStatus</i>	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:

<i>nComponent value</i>	<i>Meaning</i>
OI_CFG_FOCUS	The Z-axis focus control.
OI_CFG_FILTER	The filter changer.
OI_CFG_FILTER2	The second filter changer.
OI_CFG_TURRET	The objective lens turret.
OI_CFG_LAMP1	The lamp channel 1.

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

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

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 OI_API OI_GetConfiguration(int nComponent, int *pnControl)

Description Returns the currently configured controller for a given component..

Parameters

<i>nComponent</i>	The component.
<i>pnControl</i>	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:

<i>nComponent value</i>	<i>Meaning</i>
OI_CFG_FOCUS	The Z-axis focus control. All functions dealing the Z-axis are routed to the specified controller.
OI_CFG_FILTER	The filter changer. All filter changer functions are routed to the specified controller.

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

<i>pnControl value</i>	<i>Meaning</i>
OI_OASIS	The OASIS controller is used for the component.

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_Configure** 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	<i>pbEnabled</i>	Returns the enabling flag.
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	<i>pbEnabled</i>	Returns the enabling flag.
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	OI_API OI_GetHardwareMode(int* pnMode)	
Description	Retrieves the hardware mode of operation, indicating either hardware access or simulated operation.	
Parameters	<i>pnMode</i>	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	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	<i>pnMode</i>	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:		
	<i>nMode</i>	<i>Value</i>	<i>Description</i>
	OI_MULTI_MODE_ID	0	Use 1 to 4*N axes, overrides OI_SelectCard setting (default).
	OI_MULTI_MODE_ROUTE	1	Use 1 to 4 axes, routed to a particular card via the OI_SelectCard setting
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.
 If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.

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)**

Description	Retrieves the total number of users that have connected to the OASIS DLL.	
Parameters	<i>pnUsers</i>	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	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:		
	nModule value	Meaning	Controller
	OASIS_AF	Video autofocus.	OASIS-4i
	OASIS_XA1	Fifth axis.	OASIS-4i
	OASIS_DC1	Trigger I/O.	OASIS-4i
	BLUE_EXPIO	Encoder and Trigger I/O.	OASIS-blue
	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	<i>nComponent</i> 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:

<i>nComponent value</i>	<i>Meaning</i>
OI_CFG_FOCUS	The Z-axis focus control. All functions dealing the Z-axis are routed to the specified controller.
OI_CFG_FILTER	The filter changer. All filter changer functions are routed to the specified controller.

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:

Bit	Code	Meaning
0	S_PCB_MOTOR_VOLTS_OK	1 = Motor supply >= 10V i.e. OK
1	S_PCB_TEMP_OK	1 = PCB Temperature too high
2	S_PCB_ADC_YIN_OK	1 = ADC analogue input Y is zero (correct) at switch on
3	S_PCB_ADC_XIN_OK	1 = ADC analogue input X is zero (correct) at switch on
4	S_PCB_AF_FITTED	1 = OASIS-AF Auto-Focus Module Detected
5	S_PCB_AF_TYPE0	2 bit code indicating module type
6	S_PCB_AF_TYPE1	2 bit code indicating module type
7	S_PCB_VIDEO_ENCODER_OK	1 = Video Encoder configured OK
8	S_PCB_CAMERA_DETECTED	1 = Camera input detected
9	S_PCB_CAMERA_CHANNEL	1 = Camera Channel 3, 0 = Camera channel 0 (default)
10	S_PCB_CAMERA_FREQ	1 = Camera Frequency is 50 Hz, 0 = 60 Hz
11	S_PCB_CAMERA_TYPE	1 = Colour Camera, 0 = Mono Camera
12	S_PCB_SERIAL_DEV_DETECTED	1 = Serial device detected on RS232_0

13	S_PCB_MOUSE_DETECTED	1 = Serial device is standard 2-button mouse
14	S_PCB_TRACKBALL_FITTED	1 = Serial device is Kensington Trackball 5
15		Reserved
16	S_PCB_JOYSTICK_FITTED	1 = Joystick unit fitted
17		Reserved
18	S_PCB_FLASH_OI_OK	1 = Flash OI area checksum OK
19	S_PCB_FLASH_USER_OK	1 = Flash user area checksum OK

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:

```

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:

<i>Type</i>	<i>Value</i>	<i>Description</i>
OASIS_BLUE	1	The controller is an OASIS-blue card.
OASIS_4I	0	The controller is an OASIS-4i card.

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 you're previous settings and positions.

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

OI_SelectCard

Syntax	OI_API OI_SelectCard(int nCard)	
Description	Selects which card is the target for API commands.	
Parameters	<i>nCard</i>	The zero-based index of the card.
Return Value	OI_OK if successful. OI_NOHARDWARE if an OASIS board is not found.	
Comments	<p>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.</p> <p>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.</p>	
See Also	OI_GetSelectedCard , OI_CountCards , OI_SetMultiAxisMode	

OI_SetAHMDelay

Syntax	OI_API OI_SetAHMDelay (DWORD dwMSecs)	
Description	Sets the base delay used when using Leica AHM components.	
Parameters	<i>dwMSecs</i>	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	<p>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.</p> <p>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.</p>	

See Also **OI_GetAHMDelay**

OI_SetAutoFocusHWMode

Syntax	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	<i>nMode</i>	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	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	<i>bEnabled</i>	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	<i>bEnabled</i>	The enabled flag.
Return Value	OI_OK if successful.	
Comments	<p>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.</p> <p>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 that an automation action is currently underway and the system is waiting for the action to complete.</p>	
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	<i>nMode</i>	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 <i>nMode</i>)	
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	<i>nMode</i>	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:

<i>nMode</i>	<i>Value</i>	<i>Description</i>
--------------	--------------	--------------------

<code>OI_MULTI_MODE_ID</code>	0	Use 1 to 4*N axes, overrides OI_SelectCard setting (default).
<code>OI_MULTI_MODE_ROUTE</code>	1	Use 1 to 4 axes, routed to a particular card via the OI_SelectCard setting

The `OI_MULTI_MODE_ID` option allows AxisID values to range from 1 to 4*N 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	<code>OI_API OI_GetDriverVersion(LPSTR lpszVersion, int nStringLen)</code>	
Description	Returns a string containing version information for the OASIS DLL.	
Parameters	<i>lpszVersion</i>	String buffer into which the version information will be copied.
	<i>nStringLen</i>	The size of the string buffer passed in the <i>lpszVersion</i> parameter.
Return Value	<code>OI_OK</code> if successful.	
Comments	The OI_GetDriverVersion returns the file version information for the OASIS4I.DLL file.	

See Also **OI_ReadPCBVersion**

OI_ReadPCBID

Syntax	OI_API OI_ReadPCBID (LPSTR lpszBuffer, int nStringLen)	
Description	Returns a string containing the firmware ID information for the OASIS controller hardware.	
Parameters	<i>lpszBuffer</i>	String buffer into which the version information will be copied.
	<i>nStringLen</i>	The size of the string buffer passed in the <i>lpszVersion</i> 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	OI_API OI_ReadPCBVersion (LPSTR lpszVersion, int nStringLen)	
Description	Returns a string containing the firmware version information for the OASIS controller hardware.	
Parameters	<i>lpszVersion</i>	String buffer into which the version information will be copied.
	<i>nStringLen</i>	The size of the string buffer passed in the <i>lpszVersion</i> 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	<i>nCard</i>	The zero-based index of the desired card. For single card installations, use a value of 0.
	<i>lpszSerialNum</i>	String buffer into which the serial number information will be copied.
	<i>nStringLen</i>	The size of the string buffer passed in the <i>lpszSerialNum</i> 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:

Axis Code	Value
OI_XAXIS	1
OI_YAXIS	2
OI_ZAXIS	3
OI_FAXIS	4
OI_TAXIS	5

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	<i>AxisID</i>	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	<i>nCard</i>	The zero-based index of the card to use, in a multi-card configuration.
	<i>AxisID</i>	The desired axis (see the introduction of this section for the appropriate constants).
	<i>nSpeed</i>	
Return Value	OI_OK if successful. If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.	
Comments	The <i>nSpeed</i> parameter specifies a speed in half-steps per second. The <i>nSpeed</i> 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 `OI_API OI_FlashReadAxisPitch(int AxisID, double *pdPitchMM)`

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).

pdPitchMM The returned pitch of the given axis, 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 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** supersede whatever value is written in the flash memory.

See Also **OI_SetPitchFromFlashXYZ, OI_SetPitchXY, OI_SetPitchZ, OI_SetPitchF**

OI_GetAxisBacklash

Syntax `OI_API OI_GetAxisBacklash(int AxisID, BOOL* pbEnabled)`

Description	Retrieves the status of backlash correction for a given axis.	
Parameters	<i>AxisID</i>	The desired axis (see the introduction of this section for the appropriate constants).
	<i>pbEnabled</i>	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	<p>OI_OK if successful.</p> <p>If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.</p>	
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	<i>nCard</i>	The zero-based index of the card to use, in a multi-card configuration.
	<i>AxisID</i>	The desired axis (see the introduction of this section for the appropriate constants).
	<i>pnCruise</i>	Returns the current cruise index.
Return Value	<p>OI_OK if successful.</p> <p>If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.</p>	
Comments	See the Comments for OI_SetAxisCruise for more information regarding cruise speed.	
See Also	OI_SetAxisCruise, OI_SetAxisRamp, OI_ReadRampValue	

OI_GetAxisInitMethod

Syntax	OI_API OI_GetAxisInitMethod (int AxisID, int *pnMethod)	
Description	Sets the.	
Parameters	<i>AxisID</i>	The desired axis (see the introduction of this section for the appropriate constants).
	<i>pnMethod</i>	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	OI_API OI_GetAxisMaxMove(int AxisID, LPDWORD lpdwValue)	
Description	Retrieves the current pre-defined ramp table in use for a given axis.	
Parameters	<i>AxisID</i>	The desired axis (see the introduction of this section for the appropriate constants).
	<i>lpdwValue</i>	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	<i>nCard</i>	The zero-based index of the card to use, in a multi-card configuration.
	<i>AxisID</i>	The desired axis (see the introduction of this section for the appropriate constants).
	<i>pdPitchMM</i>	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	<i>nCard</i>	The zero-based index of the card to use, in a multi-card configuration.
	<i>AxisID</i>	The desired axis (see the introduction of this section for the appropriate constants).

	<i>pnRamp</i>	Returns the current ramp, as described in the Comments section below.								
Return Value	<p>OI_OK if successful.</p> <p>If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.</p>									
Comments	<p>Three pre-defined tables may be selected for a given axis, as indicated by the <i>pnRamp</i> parameter:</p> <table><tr><th><i>pnRamp value</i></th><th><i>Acceleration</i></th></tr><tr><td>0</td><td>Slow</td></tr><tr><td>1</td><td>Medium</td></tr><tr><td>2</td><td>Fast</td></tr></table>		<i>pnRamp value</i>	<i>Acceleration</i>	0	Slow	1	Medium	2	Fast
<i>pnRamp value</i>	<i>Acceleration</i>									
0	Slow									
1	Medium									
2	Fast									
See Also	OI_SetAxisRamp, OI_SetAxisCruise, OI_GetAxisCruise, OI_GetAxisRampValue									

OI_GetAxisRange

Syntax	OI_API OI_GetAxisRange(int AxisID, double* pdMin, double* pdMax)	
Description	Read the available range of travel for the axis, as defined by the minimum and maximum position values of the User Limits.	
Parameters	<i>AxisID</i>	The desired axis (see the introduction of this section for the appropriate constants).
	<i>pdMin</i>	Returns the minimum coordinate value for the axis, in microns.
	<i>pdMax</i>	Returns the maximum coordinate value for the axis, in microns.
Return Value	<p>OI_OK if successful.</p> <p>If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.</p>	
Comments	<p>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.</p>	
See Also	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	<i>AxisID</i>	The desired axis (see the introduction of this section for the appropriate constants).
	<i>pnSense</i>	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	<i>AxisID</i>	The desired axis (see the introduction of this section for the appropriate constants).
	<i>pdStepSize</i>	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	<i>AxisID</i>	The desired axis (see the introduction of this section for the appropriate constants).
	<i>lpdwStepsPerRev</i>	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	<p>The number of microsteps per revolution determines the resolution of motor stepping. The more microsteps made per revolution, the finer the step size.</p> <p>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 micosteps to encoder steps, such as 2:1.</p>	
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	<i>AxisID</i>	The desired axis (see the introduction of this section for the appropriate constants).
	<i>pdMin</i>	The returned value for the minimum available travel, in microns.
	<i>pdMax</i>	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	<i>AxisID</i>	The desired axis (see the introduction of this section for the appropriate constants).
	<i>pdMin</i>	The returned position of the minimum user limit.
	<i>pdMax</i>	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) OI_API OI_HaltAxisEx(int nCard, int AxisID)	
Description	Stops the indicated axis.	
Parameters	<i>nCard</i>	The zero-based index of the card to use, in a multi-card configuration.

	<i>AxisID</i>	The desired axis (see the introduction of this section for the appropriate constants).
Return Value	<p>OI_OK if successful.</p> <p>If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.</p>	
Comments	<p>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.</p> <p>To immediately stop an axis, without using the deceleration ramp, use the OI_EmergencyStopAll function.</p>	
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	<i>AxisID</i>	The desired axis (see the introduction of this section for the appropriate constants).
	<i>nCruise</i>	The cruise speed index.
	<i>pdSpeed</i>	The returned speed in mm/sec.
Return Value	<p>OI_OK if successful.</p> <p>If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.</p>	
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	<i>nCard</i>	The zero-based index of the OASIS controller to use.
	<i>AxisID</i>	The desired axis (see the introduction of this section for the appropriate constants).
	<i>dValue</i>	The desired position for the move. This value is specified in microns.
	<i>nWait</i>	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 <i>nWait</i> parameter.	
	See the section “Return Values” above for a list of possible return codes.	
See Also	OI_MoveToXY, OI_MoveToZ, OI_MoveToF	

OI_ReadAxis

Syntax	OI_API OI_ReadAxis(int AxisID, double* pdValue)	
	OI_API OI_ReadAxisEx(int nCard, int AxisID, double* pdValue)	
Description	Read the current axis position.	
Parameters	<i>nCard</i>	The zero-based index of the card to use, in a multi-card configuration.
	<i>AxisID</i>	The desired axis (see the introduction of this section for the appropriate constants).
	<i>pdValue</i>	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	

axis.

See Also **OI_ReadXY, OI_ReadZ, OI_ReadF, OI_InitializeXY, OI_InitializeZ, OI_InitializeF**

OI_ReadAxisAtLimit

Syntax **OI_API OI_ReadAxisAtLimit(int AxisID, BOOL* pbAtNegLimit, BOOL* pbAtPosLimit)**

Description Reads whether the axis is at the negative or positive limit of travel.

Parameters	<i>AxisID</i>	The desired axis (see the introduction of this section for the appropriate constants).
	<i>pbAtNegLimit</i>	A flag indicating whether the axis is at the negative limit of travel.
	<i>pbAtPosLimit</i>	A flag indicating whether the axis is at the positive limit 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_ReadAxisAtLimit** function is used to quickly determine with the axis is located at either a user or physical limit of travel.

For a full report of the status of an axis, including whether the physical or user limit is reached, use the **OI_ReadAxisStatus** function.

See Also **OI_ReadAxisStatus, OI_ReadAxisMoving**

OI_ReadAxisMoving

Syntax **OI_API OI_ReadAxisMoving(int AxisID, BOOL* pblsMoving)**

Description Reads whether the axis is currently moving.

Parameters	<i>AxisID</i>	The desired axis (see the introduction of this section for the appropriate constants).
	<i>pblsMoving</i>	A flag indicating whether the axis is moving.
		A TRUE value indicates the axis is in motion.
		A FALSE value indicates the axis is stopped.

Return Value	<p>OI_OK if successful.</p> <p>If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.</p>
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	<i>AxisID</i>	The desired axis (see the introduction of this section for the appropriate constants).
	<i>wIndex</i>	The ramp table index to be read.
	<i>lpwInterval</i>	The returned interval, in microseconds, during which that index is applied.
	<i>lpwStepSize</i>	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	<p>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.</p> <p>The ramp table holds 512 values, leading to valid indices of 0 to 511.</p>	
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)
---------------	---

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	<i>nCard</i>	The zero-based index of the card to use, in a multi-card configuration.
	<i>AxisID</i>	The desired axis (see the introduction of this section for the appropriate constants).
	<i>nCruise</i>	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	<i>AxisID</i>	The desired axis (see the introduction of this section for the appropriate constants).
	<i>bEnabled</i>	Indicates whether the encoder counter is enabled.
	<i>bAutoCorrect</i>	Indicates 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	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	<i>AxisID</i>	The desired axis (see the introduction of this section for the appropriate constants).
	<i>nMethod</i>	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	<p>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.</p> <p>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.</p> <p>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</p>	

function is only relevant to the X and Y axis for stage initialisation.

<i>nMethod</i>	<i>Meaning</i>
0	Normal, seek limit switches at each end of travel
1	Seek only the limit switch at the negative limit of travel.

See Also **OI_GetAxisInitMethod, OI_InitializeXY, OI_SetAxisTravel, OI_GetAxisTravel**

OI_SetAxisPitch

Syntax	OI_API OI_SetAxisPitch(int AxisID, double dPitchMM) OI_API OI_SetAxisPitch (int nCard, int AxisID, double dPitchMM)	
Description	Sets the pre-defined acceleration / deceleration ramp table to use.	
Parameters	<i>nCard</i>	The zero-based index of the card to use, in a multi-card configuration.
	<i>AxisID</i>	The desired axis (see the introduction of this section for the appropriate constants).
	<i>dPitchMM</i>	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	OI_API OI_SetAxisRamp(int AxisID, int nRamp)
	OI_API OI_SetAxisRampEx(int nCard, int AxisID, int nRamp)
Description	Sets the pre-defined acceleration / deceleration ramp table to use.

Parameters	<i>nCard</i>	The zero-based index of the card to use, in a multi-card configuration.
	<i>AxisID</i>	The desired axis (see the introduction of this section for the appropriate constants).
	<i>nRamp</i>	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:

<i>nRamp value</i>	<i>Acceleration</i>
0	Slow
1	Medium
2	Fast

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

<i>AxisID</i>	The desired axis (see the introduction of this section for the appropriate constants).
<i>nSense</i>	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 counter-clockwise 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	<i>AxisID</i>	The desired axis (see the introduction of this section for the appropriate constants).
	<i>dStepSize</i>	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	<i>AxisID</i>	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:

<i>Setting</i>	<i>Default Value</i>
Acceleration Ramp	1 (Normal)
Cruise Speed	200
Step Size	0.078
Backlash	FALSE
Drive Sense	0 (Normal)

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	<i>AxisID</i>	The desired axis (see the introduction of this section for the appropriate constants).
	<i>dMin</i>	The value for the minimum available travel, in microns.
	<i>dMax</i>	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.

See Also **OI_SetAxisInitMethod, OI_InitializeXY, OI_GetAxisTravel**

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	<i>AxisID</i>	The desired axis (see the introduction of this section for the appropriate constants).
	<i>dMin</i>	The position to use as the minimum user limit.
	<i>dMax</i>	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	<i>nCard</i>	The zero-based index of the card to use, in a multi-card configuration.
	<i>AxisID</i>	The desired axis (see the introduction of this section for the appropriate constants).
	<i>dValue</i>	The relative distance to move, in microns.
	<i>nWait</i>	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	<i>nCard</i>	The zero-based index of the card to use, in a multi-card configuration.
	<i>AxisID</i>	The desired axis (see the introduction of this section for the appropriate constants).
	<i>lSteps</i>	The relative distance to move, in microsteps.
	<i>nWait</i>	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

Syntax	OI_API OI_WaitForAxisStopped(int AxisID)	
	OI_API OI_WaitForAxisStoppedEx(int nCard, int AxisID)	
Description	Waits for a given axis to stop moving before returning.	
Parameters	<i>nCard</i>	The zero-based index of the card to use, in a multi-card configuration.
	<i>AxisID</i>	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

Syntax	OI_API OI_DriveContinuousXYZ(int nXSpeed, int nYSpeed, int nZSpeed)	
Description	Drives the X and Y axes at continuous speeds.	
Parameters	<i>nXSpeed</i>	A signed integer indicating the direction and speed at which to drive the X axis.
	<i>nYSpeed</i>	A signed integer indicating the direction and speed at which to drive the Y axis.
	<i>nZSpeed</i>	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 *nXSpeed*, *nYSpeed* and *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 **OI_HaltXY** and/or **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, **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_HaltZ**, **OI_DriveAxisContinuous**, **OI_DriveContinuousXY**, **OI_DriveContinuousZ**

OI_HaltAllAxes

Syntax OI_API OI_HaltAllAxes(void)

Description Immediately stops all axes, using deceleration ramps.

Parameters None.

Return Value OI_OK if successful.

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

See Also **OI_EmergencyStopAll**, **OI_HaltXY**, **OI_HaltZ**, **OI_HaltF**

OI_MoveToXYZ

Syntax 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	<i>dX</i>	The desired X-axis position, in microns.
	<i>dY</i>	The desired Y-axis position, in microns.
	<i>dZ</i>	The desired Z-axis position, in microns.
	<i>nWait</i>	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 <i>nWait</i> parameter tells the function whether to return immediately (i.e., <i>nWait</i> =0) or to wait until all the moves are complete (i.e., <i>nWait</i> 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	<i>dX</i>	The desired X-axis position, in microns.
	<i>dY</i>	The desired Y-axis position, in microns.

	<i>dZ</i>	The desired Z-axis position, in microns.
	<i>nWait</i>	A flag indicating whether the function waits for the move to be completed before returning.
Return Value	<p>OI_OK if successful.</p> <p>If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.</p>	
Comments	<p>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.</p> <p>To test whether the move is complete, use the OI_WaitForStoppedXYZ OI_WaitForAutoFocus functions.</p> <p>NOTE: An OASIS-AF hardware module is required for automatic focus operation.</p>	
See Also	OI_MoveToXYZ, OI_WaitForStoppedXYZ, OI_SetAutoFocus	

OI_ReadMaxMoveXYZ

Syntax	OI_API OI_ReadMaxMoveXYZ (LPDWORD lpdwXSteps, LPDWORD lpdwXSteps, LPDWORD lpdwXSteps)	
Description	Retrieves the current pre-defined ramp table in use for a given axis.	
Parameters	<i>lpdwXSteps</i>	Returns the maximum allowable move for the X axis, as described in the Comments section below.
	<i>lpdwYSteps</i>	Returns the maximum allowable move for the Y axis, as described in the Comments section below.
	<i>lpdwZSteps</i>	Returns the maximum allowable move for the Z axis, as described in the Comments section below.
Return Value	<p>OI_OK if successful.</p> <p>If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.</p>	
Comments	<p>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.</p> <p>This protection is implemented as a maximum allowable move, defined in microsteps. The OI_ReadMaxMoveXYZ function allows the current setting for</p>	

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	<i>pdX</i>	The current X-axis position, in microns.
	<i>pdY</i>	The current Y-axis position, in microns.
	<i>pdZ</i>	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	OI_API OI_SetPositionXYZ(double dX, double dY, double dZ)	
Description	Sets the current position for the X, Y and Z axes.	
Parameters	<i>dX</i>	The desired X-axis position, in microns.
	<i>dY</i>	The desired Y-axis position, in microns.
	<i>dZ</i>	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 <i>dX</i> , <i>dY</i> , and <i>dZ</i> .	
	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	OI_API OI_WaitForStoppedXYZ(int nXWait, int nYWait, int nZWait)
---------------	--

Description	Waits for the X, Y, and Z axes to stop moving.	
Parameters	<i>nXWait</i>	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.
	<i>nYWait</i>	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.
	<i>nZWait</i>	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 **OI_API OI_DriveContinuousXY(int nXSpeed, int nYSpeed)**

Description Drives the X and Y axes at continuous speeds.

Parameters	<i>nXSpeed</i>	A signed integer indicating the direction and speed at which to drive the X axis.
	<i>nYSpeed</i>	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	<i>pdX</i>	The current X-axis backlash correction, in microns.
	<i>pdY</i>	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	<i>pnXCruise</i>	The returned X axis cruise speed index.
	<i>pnYCruise</i>	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* pblnit)	
Description	Returns the full available travel of the XY stage.	
Parameters	<i>pdXMin</i>	The returned X axis minimum value
	<i>pdXMax</i>	The returned X axis maximum value
	<i>pdYMin</i>	The returned Y axis minimum value
	<i>pdYMax</i>	The returned Y axis maximum value
	<i>pblnit</i>	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 <i>pblnit</i> parameter indicates whether the stage has ever been initialised. If it has not, the full limits of travel are undefined.	

See Also **OI_InitializeXY**

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

<i>pdXPitch</i>	The returned X axis lead screw pitch, in millimetres.
<i>pdYPitch</i>	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

<i>pnXRamp</i>	The returned current X-axis ramp, as described in the Comments below.
<i>pnYRamp</i>	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:

<i>nRamp value</i>	<i>Acceleration</i>
0	Slow
1	Medium

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	<i>pdXSpeed</i>	The returned current X-axis speed, in mm/s.
	<i>pdYSpeed</i>	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	<i>pdXDistanceMicrons</i>	The returned value for the X-axis physical to software limit buffer region, in microns.
	<i>pdYDistanceMicrons</i>	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.	

See Also **OI_SetUserLimitGuardDistanceXY, OI_SetAxisInitMethod, OI_InitializeXY, OI_GetAxisTravel**

OI_GetUserLimitsXY

Syntax **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	<i>pdXMin</i>	The minimum coordinate for the X axis, in microns.
	<i>pdXMax</i>	The maximum coordinate for the X axis, in microns.
	<i>pdYMin</i>	The minimum coordinate for the Y axis, in microns.
	<i>pdYMax</i>	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_InitializeXY**

OI_HaltXY

Syntax **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	<p>Stage initialisation is necessary for determining the precise range of travel available in the X and Y directions of travel.</p> <p>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.</p> <p>The initialisation procedure is defined as:</p> <ol style="list-style-type: none">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. <p>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.</p> <p>Note that some stages are fitted with adjustable limit switches, and any change to the position of these switches will require another stage initialisation.</p>

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	<i>nCruiseX, nCruiseY</i>	The cruise speeds of the X and Y axes for which the actual speed is desired.
	<i>pdSpeedX, pdSpeedY</i>	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	OI_API OI_MoveToXY(double dX, double dY, int nWait)	
Description	Moves to the specified X-Y position.	
Parameters	<i>dX</i>	The desired X-axis position, in microns.
	<i>dY</i>	The desired Y-axis position, in microns.
	<i>nWait</i>	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 IX, long IY, int nWait)	
Description	Moves to the specified X-Y position, in microsteps.	
Parameters	<i>IX</i>	The desired X-axis position, in microsteps.
	<i>IY</i>	The desired Y-axis position, in microsteps.
	<i>nWait</i>	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	<i>dX</i>	The desired X-axis position, in microns.
	<i>dY</i>	The desired Y-axis position, in microns.
	<i>nWait</i>	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 and 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

<i>pnXNeg</i>	Status of the negative limit for the X axis.
<i>pnXPos</i>	Status of the positive limit for the X axis.
<i>pnYNeg</i>	Status of the negative limit for the Y axis.
<i>pnYPos</i>	Status of the positive limit for 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 **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:

Status Code	Meaning
0	The axis is not at the limit
1	The axis is at a user limit
2	The axis is at a hardware limit
3	The axis is at both a user and a hardware limit

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:

Status Code	Meaning
S_LIMIT_PHY_NEG	The axis is at the negative physical limit
S_LIMIT_USR_NEG	The axis is at the negative user limit
S_LIMIT_PHY_POS	The axis is at the positive physical limit
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_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* pIX, long* pIY)	
Description	Reads the current X-Y position of the stage, in microsteps.	
Parameters	<i>pIX</i>	The current X-axis position, in microsteps.
	<i>pIY</i>	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	<i>dXmmPerSec, dYmmPerSec</i>	The desired speeds for the X and Y axis, in mm per second.
	<i>nFlags</i>	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 *nFlags* parameter specifies how the search is to be carried out:

<i>nFlags value</i>	<i>Meaning</i>
0	A cruise value is found that gives an actual speed as close to, but not exceeding, the desired speed.
1	A cruise value is found that gives the closest actual speed to the desired speed, including speeds that are greater than the desired speed.

The net effect of the **OI_SelectSpeedXY** function is equivalent to a call to **OI_SetCruiseXY** with parameters that give the best match to the desired actual speeds.

Note that you may use the **OI_GetSpeedXY** and the **OI_GetCruiseXY** to read the actual speeds and cruise values that have been selected.

See Also **OI_GetSpeedXY, OI_GetCruiseXY, OI_SetCruiseXY**

OI_SetCruiseXY

Syntax **OI_API OI_SetCruiseXY(int nXCruise, int nYCruise)**

Description Sets the cruising speed for the X and Y axes.

Parameters *nXCruise* The X axis cruise speed index.
 nYCruise The 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 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_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 **OI_API OI_SetPitchXY(double dXPitch, double dYPitch)**

Description Sets the pitch of the lead screws for the X and Y axes.

Parameters

<i>dXPitch</i>	The X axis lead screw pitch, in millimetres.
<i>dYPitch</i>	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 **OI_API OI_SetPositionXY(double dX, double dY)**

Description	Sets the current position for the X and Y axes.	
Parameters	<i>dX</i>	The desired X-axis position, in microns.
	<i>dY</i>	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	<p>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 <i>dX</i> and <i>dY</i>.</p> <p>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.</p>	
See Also	OI_SetPositionXYZ, OI_SetPositionZ, OI_SetPositionF, OI_SetOriginXY, OI_SetOriginZ, OI_SetOriginF, OI_InitializeXY, OI_InitializeZ, OI_InitializeF	

OI_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	<i>nXRamp</i>	The X-axis ramp code.
	<i>nYRamp</i>	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 <i>nRamp</i> parameter:	

<i>nRamp value</i>	<i>Acceleration</i>
0	Slow
1	Medium
2	Fast

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 is 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	<i>dXMin</i>	The minimum coordinate for the X axis, in microns.
	<i>dXMax</i>	The maximum coordinate for the X axis, in microns.
	<i>dYMin</i>	The minimum coordinate for the Y axis, in microns.
	<i>dYMax</i>	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	<p>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.</p> <p>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.</p> <p>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.</p>	
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	<i>dXDistance</i>	The desired distance to move, in microns.
	<i>nWait</i>	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 <i>dXDistance</i> 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

<i>dXDistance</i>	The desired distance to move the X axis, in microns.
<i>dYDistance</i>	The desired distance to move the Y axis, in microns.
<i>nWait</i>	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

<i>dYDistance</i>	The desired distance to move, in microns.
<i>nWait</i>	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

Syntax	OI_API OI_WaitForStoppedXY()
Description	Waits for the X and Y axes 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_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

Syntax	OI_API OI_ClearUserLimitsZ(void)
Description	Clears the user limits for 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 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	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	OI_API OI_DriveContinuousZ(int nSpeed)
Description	Drives the Z axis at a continuous speed.
Parameters	<i>nSpeed</i> 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 <i>nSpeed</i> 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	<code>OI_API OI_GetBacklashZ(double* pdZ,)</code>
Description	Reads the current Z backlash correction information of the focus.
Parameters	<i>pdZ</i> 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	<code>OI_API OI_GetCruiseZ(int* pnZCruise)</code>
Description	Retrieves the current Z axis cruise speed index.
Parameters	<i>pnZCruise</i> 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	OI_API OI_GetDriveSenseZ(int* pnZDir)	
Description	Retrieves the current direction of rotation setting for the Z axis.	
Parameters	<i>pnZDir</i>	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	OI_API OI_GetMouseWheelPars(double *pdStepSize, int *pnSpeed)	
Description	Retrieves the current settings for control of the Z axis using the mouse wheel.	
Parameters	<i>pdStepSize</i>	The step size, in microns, for each step of the mouse wheel.
	<i>pnSpeed</i>	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	OI_API OI_GetMouseWheelZ(BOOL *pbEnabled)
Description	Retrieves whether mouse wheel control of the Z axis is enabled.

Parameters	<i>pbEnabled</i>	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	OI_API OI_GetRampZ(int* pnZRamp)	
Description	Retrieves which pre-defined acceleration / deceleration ramp is in use for the Z axis.	
Parameters	<i>pnZRamp</i>	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 <i>nRamp</i> parameter:	

<i>nRamp value</i>	<i>Acceleration</i>
0	Slow
1	Medium
2	Fast

See Also	OI_SetRampZ , OI_SetAxisRamp
-----------------	--

OI_GetSpeedZ

Syntax	OI_API OI_GetSpeedZ(double* pdSpeed)	
Description	Retrieves the current speeds, in mm per second, in use by the Z axis.	
Parameters	<i>pdSpeed</i>	The returned current Z-axis speed, in mm/s.

Return Value	<p>OI_OK if successful.</p> <p>If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.</p>
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	OI_API OI_GetUserLimitsZ(double* pdZMin, double* pdZMax)	
Description	Retrieves the current user limit settings for the Z axis.	
Parameters	<i>pdZMin</i>	The minimum coordinate for the Z axis, in microns.
	<i>pdZMax</i>	The maximum coordinate for the Z axis, in microns.
Return Value	<p>OI_OK if successful.</p> <p>If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.</p>	
Comments	See the Comments for the OI_SetUserLimitsZ for more information about user limits.	
See Also	OI_SetUserLimitsZ, OI_InitializeZ	

OI_HaltZ

Syntax	OI_API OI_HaltZ(void)	
Description	Stops any motion of the Z axis.	
Parameters	None.	
Return Value	<p>OI_OK if successful.</p> <p>If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.</p>	
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_InitializeZ

Syntax	OI_API OI_InitializeZ(double dZRangeAbove, double dZRangeBelow)	
Description	Initialises the Z axis for focus control.	
Parameters	<i>dZRangeAbove</i>	The desired allowable distance above (e.g., in the positive direction) the current position, in microns.
	<i>dZRangeBelow</i>	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_InitializeZLimits

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	<i>nCruise</i>	The cruise speed for which the actual speed is desired.
	<i>pdSpeed</i>	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 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	<i>dZ</i>	The desired Z position, in microns.
	<i>nWait</i>	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 <i>nWait</i> 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 IZ, int nWait)	
Description	Moves to a given Z position, in microsteps.	
Parameters	<i>IZ</i>	The desired Z position, in microsteps.
	<i>nWait</i>	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 <i>nWait</i> 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	<i>hinstModule</i>	The calling application's module instance handle.
	<i>dwThreadId</i>	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 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** functions 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:

Status Code	Meaning
0	The axis is not at the limit
1	The axis is at a user limit
2	The axis is at a hardware limit
3	The axis is at both a user and a hardware limit

See Also **OI_ReadStatusZ, OI_ReadLimitAlarmsXY, OI_ReadLimitAlarmsF**

OI_ReadRangeZ

Syntax 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:

Status Code	Meaning
S_LIMIT_PHY_NEG	The axis is at the negative physical limit
S_LIMIT_USR_NEG	The axis is at the negative user limit
S_LIMIT_PHY_POS	The axis is at the positive physical limit
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
S_MOTOR_DETECTED	A motor was detected on the axis on startup.

See Also **OI_ReadStatusXY, OI_ReadStatusF, OI_ReadAxisStatus**

OI_ReadSyncZ

Syntax	OI_API OI_ReadSyncZ(LPWORD pwFieldNum, double *pdZ)	
Description	Reads the Z-axis position synchronized with the most recent camera frame.	
Parameters	<i>pwFieldNum</i>	The camera field index counter value.
	<i>pdZ</i>	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	OI_API OI_ReadZ(double *pZ)	
Description	Reads the current Z axis position.	
Parameters	<i>pZ</i>	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	OI_API OI_ReadZ_Abs(long* pIZ)	
Description	Reads the current Z axis position, in microsteps.	
Parameters	<i>pIZ</i>	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	OI_API OI_RockZ(BOOL bOn, double dZRange, int nSpeed)	
Description	Enables / disables continuous movement of the focus over a range of travel.	
Parameters	<i>bOn</i>	TRUE starts the movement; FALSE will terminate the movement.
	<i>dZRange</i>	The desired range of travel, in microns, above and below the current Z position.
	<i>nSpeed</i>	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	OI_API OI_SelectSpeedZ(double dMmPerSec, int nFlags)
---------------	--

Description Automatically selects the cruise speed corresponding to a desired speed in mm per second.

Parameters

<i>dMmPerSec</i>	The desired speeds for the Z axis, in mm per second.
<i>nFlags</i>	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:

<i>nFlags</i> value	Meaning
0	A cruise value is found that gives an actual speed as close to, but not exceeding, the desired speed.
1	A cruise value is found that gives the closest actual speed to the desired speed, including speeds that are greater than the desired speed.

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

<i>nZCruise</i>	The Z axis cruise speed index.
-----------------	--------------------------------

Return Value	<p>OI_OK if successful.</p> <p>If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.</p>
Comments	<p>The cruise speed is specified via the maximum index to be used in the currently defined acceleration / deceleration ramp for a given axis.</p> <p>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.</p> <p>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.</p>
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	<p><i>nZDir</i> The Z axis drive sense.</p>
Return Value	<p>OI_OK if successful.</p> <p>If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.</p>
Comments	<p>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.</p> <p>A value of zero (0) indicates standard movement.</p> <p>A non-zero value indicates reversed movement.</p>
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	<i>dStepSize</i>	The step size, in microns, for each step of the mouse wheel.
	<i>nSpeed</i>	The desired maximum speed of movement to continuous driving when the mouse wheel is pressed.
Return Value	OI_OK if successful.	
Comments	<p>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.</p> <p>The OI_SetMouseWheelPars function defines the parameters used for stepping and driving the focus using the mouse wheel.</p>	
See Also	OI_GetMouseWheelPars, OI_OpenMouseWheelForFocus , OI_CloseMouseWheelForFocus	

OI_SetMouseWheelZ

Syntax	OI_API OI_SetMouseWheelZ(BOOL bEnabled)	
Description	Sets whether mouse wheel control of the Z axis is enabled.	
Parameters	<i>bEnabled</i>	Set to TRUE to enable, FALSE to disable.
Return Value	OI_OK if successful.	
Comments	<p>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.</p> <p>To terminate the hook function used to trap the mouse wheel events, use the OI_CloseMouseWheelForFocus function.</p>	
See Also	OI_GetMouseWheelPars, OI_OpenMouseWheelForFocus, OI_CloseMouseWheelForFocus	

OI_SetOriginZ

Syntax	OI_API OI_SetOriginZ(void)
Description	Sets the current Z position to be the origin (e.g., 0).

Parameters	None.
Return Value	<p>OI_OK if successful.</p> <p>If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.</p>
Comments	<p>The OI_SetOriginZ function is used to establish the origin of the overall, micron-based coordinate system of the Z focus.</p> <p>The origin is defined to be position [Z=0], and all positions are made relative to this origin.</p> <p>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</p> <p>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.</p>
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	OI_API OI_SetPitchZ(double dZPitch)	
Description	Sets the current position for the Z axis.	
Parameters	<i>dZPitch</i>	The pitch, in millimetres, of the Z axis.
Return Value	<p>OI_OK if successful.</p> <p>If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.</p>	
Comments	<p>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.</p> <p>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.</p> <p>To retrieve the current step size value, you may use the OI_GetAxisStepSize function.</p>	

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	<i>dZ</i>	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 <i>dZ</i> . 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	<i>nZRamp</i>	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 <i>nRamp</i> parameter:	

<i>nRamp value</i>	<i>Acceleration</i>
0	Slow
1	Medium
2	Fast

See Also **OI_GetRampZ, OI_SetRampXY, OI_SetRampF, OI_SetAxisRamp**

OI_SetUserLimitsZ

Syntax 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** functions 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 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.

	<i>nWait</i>	A flag indicating whether the function waits for the move to be completed before returning.
Return Value	<p>OI_OK if successful.</p> <p>If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.</p>	
Comments	<p>If the <i>nWait</i> 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.</p>	
See Also	OI_WaitForStoppedXYZ, OI_MoveToZ	

OI_WaitForStoppedZ

Syntax	OIAPI OI_WaitForStoppedZ (void)	
Description	Waits for the Z axis to stop moving.	
Parameters	None.	
Return Value	<p>OI_OK if successful.</p> <p>If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.</p>	
Comments	<p>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.</p>	
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	<p>OI_OK if successful.</p> <p>If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.</p>
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_GetCruiseZ(int* pnFCruise)		
Description	Retrieves the current F axis cruise speed index.		
Parameters	<i>pnFCruise</i>	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	<i>pnFDir</i>	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	OI_API OI_GetRampF(int* pnFRamp)	
Description	Retrieves which pre-defined acceleration / deceleration ramp is in use for the F axis.	
Parameters	<i>pnFRamp</i>	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 <i>nRamp</i> parameter:	
	<i>nRamp value</i>	<i>Acceleration</i>
	0	Slow
	1	Medium
	2	Fast
See Also	OI_SetRampF, OI_SetAxisRamp	

OI_GetSpeedF

Syntax	OI_API OI_GetSpeedF(double* pdSpeed)	
Description	Retrieves the current speeds, in mm per second, in use by the F axis.	
Parameters	<i>pdSpeed</i>	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 4 th 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	<i>pdFMin</i>	The minimum coordinate for the F axis, in microns.
	<i>pdFMax</i>	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	<p>OI_OK if successful.</p> <p>OI_ABORT if the user aborts the initialisation process using either the ESC or CTRL-C key press.</p> <p>If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.</p>
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	<i>dFNegLimit</i>	The desired allowable distance in the negative direction from the current position, in microns.
	<i>dFPosLimit</i>	The desired allowable distance in the positive direction from the current position, in microns.
Return Value	<p>OI_OK if successful.</p> <p>If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.</p>	
Comments	<p>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.</p> <p>The current position is set to zero, and the F user limits are set to the given ranges above and below the current position.</p>	
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	<i>nCruise</i>	The cruise speed for which the actual speed is desired.
	<i>pdSpeed</i>	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 4 th 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	<i>dF</i>	The desired F position, in microns.
	<i>nWait</i>	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 <i>nWait</i> 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_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, OI_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:

Status Code	Meaning
0	The axis is not at the limit
1	The axis is at a user limit
2	The axis is at a hardware limit

3	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:

Status Code	Meaning
S_LIMIT_PHY_NEG	The axis is at the negative physical limit
S_LIMIT_USR_NEG	The axis is at the negative user limit
S_LIMIT_PHY_POS	The axis is at the positive physical limit

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 **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:

<i>nFlags value</i>	<i>Meaning</i>
0	A cruise value is found that gives an actual speed as close to, but not exceeding, the desired speed.
1	A cruise value is found that gives the closest actual speed to the desired speed, including speeds that are greater than the desired speed.

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	<i>nFCruise</i>	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	<i>nFDir</i>	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.

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 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 OI_API OI_SetPitchF(double dFPitch)

Description	Sets the current position for the F axis.	
Parameters	<i>dFPitch</i>	The pitch, in millimetres, of the F axis.
Return Value	<p>OI_OK if successful.</p> <p>If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.</p>	
Comments	<p>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.</p> <p>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.</p> <p>To retrieve the current step size value, you may use the OI_GetAxisStepSize function.</p> <p>Consult the specifications for your specific F axis to determine the actual lead screw pitch.</p>	
See Also	OI_SetAxisStepSize, OI_GetAxisStepSize, OI_SetPitchXY, OI_SetPitchZ	

OI_SetPositionF

Syntax	OI_API OI_SetPositionF(double dF)	
Description	Sets the current position for the F axis.	
Parameters	<i>dF</i>	The desired F-axis position, in microns.
Return Value	<p>OI_OK if successful.</p> <p>If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.</p>	
Comments	<p>The OI_SetPositionF function resets the coordinate system for the F axis. The current F position is redefined to be the values passed in <i>dF</i>.</p> <p>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.</p>	
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	<i>nFRamp</i>	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 <i>nRamp</i> parameter:	
	<i>nRamp value</i>	<i>Acceleration</i>
	0	Slow
	1	Medium
	2	Fast
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	<i>dFMin</i>	The minimum coordinate for the F axis, in microns.
	<i>dFMax</i>	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 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	<i>dFDistance</i>	The desired distance for the F move, in microns.
	<i>nWait</i>	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 <i>nWait</i> 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 daughter 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	<i>nSpeed</i>	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 <i>nSpeed</i> 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	<i>pnCruise</i>	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	OI_API OI_GetDriveSenseT(int* pnDir)	
	OI_API OI_GetDriveSenseS(int* pnDir)	
Description	Retrieves the current direction of rotation setting for the axis.	
Parameters	<i>pnDir</i>	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	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	<i>pnRamp</i>	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 <i>pnRamp</i> parameter:	
	<i>pnRamp value</i>	<i>Acceleration</i>
	0	Slow
	1	Medium
	2	Fast
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	<i>pdSpeed</i>	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 5 th 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	<i>pdMin</i>	The minimum coordinate for the axis, in microns.
	<i>pdMax</i>	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	<p>OI_OK if successful.</p> <p>If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.</p>
Comments	<p>The OI_HaltT function uses the T axis's deceleration ramp to stop the stage smoothly.</p> <p>To immediately stop the stage without using the deceleration ramp, use the OI_EmergencyStopAll function.</p>
See Also	OI_EmergencyStopAll

OI_InitializeT

Syntax	<p>OI_API OI_InitializeT(void)</p> <p>OI_API OI_InitializeS(void)</p>
Description	Initialises the axis by searching for negative and positive physical limit switches.
Parameters	None.
Return Value	<p>OI_OK if successful.</p> <p>OI_ABORT if the user aborts the initialisation process using either the ESC or CTRL-C key press.</p> <p>If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.</p>
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_InitializeTRange, OI_ReadRangeT

OI_InitializeTRange

Syntax	<p>OI_API OI_InitializeTRange(double dNegLimit, double dPosLimit)</p> <p>OI_API OI_InitializeSRange(double dNegLimit, double dPosLimit)</p>
---------------	---

Description	Initialises the axis using a specified distance on each side of the current position.	
Parameters	<i>dNegLimit</i>	The desired allowable distance in the negative direction from the current position, in microns.
	<i>dPosLimit</i>	The desired allowable distance in the positive direction from 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	In situation where the T axis is fitted to a device that does not provide physical limit switches, use the 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	OI_InitializeT, OI_ReadRangeT	

OI_LookupSpeedT

Syntax	OI_API OI_LookupSpeedT(int nCruise, double* pdSpeed)	
	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	<i>nCruise</i>	The cruise speed for which the actual speed is desired.
	<i>pdSpeed</i>	The returned current 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_LookupSpeedT returns the actual drive speed of the 5 th 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_GetSpeedT function, which returns the speed corresponding to the currently selected cruise, the OI_LookupSpeedT function returns the speed for a given cruise value.	

See Also **OI_GetSpeedT, OI_SelectSpeedT, OI_GetCruiseT**

OI_MoveToT

Syntax	OI_API OI_MoveToT (double dPos, int nWait)	
	OI_API OI_MoveToS (double dPos, int nWait)	
Description	Moves to a given position.	
Parameters	<i>dPos</i>	The desired position, in microns.
	<i>nWait</i>	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 <i>nWait</i> 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	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	<i>pnNeg</i>	Status of the negative limit for the axis.
	<i>pnPos</i>	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	Meaning
0	The axis is not at the limit
1	The axis is at a user limit
2	The axis is at a hardware limit
3	The axis is at both a user and a hardware limit

See Also **OI_ReadStatusT**, **OI_ReadLimitAlarmsXY**, **OI_ReadLimitAlarmsZ**, **OI_ReadLimitAlarmsF**

OI_ReadRangeT

Syntax **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 **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:

Status Code	Meaning
S_LIMIT_PHY_NEG	The axis is at the negative physical limit
S_LIMIT_USR_NEG	The axis is at the negative user limit
S_LIMIT_PHY_POS	The axis is at the positive physical limit
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_ReadStatusF, OI_ReadAxisStatus**

OI_ReadT

Syntax **OI_API OI_ReadT(double *pPos)**

OI_API OI_ReadS(double *pPos)

Description Reads the current axis position.

Parameters *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	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	<i>dMmPerSec</i>	The desired speeds for the axis, in mm per second.
	<i>nFlags</i>	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 5 th axis cruise speed to a specified actual speed target, in mm per second.	
	The <i>nFlags</i> parameter specifies how the search is to be carried out:	

<i>nFlags value</i>	<i>Meaning</i>
0	A cruise value is found that gives an actual speed as close to, but not exceeding, the desired speed.
1	A cruise value is found that gives the closest actual speed to the desired speed, including speeds that are greater than the desired speed.

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 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	<i>nCruise</i>	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	<p>The cruise speed is specified via the maximum index to be used in the currently defined acceleration / deceleration ramp for a given axis.</p> <p>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.</p> <p>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.</p>	
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	<i>nDir</i>	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	<p>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.</p> <p>A value of zero (0) indicates standard movement.</p> <p>A non-zero value indicates reversed movement.</p>	
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	<p>The OI_SetOriginF function is used to establish the origin of the overall, micron-based coordinate system of the F axis.</p> <p>The origin is defined to be position [F=0], and all positions are made relative to this origin.</p> <p>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</p> <p>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.</p>
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	<i>dPitch</i> 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 **OI_API OI_SetPositionT(double dNewPos)**

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	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	<i>nRamp</i>	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 <i>nRamp</i> parameter:	

<i>nRamp value</i>	<i>Acceleration</i>
0	Slow
1	Medium
2	Fast

See Also OI_GetRampF, OI_SetRampXY, OI_SetRampF, OI_SetAxisRamp

OI_SetUserLimitsT

Syntax	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	<i>dMin</i>	The minimum coordinate for the axis, in microns.
	<i>dMax</i>	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	<i>dDistance</i>	The desired distance for the move, in microns.
	<i>nWait</i>	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	<i>AxisID</i>	The desired axis (see the introduction of this section for the appropriate constants).
	<i>lpbEnabled</i>	Returns whether the encoder counter is enabled.
	<i>lpbAutoCorrect</i>	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 OI_API OI_GetAxisEncoderFitted(int AxisID, LPBOOL lpbFitted)

Description Retrieves whether an encoder has been configured as fitted for a given axis.

Parameters

<i>AxisID</i>	The desired axis (see the introduction of this section for the appropriate constants).
<i>lpbFitted</i>	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 OI_API OI_GetAxisEncoderStepSize(int AxisID, double *pdMicrons)

Description Retrieves the step size in microns for a given axis' encoder, if fitted.

Parameters

<i>AxisID</i>	The desired axis (see the introduction of this section for the appropriate constants).
<i>pdMicrons</i>	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* pnResponse Z)	
Description	Returns the closed loop response rate for XYZ.	
Parameters	<i>pnResponseX</i> <i>pnResponseY</i> <i>pnResponseZ</i>	Returns the closed loop response rate for the specified axis, as follows: 0 = slow 1 = fast
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* pnTolX, LPBOOL pbEnabledY, int* pnTolY)	
Description	Defines the encoder setup for the X and Y axes.	
Parameters	<i>pbEnabledX</i> <i>pnTolX</i>	Returns the enabled state of the X axis encoder. Returns the tolerance of the closed-loop mode for

	the X axis encoder.
<i>pbEnabledY</i>	Returns the enabled state of the Y axis encoder.
<i>pnTolY</i>	Returns the tolerance of the closed-loop mode for the Y axis encoder.
Return Value	<p>OI_OK if successful.</p> <p>If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.</p>
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	<i>pbEnabledZ</i>	Returns the enabled state of the Z axis encoder.
	<i>pnTolZ</i>	Returns the tolerance of the closed-loop mode for the Z axis encoder.
Return Value	<p>OI_OK if successful.</p> <p>If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.</p>	
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_GetEncoderEnabledXY	

OI_ReadEncoderModule

Syntax	OI_API OI_ReadEncoderModule(LPWORD pwInfo, LPWORD pwOptions, LPWORD pwOutputPulseWidth_usec, LPDWORD pdwComparator1, LPDWORD pdwComparator2, LPDWORD pdwReserved, LPDWORD pdwReserved2)
Description	Retrieves the information for the BLUE-EXPIO triggering.

Parameters	<i>pwInfo</i>	BLUE-EXPIO module ID Register, 8 bits, provides Module/Altera version information - 0xFF = Module not fitted.
	<i>pwOptions</i>	BLUE-EXPIO control register value.
	<i>pwOutputPulseWidth_usec</i>	Output pulse width, in microseconds.
	<i>pdwComparator1</i>	Comparator 1 interval, in encoder counts.
	<i>pdwComparator2</i>	Comparator 2 interval, in encoder counts.
	<i>pdwReserved</i> , <i>pdwReserved2</i>	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

Syntax	OI_API OI_SetEncoderClosedLoopResponseXYZ (int nResponseX, int nResponseY, int nResponse Z)	
Description	Specifies the closed loop response rate for XYZ.	
Parameters	<i>nResponseX</i> <i>nResponseY</i> <i>nResponseZ</i>	The closed loop response rate for the specified axis, as follows: 0 = slow 1 = fast
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 nToIX, BOOL bEnabledY, int nToIY)	
Description	Defines the encoder setup for the X and Y axes.	
Parameters	<i>bEnabledX</i>	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.
	<i>nToIX</i>	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.
	<i>bEnabledY</i>	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.
	<i>nToIY</i>	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	<p>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.</p> <p>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.</p> <p>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.</p>	

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 changes 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	OI_API OI_SetEncoderEnabledZ(BOOL bEnabledZ, int nTolZ)	
Description	Defines the encoder setup for the Z axis.	
Parameters	<i>bEnabledZ</i>	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.
	<i>nTolZ</i>	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 changes 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 **OI_API OI_SetEncoderModule (WORD wOutputPulseWidth_usec,
WORD wComparator1On, DWORD dwComparator1Value,
WORD wComparator2On, DWORD dwComparator2Value,
WORD wReserved, DWORD dwReserved, WORD wReserved2,
DWORD dwReserved2, WORD wOptions)**

Description Defines the operation of the BLUE-EXPIO comparitors.

Parameters	<i>wOutputPulseWidth</i>	The output signal pulse width, in microseconds.
	<i>wComparator1On</i>	Enables Comparator 1 output. Set to 1 to enable, 0 (zero) to disable.
	<i>dwComparator1Value</i>	The encoder count period for the output of Comparator 1.
	<i>wComparator2On</i>	Enables Comparator 2 output. Set to 1 to enable, 0 (zero) to disable.
	<i>dwComparator2Value</i>	The encoder count period for the output of Comparator 2.
	<i>wReserved, dwReserved, wReserved2, dwReserved2</i>	Unused.
	<i>wOptions</i>	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 comparitors 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 *wOptions* bits 8 and 9, respectively.

The encoder signals may be either RS422 or TTL, as set by the *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 allow 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:

<i>wOptions Bit</i>	<i>Meaning</i>
9	Comparator 2 input select. 0 - Comparator 2 generated from Y input 1 - Comparator 2 generated from Z input
8	Comparator 1 input select. 0 - Comparator 1 generated from X input 1 - Comparator 1 generated from F input
7	F encoder input type. 0 - RS422 1 - TTL
6	Z encoder input type. 0 - RS422 1 - TTL
5	Y encoder input type. 0 - RS422 1 - TTL
4	X encoder input type. 0 - RS422 1 - TTL
[3..2]	Position latch function. 0 - XYZF position latch via DSP 1 - XYZF position latch via trigB input on BLUE-CONNECT PCB. 2 - XYZF position latch via trigA input on BLUE-CONNECT PCB.
[1..0]	General-purpose comparator function select. 0 - Comparator trigger on X axis. 1 - Comparator trigger on Y axis. 2 - Comparator trigger on Z axis.

See Also **OI_ReadEncoderModule**

Automatic Focus

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	<i>dRange</i>	The range of positions over which to search for focus.
	<i>nSamples</i>	The number of focus threshold measurements to average at each position.
	<i>dStepSize</i>	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	<p>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:</p> <ol style="list-style-type: none">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. <p>The <i>dStepSize</i> parameter specifies the distance to move between measurements of the focus score. The <i>nSamples</i> parameter allows focus score values to be averaged at each measurement, reducing the effects of noise.</p> <p>The searching is limited by the <i>dRange</i> parameter, preventing the focus operation from moving farther than a given distance.</p> <p>In some situations, this may produce more accurate results, although the overall time taken to achieve focus is usually increased.</p>
See Also	OI_WaitForAutoFocus, OI_AutoFocusEx, OI_AutoFocus

OI_AutoFocus_Step

Syntax	OI_API OI_AutoFocus_Step(double dRange, int nSpeed, int nTolerance, double dStepSize)	
Description	Performs a stepwise automatic focus.	
Parameters	<i>dRange</i>	The range of positions over which to search for focus.
	<i>nSpeed</i>	The cruise speed at which the Z axis is moved during the focus search.
	<i>nTolerance</i>	The peak-definition tolerance value.
	<i>dStepSize</i>	The Z step distance between discrete moves during the autofocus sweep.
Return	OI_OK if successful.	

Value	If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.
Comments	<p>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 <i>dStepSize</i> parameter.</p> <p>In some situations, this may produce more accurate results, although the overall time taken to achieve focus is usually increased.</p>
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	<i>dRange</i>	The range of positions over which to search for focus.
	<i>nSpeed</i>	The cruise speed at which the Z axis is moved during the focus search.
	<i>nTolerance</i>	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	<i>pdRange</i>	The returned range of travel for the autofocus, in microns.

	<i>pnSpeed</i>	The returned cruise speed at which the focus is moved.
	<i>pnTolerance</i>	The returned peak-finding tolerance value.
Return Value	<p>OI_OK if successful.</p> <p>If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.</p>	
Comments	See the Comments for the OI_AutoFocusEx for more information about the AutoFocus settings.	
See Also	OI_AutoFocusEx, OI_GetAutoFocusThreshold	

OI_GetAutoFocusEx

Syntax	OI_API OI_GetAutoFocusEx(double* pdRange, int* pnSpeed, int* pnTolerance, double *pdStepSize)	
Description	Retrieves the current AutoFocus extended settings.	
Parameters	<i>pdRange</i>	The range of positions over which to search for focus.
	<i>pnSpeed</i>	The cruise speed at which the Z axis is moved during the focus search.
	<i>pnTolerance</i>	The peak-definition tolerance value.
	<i>pdStepSize</i>	The step size in microns between samples.
Return Value	<p>OI_OK if successful.</p> <p>If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.</p>	
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	OI_API OI_GetAutoFocusThreshold(int* pnThresh)
---------------	---

Description	Retrieves the current AutoFocus threshold value.	
Parameters	<i>pnThresh</i>	The returned threshold value, ranging from 0 to 255.
Return Value	<p>OI_OK if successful.</p> <p>If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.</p>	
Comments	See the Comments for the OI_SetAutoFocusThreshold for more information about the AutoFocus threshold.	
See Also	OI_SetAutoFocusThreshold	

OI_GetFineFocusSamples

Syntax	OI_API OI_GetFineFocusSamples(int* pnSamples)	
Description	Retrieves the number of focus score samples to be taken at each step in the fine focus operation.	
Parameters	<i>nSamples</i>	The number of focus score samples.
Return Value	<p>OI_OK if successful.</p> <p>If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.</p>	
Comments	See the OI_SetFineFocusSamples function for a description of the focus samples.	
See Also	OI_SetFineFocusSamples, OI_AutoFocus_Fine	

OI_ReadFocusProfile

Syntax	OI_API OI_ReadFocusProfile(double* pdScores, double* pdZPos, int nSize, int* pnSamples)	
Description	Reads the AutoFocus profile from the last automatic focus operation.	
Parameters	<i>pdScores</i>	Pointer to an array, of length <i>nSize</i> , to receive the focus score values.
	<i>pdZPos</i>	Pointer to an array, of length <i>nSize</i> , to receive the Z position values.
	<i>nSize</i>	The length of the <i>pdScores</i> and <i>pdZPos</i> arrays.

	<i>pnSamples</i>	The returned actual number of samples.
Return Value	<p>OI_OK if successful.</p> <p>If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.</p>	
Comments	<p>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.</p> <p>The OI_ReadFocusProfile function will return these pairs of values in two arrays, one for the focus scores (returned in the <i>pdScores</i> array) and another for the corresponding Z positions (returned in the <i>pdZPos</i> array).</p> <p>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.</p> <p>The actual number of samples taken during the last AutoFocus operation is returned in the <i>pnSamples</i> value. The <i>nSize</i> 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 <i>nSize</i> value, only the first <i>nSize</i> values are returned in the arrays. If the total number of samples is less than the <i>nSize</i> value, only the first <i>pnSamples</i> values contain valid data.</p>	
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	<i>pdScore</i>	The returned focus score.
Return Value	<p>OI_OK if successful.</p> <p>If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.</p>	
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	<i>pdScore</i>	The returned focus score.
	<i>pdZPos</i>	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	<i>pwFinished</i>	The returned status indicating whether the previous autofocus is complete.
	<i>pwSuccess</i>	The returned success status of the last autofocus.
	<i>pwNumSamples</i>	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	OI_API OI_SetAutoFocus(double dRange, int nSpeed, int nTolerance)	
Description	Sets the default focus parameters.	
Parameters	<i>dRange</i>	The range of positions over which to search for focus.
	<i>nSpeed</i>	The cruise speed at which the Z axis is moved during the focus search.
	<i>nTolerance</i>	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	OI_API OI_SetAutoFocusEx(double dRange, int nSpeed, int nTolerance, double dStepSize)	
Description	Sets the default focus parameters, including step size.	
Parameters	<i>dRange</i>	The range of positions over which to search for focus.
	<i>nSpeed</i>	The cruise speed at which the Z axis is moved during the focus search.
	<i>nTolerance</i>	The peak-definition tolerance value.
	<i>dStepSize</i>	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.	

See Also **OI_GetAutoFocusEx, OI_AutoFocus, OI_SetAutoFocus**

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	<i>nThresh</i>	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	<i>nSamples</i>	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	OI_API OI_GetAutoPredictiveZ (LPBOOL pbFlag, LPBOOL pbValid, LPDOUBLE pdZ)	
Description	Returns the status of the automatic predictive focus tracking.	
Parameters	<i>pbFlag</i>	Flag indicating whether automatic predictive focus tracking is enabled or disabled.
	<i>pbValid</i>	Flag indicating whether the predictive focus setup is valid.
	<i>pdZ</i>	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	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.
	<i>pnDomainIndex</i>	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	OI_API OI_GetMultiPredictiveZ (int* pnMethod, int* pnPoints, double *xvals, double *yvals, double *zvals)	
Description	Returns the predictive focus information.	
Parameters	<i>pnMethod</i>	Returns the method to use to divide the sample area into regions.
	<i>pnPoints</i>	Returns the number of XYZ points used to define the predictive focus map. The maximum number of points supported is 256.
	<i>xvals</i>	Pointer to an array to receive the X-axis positions, in microns, representing the measurement X positions.
	<i>yvals</i>	Pointer to an array to receive the Y-axis positions, in microns, representing the measurement Y positions.
	<i>zvals</i>	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 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 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

reason for failure.

Comments 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.

<i>nStatus value</i>	<i>Meaning</i>
0	The predictive focus has been defined and is valid.
1	Invalid predictive focus coefficients. This typically indicates the input positions set via the OI_SetPredictiveZ function did not contain three unique XYZ locations.
2	The predictive focus has not been defined.

See Also **OI_SetPredictiveZ**

OI_GetPredictiveZDomains

Syntax **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	<i>pnDomains</i>	Returns the method to use to divide the sample area into regions.
	<i>ax, ay, az</i>	Pointers to arrays to receive the first corner data, i.e., vertex <i>a</i> .
	<i>bx, by, bz</i>	Pointers to arrays to receive the second corner data, i.e., vertex <i>b</i> .
	<i>cx, cy, cz</i>	Pointers to arrays to receive the third corner data, i.e., vertex <i>c</i> .

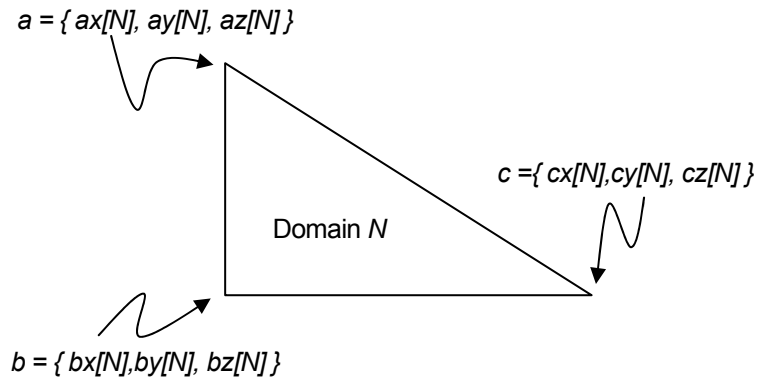
Return Value OI_OK if successful.

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	<i>bFlag</i> 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	<p>The OASIS controller can automatically perform movement of the Z-axis to ensure the Z-position is maintained to the predicted focus position based 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.</p> <p>The OI_SetAutoPredictiveZ function enables and disabled automatic predictive focus tracking.</p> <p>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.</p>

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	<i>nMethod</i>	The method to use to divide the sample area into regions. Use OI_PF_MULTI_PLANE.
	<i>nPoints</i>	The number of XYZ points used to define the predictive focus map, i.e., the length of arrays <i>xvals</i> , <i>yvals</i> , and <i>zvals</i> . The maximum number of points supported is 256.
	<i>xvals</i>	Pointer to array of X-axis positions, in microns, representing the measurement X positions.
	<i>yvals</i>	Pointer to array of Y-axis positions, in microns, representing the measurement Y positions.
	<i>zvals</i>	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	OI_API OI_SetPredictiveFlag (BOOL bFlag)	
Description	Sets the flag indicating whether the predictive focus has been previously defined.	
Parameters	<i>bFlag</i>	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	OI_API OI_SetPredictiveZ (double *pX, double *pY, double *pZ)	
Description	Sets the predictive focus locations.	
Parameters	<i>pX</i>	Pointer to array of three X-axis positions, in microns, representing the measurement locations X positions.
	<i>pY</i>	Pointer to array of three Y-axis positions, in microns, representing the measurement locations Y positions.
	<i>pZ</i>	Pointer to array of three Z-axis positions, in microns, representing the measurement

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 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 **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 NSTC 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	<i>pnXStart</i>	The start X value for the video window, in pixels.
	<i>pnXStop</i>	The stop X value for the video window, in pixels.
	<i>pnYStart</i>	The start Y value for the video window, in video field lines.
	<i>pnYStop</i>	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**

OI_IsVideoDetected

Syntax	OI_API OI_IsVideoDetected(BOOL* pbFound)	
Description	Reads whether a valid video signal has been detected.	
Parameters	<i>pbFound</i>	Pointer to BOOL result indicating if video has been detected.
Return Value	OI_OK if successful. If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.	
Comments	The <i>pbFound</i> 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 <i>pbFound</i> parameter will be set to FALSE.	
See Also	OI_GetAFFitted, OI_ReadPCBStatus	

OI_ReadVideoData

Syntax	OI_API OI_ReadVideoData(LPWORD lpwStatus, LPDWORD pdwArea, LPWORD pwMaxChord, LPWORD pwMaxGradient)	
Description	Reads the current video measurement data.	
Parameters	<i>lpwStatus</i>	The status of the video measurements, as

	described in the Comments below.								
<i>pdwArea</i>	The total count of pixels measured in the detected phase.								
<i>pwMaxChord</i>	The pixel length of the longest chord in the detected phase.								
<i>pwMaxGradient</i>	Not used.								
Return Value	<p>OI_OK if successful.</p> <p>If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.</p>								
Comments	<p>The <i>pnStatus</i> variable indicates the video field for which the current results are valid, and is one of the following values:</p> <table> <tr> <th><i>pnStatus Value</i></th><th><i>Meaning</i></th></tr> <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> </table>	<i>pnStatus Value</i>	<i>Meaning</i>	0	The video measurement data is not available.	1	The returned results are for the Odd video field.	2	The returned results are for the Even video field.
<i>pnStatus Value</i>	<i>Meaning</i>								
0	The video measurement data is not available.								
1	The returned results are for the Odd video field.								
2	The returned results are for the Even video field.								
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	<i>nMode</i>	Specifies the desired results, as described in the Comments below.
	<i>pdwArea</i>	The total count of pixels measured in the detected phase.
	<i>pwMaxChord</i>	The pixel length of the longest chord in the detected phase.
Return Value	<p>OI_OK if successful.</p> <p>OI_TIMEOUT if the desired results are not available within 200 msec.</p> <p>If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.</p>	

Comments The *nMode* parameter indicates which video field is to be read, using the following values:

<i>nMode Value</i>	<i>Meaning</i>
1	The function should return the Odd video field results.
2	The function should return the Even video field results.
3	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_ReadVideoResultsEx

Syntax `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

<i>nMode</i>	Specifies the desired results, as described in the Comments below.
<i>pdwArea</i>	The total count of pixels measured in the detected phase.
<i>pwMaxChord</i>	The pixel length of the longest chord in the detected phase.
<i>pwMaxGradient</i>	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:

<i>nMode Value</i>	<i>Meaning</i>
--------------------	----------------

1	The function should return the Odd video field results.
2	The function should return the Even video field results.
3	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 **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

<i>nMode</i>	Specifies the desired results, as described in the Comments below.
<i>pdResults</i>	An array of doubles into which the video results are to be copied.
<i>pdXPos</i>	The position of the X Axis, in microns.
<i>pdYPos</i>	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:

<i>nMode Value</i>	<i>Meaning</i>
1	The function should return the Odd video field results.
2	The function should return the Even video field

results.

- 3 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:

<i>pdResults</i> Array Index	Video Result Value
0	Focus score (full video frame)
1	Detected area
2	Detected maximum chord length
3	Detected maximum gradient

See Also `OI_SetVideoSettings`, `OI_ReadVideoData`, `OI_ReadVideoResults`, `OI_ReadVideoResultsEx`, `OI_ReadVideoResultsXYZF`, `OI_ReadVideoResultsZ`

OI_ReadVideoResultsXYZF

Syntax `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

<i>nMode</i>	Specifies the desired results, as described in the Comments below.
<i>pdResults</i>	Array of doubles into which the video results are to be copied.
<i>pdPositions</i>	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:

<i>nMode Value</i>	<i>Meaning</i>
1	The function should return the Odd video field results.
2	The function should return the Even video field results.
3	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 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:

<i>pdResults Array Index</i>	<i>Video Result Value</i>
0	Focus score (full video frame)
1	Detected area
2	Detected maximum chord length
3	Detected maximum gradient

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:

<i>pdPositions Array Index</i>	<i>Axis Position Value</i>
0	X Axis position
1	Y Axis position
2	Z Axis position
3	F Axis position

See Also **OI_SetVideoSettings, OI_ReadVideoData, OI_ReadVideoResults, OI_ReadVideoResultsEx, OI_ReadVideoResultsXY, OI_ReadVideoResultsZ**

OI_ReadVideoResultsZ

Syntax OI_API OI_ReadVideoResultsZ(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

<i>nMode</i>	Specifies the desired results, as described in the Comments below.
<i>pdResults</i>	An array of doubles into which the video results are to be copied.
<i>pdZPos</i>	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:

<i>nMode Value</i>	<i>Meaning</i>
1	The function should return the Odd video field results.
2	The function should return the Even video field results.
3	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:

<i>pdResults Array Index</i>	<i>Video Result Value</i>
0	Focus score (full video frame)
1	Detected area
2	Detected maximum chord length
3	Detected maximum gradient

See Also **OI_SetVideoSettings, OI_ReadVideoData, OI_ReadVideoResults, OI_ReadVideoResultsEx, OI_ReadVideoResultsXY, OI_ReadVideoResultsXYZF**

OI_SetVideoSettings

Syntax **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	<i>nEvenPhase</i>	Specifies the desired phase, either light or dark, for the Even video fields.
	<i>nEvenThreshold</i>	Specifies the desired threshold, from 0 to 255, for the Even video fields
	<i>nOddPhase</i>	Specifies the desired phase, either light or dark, for the Odd video fields.
	<i>nOddThreshold</i>	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:

<i>Phase Value</i>	<i>Meaning</i>
0	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.
1	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.

See Also **OI_ReadVideoData, OI_ReadVideoResults**

OI_SetVideoWindow

Syntax	OI_API OI_SetVideoWindow(int <i>nXStart</i> , int <i>nXStop</i> , int <i>nYStart</i> , int <i>nYStop</i>)	
Description	Sets the positions defining the video window placement.	
Parameters	<i>nXStart</i>	The start X value for the video window, in pixels.
	<i>nXStop</i>	The stop X value for the video window, in pixels.
	<i>nYStart</i>	The start Y value for the video window, in video field lines.
	<i>nYStop</i>	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 <i>nXStart</i> and <i>nXStop</i> values must be set on 4-pixel boundaries, e.g., 4, 8, 12, etc.	
	The <i>nYStart</i> and <i>nYStop</i> 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 <i>nYStart</i> video window Y start position is restricted to values from 0 to 254, i.e. roughly the top half of the video field.	
	The <i>nYStop</i> 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 Microsystems' DMR automated microscope fluorescence module. See the following table for the supported functions for the OASIS controller and the Leica Microsystems DMR microscope.

	OASIS	Leica DMR
Supported Functions	OI_SetFilterCount OI_GetFilterCount OI_DeleteFilter OI_MoveToFilter OI_GetFilterPosition OI_InitFilterChanger OI_SetFilterHomeOffset OI_GetFilterHomeOffset OI_WaitForStoppedFilter	OI_GetFilterCount OI_MoveToFilter OI_GetFilterPosition OI_GetFilterName OI_GetFilterDescription OI_SetShutter OI_GetShutter

	OI_SetFilterName OI_GetFilterName OI_SetFilterDescription OI_GetFilterDescription OI_ReadFilterHomeInfo OI_ReadFilterChangerInfo OI_SetFilterOffset OI_GetFilterOffset OI_SetFilterLocation	
--	---	--

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	<i>nPosition</i> 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	<p>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.</p> <p>After a call to OI_DeleteFilter, the filter count will be reduced by one.</p>
See Also	OI_SetJoystickEnabled

OI_GetFilterChanger

Syntax	OI_API OI_GetFilterChanger(int* pnChanger)	
Description	Retrieves the active filter changer.	
Parameters	<i>pnChanger</i>	Pointer to receive the zero-based index of the currently active filter changer.
Return Value	<p>OI_OK if successful.</p> <p>If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.</p>	
Comments	<p>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.</p>	
See Also	OI_SetFilterChanger, OI_GetFilterChangerCount	

OI_GetFilterChangerCount

Syntax	OI_API OI_GetFilterChangerCount(int* pnChangers)	
Description	Retrieves the number of filter changers available.	
Parameters	<i>pnChangers</i>	Pointer to receive the value of the number of filter changes fitted.
Return Value	<p>OI_OK if successful.</p> <p>If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.</p>	
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	<code>OI_API OI_GetFilterCount(int *pnFilters)</code>		
Description	Retrieves the current number of filter positions in the filter changer.		
Parameters	<table><tr><td><i>pnFilters</i></td><td>The returned number of filters.</td></tr></table>	<i>pnFilters</i>	The returned number of filters.
<i>pnFilters</i>	The returned number of filters.		
Return Value	<p>OI_OK if successful.</p> <p>If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.</p>		
Comments	See the OI_SetFilterCount function for more information about the filter count.		
See Also	OI_SetFilterCount		

OI_GetFilterDescription

Syntax	<code>OI_API OI_GetFilterDescription(int nPosition, LPSTR szBuffer, int nBufferLen)</code>						
Description	Retrieves the current description for a given filter position.						
Parameters	<table><tr><td><i>nPosition</i></td><td>The one-based filter position.</td></tr><tr><td><i>szBuffer</i></td><td>The destination text buffer.</td></tr><tr><td><i>nBufferLen</i></td><td>The length of the destination buffer specified in the <i>szBuffer</i> parameter.</td></tr></table>	<i>nPosition</i>	The one-based filter position.	<i>szBuffer</i>	The destination text buffer.	<i>nBufferLen</i>	The length of the destination buffer specified in the <i>szBuffer</i> parameter.
<i>nPosition</i>	The one-based filter position.						
<i>szBuffer</i>	The destination text buffer.						
<i>nBufferLen</i>	The length of the destination buffer specified in the <i>szBuffer</i> parameter.						
Return Value	<p>OI_OK if successful.</p> <p>If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.</p>						
Comments	See the OI_SetFilterDescription function for more information about filter descriptions.						
See Also	OI_SetFilterDescription, OI_SetFilterName, OI_GetFilterName						

OI_GetFilterHomeOffset

Syntax	OI_API OI_GetFilterHomeOffset(double *pdOffset)	
Description	Retrieves the current offset, in calibrated units, between the home position and the first filter.	
Parameters	<i>pdOffset</i>	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	OI_API OI_GetFilterName(int nPosition, LPSTR szBuffer, int nBufferLen)	
Description	Retrieves the current name for a given filter position.	
Parameters	<i>nPosition</i>	The one-based filter position.
	<i>szBuffer</i>	The destination text buffer.
	<i>nBufferLen</i>	The length of the destination buffer specified in the <i>szBuffer</i> 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	OI_API OI_GetFilterOffset(int nPosition, double *pdOffset)
---------------	--

Description	Retrieves the offset from the home position to a given filter's position.	
Parameters	<i>nPosition</i>	The one-based filter position.
	<i>dOffset</i>	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	<i>pnPosition</i>	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	<i>lpdwMsecs</i>	Pointer to receive the current filter changer timeout, in milliseconds.

Return Value	<p>OI_OK if successful.</p> <p>If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.</p>
Comments	<p>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.</p> <p>The OI_GetFilterTimeout function allows you to find out how long these functions currently allow before a timeout occurs.</p>
See Also	OI_SetFilterTimeout

OI_GetAvailableShutterCount

Syntax	OI_API OI_GetAvailableShutterCount(int * pnTotalSupported)	
Description	Retrieves the maximum number of available shutters for the current configuration.	
Parameters	<i>pnTotalSupported</i>	The returned count of available shutters.
Return Value	<p>OI_OK if successful.</p> <p>If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.</p>	
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	OI_API OI_GetShutter(int *pnPosition, int nShutter)	
Description	Retrieves the current status of a shutter.	
Parameters	<i>pnPosition</i>	The returned position value.
	<i>nShutter</i>	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 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** 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:

<i>SHUTTERINFO Member</i>	<i>Meaning</i>
wDurationHV	Indicates the High Voltage pulse duration (20-100 msec)
wTimerHV	Indicates the current High Voltage timer value (20-100 msec)
wDurationLV	Indicates the Low (holding) Voltage duration (0 to 65535 msec)

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_GetShutterShutter** 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

pnStates value.

See Also **OI_SetShutterMulti**

OI_InitFilterChanger

Syntax **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	<i>nMethod</i>	The method to use for initialisation.
	<i>dUnitsPerRev</i>	The number of calibrated units per full filter changer revolution or travel. Note this value may be different 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 discrete 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.

<i>nMethod Value</i>	<i>Meaning</i>
OI_FILTER_INIT_HOME	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.

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 <i>dUnitsPerRev</i> 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	<i>nPosition</i>	The one-based index of the filter to move to.
	<i>nWait</i>	A non-zero <i>nWait</i> 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)
---------------	---

Description	Retrieves the current definition values of the filter wheel steps per revolution and filter spacing.	
Parameters	<i>pdUnitsPerRev</i>	The current value for calibrated units per filter revolution.
	<i>pdFilterSpacing</i>	The calculated distance between each filter, 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	When the filter changer is initialised, the calibrated distance per revolution is measured (or specified, depending of the type of initialisation). This value is used in conjunction with the current number of filters to calculate the spacing between filters.	
	The OI_ReadFilterChangerInfo allows you to retrieve the current values for these parameters. Use the OI_GetFilterCount to get the current number of filters.	
See Also	OI_GetFilterCount , OI_SetFilterCount , OI_InitFilterChanger	

OI_ReadFilterHomeInfo

Syntax	OI_API OI_ReadFilterHomeInfo(LPBOOL lpbHomeFound, double *pdHomeLeft, double *pdHomeRight)	
Description	Retrieves the current information regarding the home switch detection.	
Parameters	<i>lpbHomeFound</i>	Flag indicating if a home switch has been detected.
	<i>pdHomeLeft</i>	The leading edge position of the home switch, in calibrated units.
	<i>pdHomeRight</i>	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 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	OI_API OI_SetFilterDescription(int nPosition, LPSTR szBuffer)	
Description	Retrieves the status of joystick control for the X, Y, and Z axes.	
Parameters	<i>nPosition</i>	The one-based filter position.
	<i>szBuffer</i>	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	OI_API OI_SetFilterHomeOffset(double dOffset)	
Description	Sets the offset, in calibrated units, between the home positions and the first filter position.	
Parameters	<i>dOffset</i>	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.	

See Also **OI_GetFilterHomeOffset, OI_InitFilterChanger**

OI_SetFilterLocation

Syntax	OI_API OI_SetFilterLocation(int nPosition)	
Description	Sets the specified filter to the current offset.	
Parameters	<i>nPosition</i>	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	<i>nPosition</i>	The one-based filter position.
	<i>szBuffer</i>	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	<i>nPosition</i>	The one-based filter position.
	<i>dOffset</i>	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	<i>dwMSecs</i>	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	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><tr><th>nPosition Value</th><th>Meaning</th></tr><tr><td>1</td><td>Opens the shutter, allowing light to pass.</td></tr><tr><td>0</td><td>Closes the shutter.</td></tr></table>		nPosition Value	Meaning	1	Opens the shutter, allowing light to pass.	0
nPosition Value	Meaning						
1	Opens the shutter, allowing light to pass.						
0	Closes the shutter.						
See Also	OI_SetShutterMulti, OI_GetShutter, OI_GetShutterMulti						

OI_SetShutterEx

Syntax	OI_API OI_SetShutterEx(LPBOOL pbShutters, LPWORD pwValues)	
Description	Sets the state of a shutter.	
Parameters	<i>pbShutters</i>	Pointer to an array of shutters to be set.
	<i>pwValues</i>	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:

```
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:

<i>Shutter Value</i>	<i>Meaning</i>
0 or OI_SHUTTER_CLOSE	Closes the shutter.
1 to 65534	Open for specified time, in milliseconds
65535 or OI_SHUTTER_OPEN	Opens the shutter, leaving open indefinitely, until closed by a subsequent command.

See Also **OI_SetShutterMulti, OI_GetShutter, OI_GetShutterMulti**

OI_SetShutterMulti

Syntax OI_API OI_SetShutterMulti(int nStates, int nStateMask)

Description Sets the state of a shutter.

Parameters

<i>nStates</i>	A combination desired shutter positions, as described in the comments.
<i>nStateMask</i>	A mask defining which bits of the <i>nStates</i> 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.

<i>nPosition Value</i>	<i>Meaning</i>
1	Opens the shutter, allowing light to pass.
0	Closes the shutter.

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 <i>wMask</i>)	
Description	Clears trackball button pressed and released flags.	
Parameters	<i>wMask</i>	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	<p>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.</p> <p>The flags to be cleared are controlled by the <i>wMask</i> parameter, which should be a bitwise OR of the pressed and released flag values as defined in the comments of the OI_ReadTrackballStatus function.</p> <p>For instance, to clear only the button 1 pressed and released flags, use the following:</p> <pre>OI_ClearTrackballStatus(OI_BUTTON1_PRESSED OI_BUTTON1_RELEASED);</pre> <p>OI_ClearTrackballStatus may be used to clear all button flags by passing a value of OI_CLEAR_ALL_BUTTONS as the <i>wMask</i> parameter.</p>	
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	<i>pbXEnabled</i>	The returned joystick control status for the X axis.
	<i>pbYEnabled</i>	The returned joystick control status for the Y axis.
	<i>pbZEnabled</i>	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	OI_API OI_GetTrackballControl(LPWORD pwEnable)	
Description	Retrieves the current settings for default trackball button action enabling.	
Parameters	<i>pwEnable</i>	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 <i>pwEnable</i> 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	OI_API OI_GetTrackballControl(BOOL* pbEnabled)	
Description	Retrieves the current trackball enabled status.	
Parameters	<i>pwEnabled</i>	The returned trackball control status.

Return Value	<p>OI_OK if successful.</p> <p>If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.</p>
Comments	The OI_GetTrackballEnabled function retrieves the status of trackball control inputs. The <i>pbEnabled</i> 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	<i>pwStatus</i>	The return status indicating the pressed and released flags for each button.
Return Value	<p>OI_OK if successful.</p> <p>If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.</p>	
Comments	<p>The OASIS controller maintains a “pressed” and “released” flag for each trackball button. These flags are set whenever the corresponding trackball action takes place.</p> <p>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.</p> <p>The pressed and released flags remain set until they are cleared using OI_ClearTrackballStatus.</p>	

The test the returned status WORD, use the following values:

<i>Value</i>	<i>Meaning</i>
OI_BUTTON1_PRESSED	Button 1 (typically top-left) has been pressed.
OI_BUTTON1_RELEASED	Button 1 has been released.
OI_BUTTON2_PRESSED	Button 2 (typically top-right) has been pressed.
OI_BUTTON2_RELEASED	Button 2 has been released.

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:

```
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	OI_API OI_SetJoystickEnabled(BOOL bXEnabled, BOOL bYEnabled, BOOL bZEnabled)	
Description	Specifies which axes are enabled for joystick control.	
Parameters	<i>bXEnabled</i>	Enables or disables joystick control of the X axis.
	<i>bYEnabled</i>	Enables or disables joystick control of the Y axis.
	<i>bZEnabled</i>	Enables or disables joystick control of the Z axis.

Return Value	<p>OI_OK if successful.</p> <p>If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.</p>
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	<p><i>wEnable</i> Enabling WORD for trackball buttons.</p>
Return Value	<p>OI_OK if successful.</p> <p>If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.</p>
Comments	By default, the controller configures the trackball buttons for the following functions:

<i>Button</i>	<i>Action</i>
Top-left (button 1)	Enable / disable trackball control of XY stage
Top-right (button 2)	Enable / disable trackball control of Z focus
Bottom-left (button 3)	Perform autofocus using current settings
Bottom-right (button 4)	Cycle through 3 pre-defined autofocus range and speed settings. (These settings are configured in the OASIS flash memory.)

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:

<i>Value</i>	<i>Action</i>
OI_BUTTON1	Button 1, typically top-left

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:

```
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 **OI_ReadTrackballStatus** function.

See Also **OI_GetTrackballControl, OI_ReadTrackballStatus**

OI_SetTrackballEnabled

Syntax	OI_API OI_SetTrackballEnabled(BOOL bEnabled)	
Description	Enables and disables the function of the trackball.	
Parameters	<i>bEnabled</i>	Enables or disables the trackball.
Return Value	<p>OI_OK if successful.</p> <p>If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.</p>	
Comments	<p>A trackball may be fitted allowing X, Y, and/or Z control inputs. The OI_SetTrackballEnabled function is used to enable and disable all trackball control functions.</p>	
See Also	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 instance, 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	<i>lpdwMSecs</i>	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	<i>lpdwMSecs</i>	The returned move timeout, in milliseconds.

Return Value	<p>OI_OK if successful.</p> <p>If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.</p>
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	<i>lpdwMSecs</i>	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	<i>dwMSecs</i>	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	<i>dwMSecs</i>	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	<i>dwMSecs</i>	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	OI_API OI_LoadPositions (LPCTSTR sFile)	
Description	Loads the OASIS position and limit data from the specified file.	
Parameters	<i>sFile</i>	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	<p>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.</p> <p>If you pass an empty string in the <i>sFile</i> parameter, the settings will be loaded from the Windows Registry.</p> <p>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.</p> <p>The positions of the Negative and Positive User Limits are also read and set from the values found in the file.</p> <p>To save the positions, use the OI_SavePositions function.</p>	
See Also	OI_SavePositions, OI_LoadSettings, OI_SaveSettings	

OI_LoadSettings

Syntax	OI_API OI_LoadSettings(LPCTSTR sFile)	
Description	Loads the OASIS settings from the specified file.	
Parameters	<i>sFile</i>	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.

Comments 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	<p>OI_OK if successful.</p> <p>If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.</p>
Comments	<p>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.</p> <p>If you pass an empty string in the <i>sFile</i> parameter, the settings will be stored to the Windows Registry.</p> <p>The positions of the Negative and Positive User Limits are also stored in the file.</p> <p>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.</p>
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	<i>sFile</i>	The name of the file to which the settings are to be stored.
Return Value	<p>OI_OK if successful.</p> <p>If unsuccessful, a combination of error codes may be returned to indicate the reason for failure.</p>	
Comments	<p>The OI_SaveSettings function stores the OASIS settings to a text file. The file is structured like a standard Windows INI file.</p> <p>If you pass an empty string in the <i>sFile</i> parameter, the settings will be stored to the Windows Registry.</p> <p>To load the settings saved by OI_SaveSettings, use the OI_LoadSettings function.</p>	
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	<i>sFile</i>	The name of the file to which the settings are to be stored.
	<i>nComponent</i>	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 <i>sFile</i> 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	<i>pnCmd</i>	Returns the command code associated with the error.
	<i>pnRetCode</i>	Returns the return value from the function where the error occurred.
	<i>szDesc</i>	Returns an extended description of the error.
	<i>nLen</i>	Specifies the maximum length of the <i>szDesc</i> 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

Syntax OI_API OI_EnableMsgReportDlg(BOOL bEnabled)

Description Enables or disables message box reporting of general exception handling.

Parameters *bEnabled* Flag enabling or disabling general exception messages.

Return Value This function always returns OI_OK.

Comments Whenever a general exception occurs and is handled in an OASIS DLL function, a general error message box is normally displayed. The **OI_EnableMsgReportDlg** function is used to programmatically enable or disable the display of these message dialogs.

See Also **OI_GetLastError**

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	<i>dMicronVal</i>	The position value, in microns, to be converted.
	<i>lpdwSteps</i>	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_MicronsToStepY , OI_MicronsToStepsZ , OI_MicronsToStepsZ	

OI_MicronsToStepsX

OI_MicronsToStepsY

OI_MicronsToStepsZ

OI_MicronsToStepsF

Syntax	OI_API OI_MicronsToStepsX(double dMicronVal, long* lpdwSteps) OI_API OI_MicronsToStepsY(double dMicronVal, long* lpdwSteps) OI_API OI_MicronsToStepsZ(double dMicronVal, long* lpdwSteps) OI_API OI_MicronsToStepsF(double dMicronVal, long* lpdwSteps)	
Description	Converts a given micron distance into the corresponding microstep distance.	
Parameters	<i>dMicronVal</i>	The distance, in microns, to be converted.
	<i>lpdwSteps</i>	The returned distance 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_MicronsToAbsoluteX , OI_MicronsToAbsoluteY , OI_MicronsToAbsoluteZ , OI_MicronsToAbsoluteF	

OI_StepsToMicronsX

OI_StepsToMicronsY

OI_StepsToMicronsZ

OI_StepsToMicronsF

Syntax	OI_API OI_StepsToMicronsX(long lSteps, double* pdMicrons) OI_API OI_StepsToMicronsY(long lSteps, double* pdMicrons) OI_API OI_StepsToMicronsZ(long lSteps, double* pdMicrons) OI_API OI_StepsToMicronsF(long lSteps, double* pdMicrons)	
Description	Converts a given micron distance into the corresponding microstep distance.	
Parameters	<i>lSteps</i>	The distance, in microsteps, to be converted.
	<i>pdMicrons</i>	The returned distance 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 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_MicronsToStepsStepsY , 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:

<i>Port</i>	<i>Function</i>	<i>Location</i>	<i>Comments</i>
CTRL1	TTL Input / Output	Main 44-way connector, pin 19	
CTRL2	TLL Input / Output	Main 44-way connector, pin 23	
Open Collector 1	General output control, up to 12V	Main 44-way connector, pin 37	Includes a 100 ohm current limiting resistor.
Open Collector 2	General output control, up to 12V, 100 mA max	Main 44-way connector, pin 41	No current limiting resistor.
INPUT0	General TTL-compatible input	Connector PL4, pin 1	3.3V and 5V compatible input
INPUT1	General TTL-compatible input	Connector PL4, pin 3	3.3V and 5V compatible input
INPUT2	General TTL-compatible input	Connector PL4, pin 5	3.3V and 5V compatible input
INPUT3	General TTL-compatible input	Connector PL4, pin 7	3.3V and 5V compatible input

OI_ReadInputPorts

Syntax 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:

<i>Bit</i>	<i>Meaning</i>
0	INPUT0 – pin 1 of PL4 connector
1	INPUT1 – pin 3 of PL4 connector
2	INPUT2 – pin 5 of PL4 connector
3	INPUT3 – pin 7 of PL4 connector
4 – 7	Not used.

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:

<i>Bit</i>	<i>Meaning</i>
0	Open Collector 1. A set bit means the transistor is on.
1	Open Collector 2. A set bit means the transistor is on.
2	CTRL1. A set bit means the TTL logic is high.
3	CTRL2. A set bit means the TTL logic is high.
4 – 7	Not used.

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:

<i>Bit</i>	<i>Meaning</i>
0	Open Collector 1. A set bit means the transistor should be turned on.
1	Open Collector 2. A set bit means the transistor should be turned on.
2	CTRL1. A set bit means the TLL logic should be set to high.
3	CTRL2. A set bit means the TTL logic should be set to high.
4 – 7	Not used.

See Also OI_ReadIO, OI_ReadInputPorts