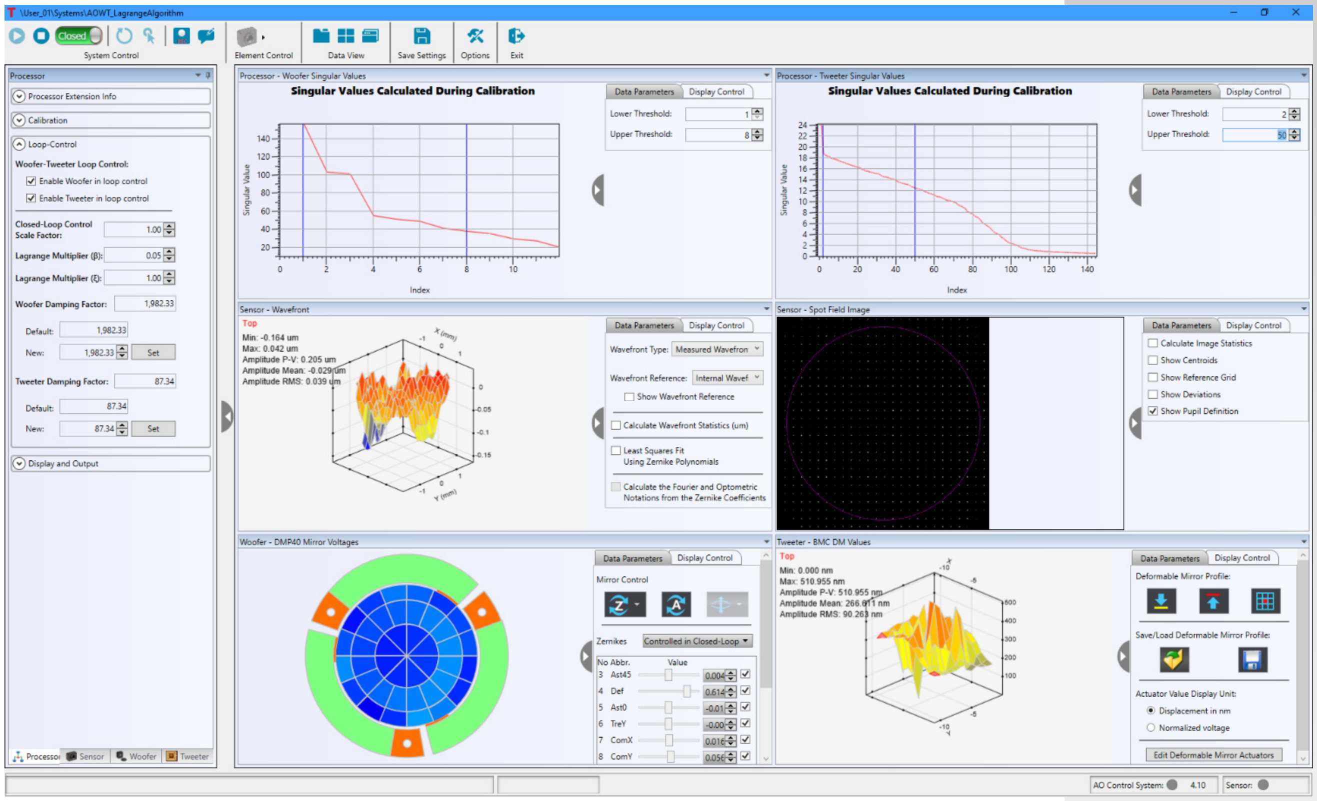
THORLABS

Adaptive Optics Kit

Programmer’s API Guide


\User_01\Systems\AOWT_LagrangeAlgorithm
Closed
System Control
Element Control
Data View
Save Settings
Options
Exit
Processor
Processor Extension Info
Calibration
Loop-Control
Display and Output
Processor
Sensor
Woofer
Tweeter
Processor - Woofer Singular Values
Singular Values Calculated During Calibration
Singular Value
Index
Processor - Tweeter Singular Values
Singular Values Calculated During Calibration
Sensor - Wavefront
Top
Sensor - Spot Field Image
Woofer - DMP40 Mirror Voltages
Tweeter - BMC DM Values
Top
Data Parameters
Display Control
Mirror Control
Deformable Mirror Profile:
Save/Load Deformable Mirror Profile:
Edit Deformable Mirror Actuators
AO Control System:
Sensor:

Woofer-Tweeter Loop Control:
- ☑ Enable Woofer in loop control
- ☑ Enable Tweeter in loop control

Closed-Loop Control Scale Factor:	1.00
Lagrange Multiplier (β):	0.05
Lagrange Multiplier (ξ):	1.00
Woofer Damping Factor:	1,982.33
Default:	1,982.33
New:	1,982.33
Tweeter Damping Factor:	87.34
Default:	87.34
New:	87.34

Woofer singular values: Lower Threshold: 1; Upper Threshold: 8

Tweeter singular values: Lower Threshold: 2; Upper Threshold: 50

Min: -0.164 um
Max: 0.042 um
Amplitude P-V: 0.205 um
Amplitude Mean: -0.029 um
Amplitude RMS: 0.039 um

Wavefront Type: Measured Wavefron
Wavefront Reference: Internal Wavef
- ☐ Show Wavefront Reference
- ☐ Calculate Wavefront Statistics (um)
- ☐ Least Squares Fit Using Zernike Polynomials
- ☐ Calculate the Fourier and Optometric Notations from the Zernike Coefficients

- ☐ Calculate Image Statistics
- ☐ Show Centroids
- ☐ Show Reference Grid
- ☐ Show Deviations
- ☑ Show Pupil Definition

Zernikes: Controlled in Closed-Loop

No	Abbr.	Value
3	Ast45	0.004
4	Def	0.614
5	Ast0	-0.01
6	TreY	-0.00
7	ComX	0.016
8	ComY	0.056

Min: 0.000 nm
Max: 510.955 nm
Amplitude P-V: 510.955 nm
Amplitude Mean: 266.611 nm
Amplitude RMS: 90.263 nm

Actuator Value Display Unit:
- ◉ Displacement in nm
- ○ Normalized voltage

AO Control System: 4.10

Table of Contents

Chapter 1 Overview

The AO Kit program comes with a Software Development Kit (SDK), which will allow users to develop their own applications. The key feature of the new AO Kit program and the SDK is plugin based code design (see Figure 1). Following the pre-defined interfaces and protocols, the Sensor and Controller plugins integrate the functions from the device Manufacturer driver and are able to fully control the physical devices. The Processor plugin implements the AO control procedures and algorithms. To add more flexibility, the plugin can load an extension to expand the key functionalities. AO Kit SDK manages the plugins. It loads the plugins and returns the plugin handles to the user. Through the handles and API functions, the user can access and control the devices and algorithms. This design allows new devices or algorithms to be added to the AO Kit without risk of destabilizing the core functionality.

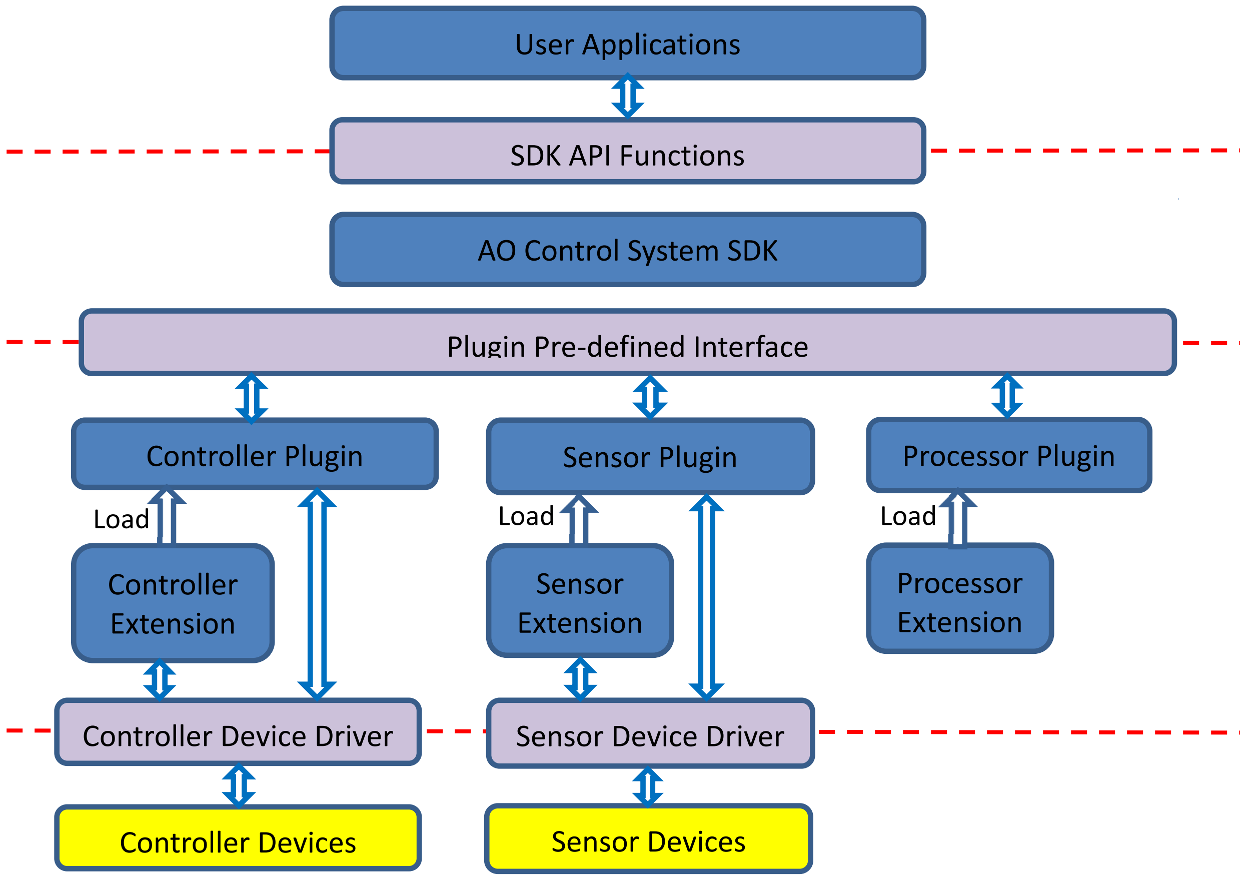


Figure 1 AO Kit Software Architecture

1.1. Plugins

There are three types of plugin modules developed for the AO Kit: Sensor, Controller and Processor. The Sensor plugin and Control plugin use the manufacturer device drivers to control the physical devices. The Processor plugin provides control procedures and algorithms.

1.1.1. Sensor Plugin

The sensor plugin module includes the functions to find, manage, and control sensor devices. By default the Thorlabs AO control system is delivered with the Thorlabs Wavefront Sensor Plugin Module (TL_WFS_Plugin.dll). This plugin works with all Thorlabs wavefront sensors.

1.1.2. Controller Plugin

The controller plugin module includes the functions to find, manage and control controller devices. By default the Thorlabs AO control system is delivered with the Thorlabs Piezo-based Deformable Mirror Plugin Module (TL_DM_Plugin.dll) and the BMC Mini/Multi Deformable Mirror Plugin Module (BMC_DM_Plugin.dll). The Thorlabs Piezo-based Deformable Mirror Plugin Module works with all Thorlabs DMP40 deformable mirrors, and the BMC Mini/Multi Deformable Mirror Plugin Module works with all Boston Micromachines' Mini- and Multi-Deformable Mirrors.

1.1.3. Processor Plugin

Processor plugin module is software DLL that includes functions to initialize, calibrate and execute the open/closed loop control of the Thorlabs AO control system. By default the Thorlabs AO control system is delivered with a Standard Processor Plugin Module (Standard_Processor_Plugin.dll).

1.2. Extension Module

Extensions are the software modules (.DLL) to expand the key functionalities of the base plugin modules.

No sensor extensions and controller extensions are developed for the AO Kit. The processor plugin must load a processor extension to implement different types of AO control. Depending on the processor extensions, the same functionality may have different implementations. For example, the implementation of calibration method, the AO Kit DMP40 Processor Extension calibrates the system by measuring the properties of the optical setup between the Deformable Mirror and the Wavefront Sensor, while the AO Kit Standard Processor Extension calibrates the system with “Poke” method.

Thorlabs provides five processor extensions; each implements a different type of AO control:

1.2.1. AO Kit DMP40 Processor Extension Module

This extension (AOKit_DMP40_Extension.dll) implemented in our AOK7 and AOK9 AO Kits, to control one Thorlabs DMP40 DM and one Thorlabs WFS. It works for the AO control system, which only includes Thorlabs devices.

1.2.2. AO Kit Standard Processor Extension Module

This extension (AOKit_Standard_Extension.dll) implemented in our AOK1 and AOK5 AO Kits, to control one deformable mirror and one wavefront sensor. In addition to the zonal poke-calibration method, implemented in the AOK1 and AOK5 AO Kits, it also includes a modal (Zernike) poke-calibration method. This process allows one to set min/max values for control signals during the calibration process, as well as a closed loop control correction factor.

1.2.3. AO Kit Woofer-Tweeter Lagrange-based Processor Extension Module

This extension (AOKit_WooferTweeter_LagrangeBased_Extension.dll) used in a Woofer-Tweeter AO control system; which implements a Lagrange-multiplier (LM)-based damped least-squares (DLS) control algorithm.

1.2.4. AO Kit Woofer-Tweeter Orthogonal-based Processor Extension Module

This extension (AOKit_WooferTweeter_OrthogonalBased_Extension.dll) implementes a decoupling control algorithm for Woofer-Tweeter control system.

1.2.5. AO Kit Woofer-Tweeter Standard Processor Extension Module

This extension (AOKit_WooferTweeter_Standard_Extension.dll) is designed to work with a Woofer-Tweeter control system with two deformable mirrors and one wavefront sensor. In a typical woofer tweeter system, the woofer deformable mirror would have fewer actuators with larger stroke compared with the tweeter deformable mirror. This is the case when using a Thorlabs DMP40 family deformable mirror as a woofer and the BMC Multi deformable mirror as a tweeter. Although this is the preferred configuration, and one that will yield better results, this extension module will allow a user to use either deformable mirror as the woofer and or tweeter.

1.3. AO Kit SDK

The AO Kit SDK is comprised of all the C API functions that developers need to access and control the Processor, Sensor and Controller plugins. This document is written for users requiring access to the available functions.

The following table details the AO Kit that use this software package.

Item #	Description
Current AO Kits Supported	
AOK5(/M)	Adaptive Optics Kit with Silver Coated 140 Actuator DM and Fast Wavefront Sensor
AOK8(/M)	Adaptive Optics Kit with Silver Coated High-Stroke 40-Actuator Piezo DM
AOKWT1(/M)	Adaptive Optics Kit with High-Stroke 40-Actuator Piezo DM, 140 Actuator DM and Fast Wavefront Sensor
Obsoleted AO Kits Still Supported by the Software:	
AOK1-UM01	Adaptive Optics Toolkit with Gold Coated 140 Actuator DM
AOK1-UP01	Adaptive Optics Toolkit with Aluminum 140 Actuator DM
AOK5-UM01	Adaptive Optics Kit with Gold Coated 140 Actuator DM and Fast Wavefront Sensor
AOK5-UP01	Adaptive Optics Kit with Aluminum Coated 140 Actuator DM and Fast Wavefront Sensor
AOK7-P01	Adaptive Optics Kit with Silver Coated 40-Actuator Piezo DM
AOK9-P01	Adaptive Optics Kit with Silver Coated 40-Actuator Piezo DM and Fast Wavefront Sensor

1.3.1. Software Requirements

- Windows 7, 64-bit or later
- Microsoft Visual Studio 2012 or later

Note: The SDK shipped with the AO Kit is 64-bit. Please contact Thorlabs customer service for the information of 32-bit SDK.

1.3.2. Installation

The following files are required to integrate the SDK functions should be found in
C:\Program Files\Thorlabs\ThorAOControl\SDK

- Bin\TL_ControlSystemSDK.dll
- Bin\TripleBufferLib_64.dll
- Inc\TL_ControlSystemSDK.h
- Inc\Tl_ControlSystemSDK_Def.h
- Lib_x64\TLControlSystemSDK.lib

1.3.3. Demo Program files

A demo program, written in Visual Studio 2017 C++, should be found in the following folder: C:\Program Files\Thorlabs\ThorAOControl\Examples\ThorAOControlSDK_WinConsole_Demo

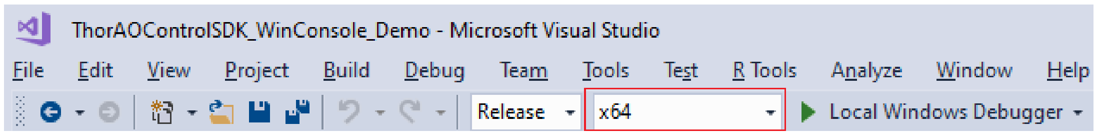

ThorAOControlSDK_WinConsole_Demo - Microsoft Visual Studio
File Edit View Project Build Debug Team Tools Test R Tools Analyze Window Help
Release
x64
Local Windows Debugger

Figure 2 Demo Program Platform Configuration

Please note the project is setup to:

1) Set the application to 64-bit;

2) Use namespace `TLControlSystem;`

3) Use the files in C:\Program Files\Thorlabs\ThorAOControl\SDK,

   (a) The include directory is set to *$(ProgramW6432)\Thorlabs\ThorAOControl\SDK\Inc\*
   (b) The library directory is set to *$(ProgramW6432)\Thorlabs\ThorAOControl\SDK\Lib_x64*

4) Copy the dll files from *$(ProgramW6432)\Thorlabs\ThorAOControl\SDK\Bin* into the output directory of the project via a post-build event setup in the project properties

Since the nature of the .NET environment lends itself to being able to swap between different languages easily, there should be only minor changes involved to use the TL_ControlSystemSDK.dll with VB.NET or C#.NET.

Chapter 2 XML Configuration File for Plugin

Each plugin has a default configuration file in XML format. This configuration file contains information such as Plugin module information, parameter list, output list, API function list, and other internal settings. The configuration file for the plugin should be placed in:

C:\Program Files\Thorlabs\ThorAOControl\Plugins\*[Plugin_Name]*\Plugin_Module\Config\.

When the SDK loads the plugin dll, the plugin dll expects to find the configuration file from above path and read the information from it.

With a plugin-based design, the SDK uses a common interface for all the plugins. To communicate with individual plugin, the SDK needs the plugin-specified parameter list, output list and other settings. It is very important for the developers to understand the XML configuration file and get all the critical information before using the SDK.

In this chapter, the configuration file of TL_WFS_Plugin is used as an example. All the definitions apply to the configuration files of other plugins.

2.1. Configuration Header

```
<Configuration_Header>
        <Configuration_Version>
                <MajorVersion>3</MajorVersion>
                <MinorVersion>0</MinorVersion>
        </Configuration_Version>
        <Plugin>
                <Name>TL_WFS_Plugin.dll</Name>
                <MajorVersion>3</MajorVersion>
                <MinorVersion>0</MinorVersion>
        </Plugin>
        <SupportedDevices>
                <Device0>WFS150</Device0>
                <Device1>WFS10</Device1>
                <Device2>WFS20</Device2>
                <Device3>WFS30</Device3>
                <Device4>WFS40</Device4>
        </SupportedDevices>
</Configuration_Header>
```

One important information in this section is the list of the device supported by this plugin. If the device is not in the list, the plugin will not create a handle for this device, therefore, the AO Kit SDK will not be able to use this device. Please contact Thorlabs customer service if your device is not in the list.

2.2. Plugin Control Parameters

This section provides the parameter list. Here are some examples,

```
<Plugin_Control_Parameters>
        Parameters for Thorlabs Wavefront Sensor Plugin
        <parameter>
                <FieldName>CamResolIndex</FieldName>
                <DataType>integer</DataType>
                <Description>Selects the camera resolution in pixels. This parameter is associated with function
                              WFS_ConfigureCam</Description>
                <Default>2</Default>
        </parameter>
        <parameter>
                <FieldName>SpotsX</FieldName>
                <DataType>integer</DataType>
                <Description>Retuns the number of spots which can be detected in Y direction, based on the
                                selected camera resolution and Microlens Array in function SetMlaIdx. This
                                parameter is associated with function WFS_ConfigureCam</Description>
                <Channel>out</Channel>
        </parameter>
        <parameter>
                <FieldName>PupilCenterX</FieldName>
                <DataType>double</DataType>
                <Description>Defines the pupil center X position in mm. This parameter is associated with
                               function WFS_GetPupil/WFS_SetPupil</Description>
                <Unit>mm</Unit>
                <Default>0</Default>
        </parameter>
        <parameter>
                <FieldName>ArrayZernikePolynomials</FieldName>
                <DataType>DataObject</DataType>
                <Description>Returns the one-dimension DataObject, whose internal single floating-point array
                                holds the Zernike polynomials used to simulate wavefront.</Description>
        </parameter>
</Plugin_Control_Parameters>
```

The plugin parameter is defined in the element with start-tag *<parameter>* and end-tag *</parameter>*. The element <parameter> has a few attributes:

- **FieldName**: specifies the name of the parameter. User must pass the exact name to the SDK functions.
- **DataType**: specifies the data type of the parameter. The valid data type includes boolean, integer, double, enumeration, string. A DataObject is used for array data. Please refer Chapter 3 on page 10 for more information about DataOjbect.
- **Description**: describes the meaning and usage of the parameter
- **Unit**: specifies the unit of the parameter
- **Channel**: value*Out* means the parameter is “get” only. If not specified, the parameter can be “set” and “get”
- **Default**: the default value of the parameter

2.3. Plugin Outputs

This section provides the plugin output list. The output could be the result from a function call or a processing taken by the plugin. The output is defined in an element begins with a start-tag *<output>* and ends with an end-tag *</output>*.

```
<Plugin_Outputs>
        Outputs from the Thorlabs Wavefront Sensor Plugin.
        <output>
                <FieldName>BeamCentroidX</FieldName>
                <DataType>double</DataType>
                <Description>Returns the beam centroid X in mm. This parameter is associated with function
                                WFS_CalcBeamCentroidDia</Description>
                <Unit>mm</Unit>
        </output>
        <output>
                <FieldName>ArrayWavefront</FieldName>
                <DataType>DataObject</DataType>
                <Description>Returns a two-dimensional DataObject, whose internal single floating-point array
                                holds the wavefront data in um. This parameter is associated with function
                                WFS_CalcWavefront/WFS_ConvertWavefrontWaves</Description>
        </output>
</Plugin_Outputs>
```

<output> element has some attributes:

- **FieldName**: specifies the name of the output. User must pass the exact name to the SDK functions.
- **DataType**: specifies the data type of the output. The valid data type includes integer, double, string. A DataObject is used for array data.
- **Description**: describes the meaning of the output
- **Unit**: specifies the unit of the output

2.4. Plugin Signals

This section defines the signal used in the plugin. The signal is the data used in the closed-loop control. For example, in a AO Kit control system, the spot deviations measured in the Thorlabs wavefront sensor is used to calculate the feedback signal for the deformable mirror, the data spot deviations is defined as Signal for the wavefront sensor.

```
<Plugin_Signals>
        Signals used in the closed-loop control.
        <parameter>
                <FieldName>Signals</FieldName>
                <DataType>enumeration</DataType>
                <Description>Available signals of the wavefront sensor.</Description>
                <Elements>
                        <Element ID="0" Value="SignalSpotDeviations"/>
                </Elements>
        </parameter>
        <signal>
                <FieldName>SignalSpotDeviations</FieldName>
                <DataType>SignalObject</DataType>
                <Description>A two-dimensional SignalObject, whose internal single floating-point array holds
                              the deviation in X and Y direction between centroid and reference spot positions in
                             pixels calculated by function CalcSpotToReferenceDeviations.</Description>
        </signal>
        <parameter>
                <FieldName>SignalSel</FieldName>
                <DataType>integer</DataType>
                <Description>Sets/Gets the selected signal.</Description>
                <Channel>inout</Channel>
                <Default>0</Default>
        </parameter>
</Plugin_Signals>
```

In this section, Parameter element *Signals* is enumeration data, it defines the available signals. Parameter element *SignalSel* is used to set the signal selection. Signal element *SignalSpotDeviations* is a SignalObject, please refer SDK function `GetElementSignalObjectHandle` for the use of SignalObject.

2.4.1. Plugin Commands

This section provides the plugin command list. The command is used to instruct the plugin to take an action. Please refer to SDK function `ExecuteCommand` for the use of plugin command.

```
<Plugin_Commands>
        Commands for the Plugin.
        <command>
                <FieldName>CaptureWavefront</FieldName>
                <Description>Captures a wavefront and saves it as reference</Description>
        </command>
```

```
	<command>
		<FieldName>CalculateZernikeWavefront</FieldName>
		<Description>Calculates the zernike wavefront and saves it as reference.</Description>
	</command>
	<command>
		<FieldName>Execute_DoSphericalRef</FieldName>
		<Description>Calculates User Reference spot positions based on an already performed
				measurement of a pure sherical wavefront. This command is associated with
				function WFS_DoSphericalRef</Description>
	</command>
</Plugin_Commands>
```

2.4.2. Factory APIs

This section provides the list of the API functions from the manufacturer device driver (Not API function from AO Kit SDK). The AO Kit SDK provides the capability for the developer to use the functions in the manufacturer device driver directly. Please refer `ExecuteApiFunction` for more information.

For example, function `WFS_CalcWavefront` is defined in the Thorlabs Wavefront Sensor driver

```
int WFS_CalcWavefront (WFS_hdl, float arrayWavefront[], int wavefrontType, int
limitToPupil);
```

It has two input parameters, *`wavefrontType`* and *`limitToPupil`*, and one output parameter *`arrayWavefront`*.

In the Factory APIs section, an <*API_Function*> element is created for the device driver API function. It uses the same function name *WFS_CalcWavefront* for the attribute <*FunctionName*>. Attribute <*InputParameters*> includes two input parameters *WavefrontTypeSel* and *LimitToPupil*. The details of both parameters can be found in <Plugin Parameters> section. Attribute <*OutputParameters*> defines the output parameter *ArrayWavefront*, which can found in <Plugin Outputs> section.

```
<Factory_APIs>
	API functions provided by the device driver.
	<API_Function>
		<FunctionName>WFS_CalcWavefront</FunctionName>
		<InputParameters>
			<param>WavefrontTypeSel</param>
			<param>LimitToPupil</param>
		</InputParameters>
		<OutputParameters>
			<param>ArrayWavefront</param>
		</OutputParameters>
	</API_Function>
</Factory_APIs>

</Plugin_Control_Parameters>
	<parameter>

		<FieldName>WavefrontTypeSel</FieldName>

		<DataType>integer</DataType>

		<Description>Defines the selected type of wavefront to calculate. This parameter is associated

				with function WFS_CalcWavefront</Description>

		<Default>0</Default>
```

```
          </parameter>

          <parameter>
                    <FieldName>LimitToPupil</FieldName>
                    <DataType>integer</DataType>
                    <Description>Defines if the Wavefront should be calculated based on all detected spots or only
                                     within the defined pupil. This parameter is associated with function
                                     WFS_CalcWavefront</Description>
                    <Default>0</Default>
          </parameter>
</Plugin_Control_Parameters>

<Plugin_Outputs>
          <output>
                    <FieldName>ArrayWavefrontWaves</FieldName>
                    <DataType>DataObject</DataType>
                    <Description>Returns a two-dimensional DataObject, whose internal single-point array holds the
                                     wavefront data in waves. This parameter is associated with function
                                     WFS_ConvertWavefrontWaves</Description>
          </output>
</Plugin_Outputs>
```

2.4.3. Plugin Parameter Initial Setting

This section provides initial value for some plugin parameters.

Chapter 3 Data and File Management

3.1. Data Management

This plugin uses DataObject to manage internal memory. DataObject is based on an abstract class defined below:

```
typedef enum _DataType
       {
              Unknown = -1,
              UChar,
              Char,
              UShort,
              Short,
              UInt32,
              Int32,
              Float,
              Double,
       };


typedef enum _DataPropertyInt
       {
              Data_Type,                  // Type of the data.
                                          // Available types are:Unknown, UChar, Char,
                                             UShort, Short, UInt32, Int32, Float, Double.
                                          // Corresponding values are from -1 to 7.
                                             Unknown is represented by -1.
              Data_Dimensions,            // Dimension of the dataset.
                                             Usually 1, 2 or 3. 0 indicates empty data.
              Data_Size1,                 // Size of the first dimension.
              Data_Size2,                 // Size of the second dimension.
              Data_Size3,                 // Size of the third dimension.
              Data_Size4,                 // Size of the fourth dimension.
              Data_TotalBytes,            // Number of bytes of the dataset.
              Data_BytesPerComponent,     // Number of bytes of a component.
              Data_ComponentsPerData,     // Number of components of a single data
       };

class AbstractDataObject
{
public:
       virtual ~AbstractDataObject() {}

       virtual int get_data_property_int(_DataPropertyInt prop) = 0;
       virtual void get_data_property_string(_DataPropertyString prop,
                                             char *str,
                                             int *strBufferLength) = 0;
       virtual void set_data_property_string(_DataPropertyString prop, const char *strValue) =
0;

       virtual void calculate_data_statistics() = 0;
       virtual double get_data_statistics(_DataStatistics dataStatistcs) = 0;

       virtual void set_data_type(_DataType data_type) = 0;

       // for example, a complex data has two components (x, y)
       virtual void set_components_per_data(int components_per_data) = 0;
```

```
        virtual void resize(int size1, int size2, int size3, int size4) = 0;

        virtual void* data_ptr() = 0;

        // Pass the the property structure from plugin to data object by void pointer.
        // Type cast is needed when returns the property structure
        virtual void set_property_structure(void *value) = 0;
        virtual void *get_property_structure() = 0;

        virtual void lock() = 0;
        virtual void unLock() = 0;
        virtual void set_data_content(void *pSource) = 0;
        virtual void get_data_content(void *pTarget) = 0;
};
```

The plugin creates DataObject for array parameter or array output. Memory is allocated inside the DataObject and operated by the DataObject's member methods. The End User Application gets the DataObject handle to access to the internal memory. Through the DataObject handle, the End User Application can transfer data from / to the plugin.

3.2. File Management

The AO Kit SDK introduces TLD (Thorlabs Data) file format to manage the file. TLD is not a plain binary data file; it comprises a combination of XML and binary data files that are contained inside a ZIP archive. The contents of a TLD file can be viewed by unzipping its contens.

- XML MetaData Files - contains information about other files available in the archive and control parameters
- Binary - contains the data from the plugins

The data in the TLD file can be exported in various formats, such as BINARY, TXT or CSV. If the data is image, it can be exported to PNG, BMP, JPEG, or TIFF.

Chapter 4 AO Kit API Functions

4.1. General Functions

4.1.1. GetError()

Return error message.

Declaration

```
bool GetError(char* message, int stringSize);
```

Input Parameters

`int stringSize`	Size of the input `char` array

Output Parameters

`char* message`	`char` array returns the error message

Return Value

False if no error

True if error

Example Code

```
#define STRING_LENGTH_BUFFER_512    512
char errStr[STRING_LENGTH_BUFFER_512];
bool rtn = GetError(errStr, STRING_LENGTH_BUFFER_512);
```

4.1.2. GetSDKPropertyString()

Return specified SDK property.

Declaration

```
void GetSDKPropertyString(_SDKProperty propertySelection,
                          char *strBuffer,
                          int *bytesOfStrBuffer);
```

Input Parameters

`_SDKProperty propertySelection`	Sets SDK property. `enum _SDKProperty { SDK_MajorVersion, SDK_MinorVersion }`
`int *bytesOfStrBuffer`	Input value is the size of the `char` array `strBuffer`. Returned value is the size of the SDK property string.

Output Parameters

`char *strBuffer`	`char` array returns the SDK property string

Return Value

None

Example Code

```
#define STRING_LENGTH_BUFFER_512    512
char str[STRING_LENGTH_BUFFER_512];
int  strLen = STRING_LENGTH_BUFFER_512;
GetSDKPropertyString(_SDKProperty::SDK_MajorVersion, str, &strLen);
```

4.2. Plugin Manager

4.2.1. OpenPluginManager()

Open the plugin manager, search for plugins, and load them.

Declaration

```
void OpenPluginManager();
```

Input Parameters

None

Output Parameters

None

Return Value

None

4.2.2. GetControlSystemDirectory()

Get the installation directory of the AO Kit software.

Declaration

```
void GetControlSystemDirectory(char *strBuffer, int *bytesOfStrBuffer);
```

Input Parameters

`int *bytesOfStrBuffer`	Input value is the size of the `char` array `strBuffer`. Returned value is the size of the AO Kit directory string

Output Parameters

`char* strBuffer`	`char` array returns AO Kit installation directory

Return Value

None

Example Code

```
#define STRING_LENGTH_BUFFER_512    512
char str[STRING_LENGTH_BUFFER_512];
int strLen = STRING_LENGTH_BUFFER_512;
GetControlSystemDirectory (str, &strLen);
```

4.2.3. GetAvailablePluginModules()

Get the available plugin modules of the specified plugin type.

Declaration

```
void *GetAvailablePluginModules(_PluginType pluginTypeSel, int *numModules);
```

Input Parameters

`_PluginType pluginTypeSel`	Specifies the plugin type. The valid types are:

```
enum _PluginType

{

    Sensor,

    Controller,

    Monitor,

    Processor

};
```

Output Parameters

`int *numModules`	Returns the number of plugin modules of specified plugin type.

Return Value

The pointer to the array returns the handles of the plugin modules.

Example Code

```
char str[STRING_LENGTH_BUFFER_512];
int strLen = STRING_LENGTH_BUFFER_512;


ModuleHandle *moduleHandles;

int numModules;
moduleHandles = (ModuleHandle *)GetAvailablePluginModules(_PluginType::Sensor,
                                                            &numModules);
if (numModules > 0)

{

   for (int moduleIndex = 0; moduleIndex < numModules; moduleIndex++)

   {

         strLength = STRING_LENGTH_BUFFER;

         GetModuleProperty(moduleHandles[moduleIndex],

                           _ModuleProperty::Module_Name,

                           str,

                           &strLength);


         strLength = STRING_LENGTH_BUFFER;

         GetModuleProperty(moduleHandles[moduleIndex],

                           _ModuleProperty::Module_Path,

                           str,
                                  &strLength);


            strLength = STRING_LENGTH_BUFFER;

            GetModuleProperty(moduleHandles[moduleIndex],

                              _ModuleProperty::Module_MajorVersion,

                              str,

                              &strLength);


            strLength = STRING_LENGTH_BUFFER;

            GetModuleProperty(moduleHandles[moduleIndex],

                              _ModuleProperty::Module_MinorVersion,

                              str,

                              &strLength);

        }

    }
```

4.2.4. GetAvailableExtensionModules()

Get the available plugin extension modules.

Declaration

```
void *GetAvailableExtensionModules(_ExtensionType extensionTypeSel,
                                   int *numModules);
```

Input Parameters

`_ExtensionType extensionTypeSel`	Specifies the extension type. Valid values are: `enum _ExtensionType { SensorExtension, ControllerExtension, MonitorExtension, ProcessorExtension };`

```
enum _ExtensionType

{

       SensorExtension,

       ControllerExtension,

       MonitorExtension,

       ProcessorExtension

};
```

Output Parameters

`int *numModules`	Returns the number of extension modules of specified extension type

Return Value

The pointer to the array returns the handles of extension modules

Example Code

```
ModuleHandle *hModules;

int numModules;
hModules = (ModuleHandle *)GetAvailableExtensionModules(_ExtensionType::SensorExtension,
                                                         &numModules);
```

4.2.5. GetModuleProperty()

Return the string of the specified plugin property.

Declaration

```
void GetModuleProperty(ModuleHandle hModule,
                       _ModuleProperty  propertySelection,
                       char *strBuffer,
                       int *bytesOfStrBuffer);
```

Input Parameters

`ModuleHandle hModule`	Handle of the plugin module
`_ModuleProperty  propertySelection`	Sets the plugin module property
`int *bytesOfStrBuffer`	Input value is the size of the `char` array `strBuffer` Returned value is the size of the module property string

Output Parameters

`char *strBuffer`	`char` array returns the module property string

Return Value

None

Example Code

See the same code for `GetAvailablePluginModules`

4.2.6. GetNumControlElements()

Get the number of control elements of the specified plugin type from the Processor extension module.

The Processor extension is developed for implementing the concrete AO control procedure and algorithm. It has the information of how many control elements and what types of control elements are used in the AO control algorithm.

Declaration

```
int GetNumControlElements(ModuleHandle hProcessorExtension, _PluginType
deviceType);
```

Input Parameters

`ModuleHandle hProcessorExtension`	Handle of the Processor extension
`_PluginType deviceType`	Specifies the type of the control element

Output Parameters

None

Return Value

The number of the control elements of the specified plugin type.

4.2.7. GetControlElementName()

Get the name of the control element specified by the plugin type and index from the Processor extension module.

Declaration

```
void GetControlElementName(ModuleHandle hProcessorExtension,
                           _PluginType deviceType,
                           int deviceIndex,
                           char *strBuffer,
                           int *bytesOfStrBuffer);
```

Input Parameters

`ModuleHandle hProcessorExtension`	Handle of the Processor extension
`_PluginType deviceType`	Specifies the plugin type of the control element
`int deviceIndex`	Index of the control element. Valid range is 0 to the number returned from `GetNumControlElements` - 1
`int *bytesOfStrBuffer`	Input value is the size of the `char` array. Returned valueis the size of the control element name string.

Output Parameters

`char *strBuffer`	`char` array returns the control element name.

Return Value

None

4.2.8. GetPluginDevices()

This function is to find the connected devices for plugin module specified by the Handle.

Declaration

```
void *GetPluginDevices(ModuleHandle hModule, int *numPluginDevices);
```

Input Parameters

`ModuleHandle hModule`	Handle of the plugin module.

Output Parameters

`int *numPluginDevices`	Pointer to a integer variable returns the number of the devices

Return Value

The pointer to the array returns the handles of the connected devices from the specified plugin module

Example Code

```
PluginHandle *deviceHandles = NULL;

int numDevices = 0;

deviceHandles = (PluginHandle *)GetPluginDevices(moduleHandle, &numDevices);
```

4.2.10. GetProcessorInstance()

Get the handle of the Processor from the Processor module.

Declaration

```
void *GetProcessorInstance(ModuleHandle hModule);
```

Input Parameters

`ModuleHandle hModule`	Handle of the Processor plugin module

Output Parameters

None

Return Value

The handle to the Processor instance returned from the Processor plugin module.

Example Code

```
PluginHandle *processorHandle = NULL;

processorHandle = (PluginHandle *)GetProcessorInstance(moduleHandle);
```

4.2.11. GetExtensionInstance()

Get the handle of the specified plugin extension.

Declaration

```
void *GetExtensionInstance(ModuleHandle hModule);
```

Input Parameters

`ModuleHandle hModule`	Handle of the plugin extension

Output Parameters

None

Return Value

The Handle to the plugin extension instance returned from the plugin extension module.

Example Code

```
ExtensionHandle *extensionHandle = NULL;
extensionHandle = (ExtensionHandle *)GetExtensionInstance(moduleHandle);
```

4.2.12. GetPluginProperty()

Return string of the specified plugin property.

Declaration

```
void GetPluginProperty(PluginHandle hPlugin,
                       _PluginDeviceProperty  propertySelection,
                       char *strBuffer,
                       int *bytesOfStrBuffer);
```

Input Parameters

`PluginHandle hPlugin`	Handle of the plugin
`_PluginDeviceProperty propertySelection`	Specifies the plugin property. Valid values are:

```
enum _PluginDeviceProperty

{

        Plugin_DeviceType,

        Plugin_DeviceDriverPath,


Plugin_DeviceResourceName,

        Plugin_DeviceName,

        Plugin_DeviceAliasName,

};
```

`int *bytesOfStrBuffer`	Input value is the size of the `char` array. Returned value is the size of the property string.

Output Parameters

`char *strBuffer`	`char` array returns the plugin property

Return Value

None

Example Code

```
char str[STRING_LENGTH_BUFFER_512];
int strLength = STRING_LENGTH_BUFFER;

GetPluginProperty(hPlugin,

                  _PluginDeviceProperty::Plugin_DeviceType,

                  str,

                  &strLength);


strLength = STRING_LENGTH_BUFFER;

GetPluginProperty(hPlugin,

                   _PluginDeviceProperty::Plugin_DeviceName,

                  str,

                  &strLength);


strLength = STRING_LENGTH_BUFFER;

GetPluginProperty(hPlugin,

                   _PluginDeviceProperty::Plugin_DeviceResourceName,

                   str,

                   &strLength);
```

4.2.13. ClosePluginManager()

Close the plugin manager.

Declaration

```
void ClosePluginManager();
```

Input Parameters

None

Output Parameters

None

Return Value

None

4.3. DataObject Operation

4.3.1. CreateData()

Create data object.

Declaration

```
DataHandle CreateData();
```

Input Parameters

None

Output Parameters

None

Return Value

The Handle of the created Data Object.

4.3.2. CreateData2()

Create data object with specified parameters.

Declaration

```
DataHandle CreateData2(_DataType data_type,
                       int size_1,
                       int size_2,
                       int size_3,
                       int size_4,
                       int components_per_data,
                       const char *data_description,
                       const char *data_unit);
```

Input Parameters

`_DataType data_type`	Specifies the data type of the Data Object. Valid values are:

```
enum _DataType
{
        Unknown = -1,
        UChar,
        Char,
        UShort,
        Short,
        UInt32,
        Int32,
        Float,
        Double,
};
```

`int size_1`	Specifies the size of dimension 1
`int size_2`	Specifies the size of dimension 2
`int size_3`	Specifies the size of dimension 3
`int size_4`	Specifies the size of dimension 4
`int components_per_data`	Sepcifies the number of components per data. Usually, this parameter is 1. But some data may have (X, Y) pair or the data is complex, in this situation, this parameter is 2
`const char *data_description`	Inforamtion of the Data Object
`const char *data_unit`	Unit of the Data Object

Output Parameters

None

Return Value

The Handle of the newly created Data Object

4.3.3. GetDataPropertyInt()

Return the selected integer property of specified data object.

Declaration

```
int GetDataPropertyInt(DataHandle hData, _DataPropertyInt propertySel);
```

Input Parameters

`DataHandle hData`	Handle of the Data Object
`_DataPropertyInt propertySel`	Specifies the property of the Data Object to be returned

Output Parameters

None

Return Value

The value of the specified property of the Data Object

Example Code

```
int iDataSize1 = GetDataPropertyInt(hDataObj, _DataPropertyInt::Data_Size1);

int iDataSize2 = GetDataPropertyInt(hDataObj, _DataPropertyInt::Data_Size2);
```

4.3.4. GetDataPropertyString()

Return the string of the specified property of the Data Object.

Declaration

```
void GetDataPropertyString(DataHandle hData,
                           _DataPropertyString propertySel,
                           char *strBuffer,
                           int *bytesOfStrBuffer);
```

Input Parameters

`DataHandle hData`	Handle of the Data Object
`_DataPropertyString propertySel`	Specifies the property of the Data Object
`int *bytesOfStrBuffer`	Input value is the size of the `char` array. Returned value is the size of the string returned by the Data Object

Output Parameters

`char *strBuffer`	`char` array returns the string of the property

Return Value

None

4.3.5. CalculateDataStatistics()

Calculate the statistics of the specified Data Object. The calculation will get the minimum value, maximum value, mean value and RMS of the Data Object.

Declaration

```
void CalculateDataStatistics(DataHandle hData);
```

Input Parameters

`DataHandle hData`	Handle of the Data Object.

Output Parameters

None

Return Value

None

4.3.6. GetDataStatistics()

Return the value of the specified data statistics.

Declaration

```
double GetDataStatistics(DataHandle hData, _DataStatistics statisticsSel);
```

Input Parameters

`DataHandle hData`	Handle of the Data Object
`_DataStatistics statisticsSel`	Specifies the data property to be returned. Valid values are:

```
enum _DataStatistics

{

    Data_Min,

    Data_Mean,

    Data_Max,

    Data_RMS

};
```

Output Parameters

None

Return Value

The value of the specified statistic of the Data Object

Example Code

```
double maxValue = GetDataStatistics(hData, _DataStatistics::Data_Max);
```

4.3.7. CopyData()

Copy data from the internal memory of the source Data Object to the internal memory of the destination Data Object.

Declaration

```
void CopyData(DataHandle hDataDst, DataHandle hDataSrc);
```

Input Parameters

`DataHandle hDataDst`	Handle of the destiny Data Object
`DataHandle hDataSrc`	Handle of the source Data Object

Output Parameters

None

Return Value

None

4.3.8. CopyDataContent()

Copy data from internal memory of the source Data Object to the external memory specified by pointer.

Declaration

```
void CopyDataContent(DataHandle hData, void *pDataDst);
```

Input Parameters

`DataHandle hData`	Handle of the source Data Object
`void *pDataDst`	Pointer to the memory

Output Parameters

None

Return Value

None

Example Code

```
int s1 = GetDataPropertyInt(hData, _DataPropertyInt::Data_Size1);

s1 = (s1 == 0) ? 1 : s1;


int s2 = GetDataPropertyInt(hData, _DataPropertyInt::Data_Size2);

s2 = (s2 == 0) ? 1 : s2;


int s3 = GetDataPropertyInt(hData, _DataPropertyInt::Data_Size3);

s3 = (s3 == 0) ? 1 : s3;


int s4 = GetDataPropertyInt(hData, _DataPropertyInt::Data_Size4);

s4 = (s4 == 0) ? 1 : s4;


int iBytesPerComponet = GetDataPropertyInt(hData,
_DataPropertyInt::Data_BytesPerComponent);


int iComponentPerData = GetDataPropertyInt(hData,
_DataPropertyInt::Data_ComponentsPerData);


int iTotalBytes = s1 * s2 * s3 * s4 * iComponentPerData * iBytesPerComponent;

void *byteArray = new byte[iTotalBytes];



CopyDataContent(hArraySpotCentroid, byteArray);


for (int i = 0; i < iTotalBytes; i++)

      byteArray[i] = i;

SetDataContent(hData, byteArray);


delete byteArray;
```

4.3.9. SetDataContent()

Copy data from external memory specified by the pointer to the internal memory of the Data Object

Declaration

```
void SetDataContent(DataHandle hData, void *pDataSrc);
```

Input Parameters

`DataHandle hData`	Handle of the destiny Data Object
`void *pDataSrc`	Pointer to the external source memory

Output Parameters

None

Return Value

None

Example Code

See `CopyDataContent`

4.3.10. SaveDataToFile()

Save the data to the specified file.

Declaration

```
void SaveDataToFile(DataHandle hData,
                    _ExportFileType fileType,
                    const char *fileName);
```

Input Parameters

`DataHandle hData`	Handle of the Data Object

`_ExportFileType fileType`	Specifies the output file type. Valid values are:
	`enum _ExportFileType`
	`{`
	`File_raw = 1,`
	`File_text,`
	`File_csv`
	`};`

`const char *fileName`	The path of the output file

Output Parameters

None

Return Value

None

4.3.11. ClearData()

Clear the data object.

Declaration

```
void ClearData(DataHandle hData);
```

Input Parameters

`DataHandle hData`	Handle of the Data Object

Output Parameters

None

Return Value

None

4.4. SignalObject Operation

The signal is the data used in the closed-loop control. For example, in an AO Kit control system, the spot deviations measured in the Thorlabs wavefront sensor are used to calculate the feedback for the deformable mirror. If the BMC Multi-Deformable-Mirror is used, the feedback is actuators' voltages, if Thorlabs DMP40 deformable mirror is used, the feedback is Zernike coefficients. The spot deviations data is defined as the Signal for the wavefront sensor, the actuators' voltages is the signal for BMC deformable mirror, and the Zernike coefficents data is the Signal for Thorlabs DMP40 deformable mirror. A SignalObject is created to represent the Signal data.

4.4.1. GetSignalType()

Return the type of the signal.

Declaration

```
_SignalType GetSignalType(DataHandle hSignalData);
```

Input Parameters

`DataHandle hSignalData`	Handle of the Signal Object

Output Parameters

None

Return Value

The signal type. Valid values are:

```
enum _SignalType

{

        Sensor_Signal,

        Controller_Signal,

        };
```

4.4.2. SetControllerSignalPokeMode()

Set the poke mode of the controller signal.

Declaration

```
void SetControllerSignalPokeMode(DataHandle hSignalData,

                                 _ControllerSignal_PokeMode mode);
```

Input Parameters

`DataHandle hSignalData` Handle of the Controller SignalObject

`_ControllerSignal_PokeMode mode` Specifies the "poke" mode of the Controller signal. The possible values are:

```
enum _ControllerSignal_PokeMode

{

        SingleCommandLinearPoke,

        DoubleCommandLinearPoke,

};
```

Output Parameters

None

Return Value

None

Example Code

For BMC deformable mirror, the “poke” mode is `SingleCommandLinearPoke`

```
SetControllerSignalPokeMode(hSignal, _ControllerSignal_PokeMode::SingleCommandLinearPoke);
```

For Thorlabs DMP40 deformable mirror, the “poke” mode is `DoubleCommandLinearPoke`

```
SetControllerSignalPokeMode(hSignal, _ControllerSignal_PokeMode::
DoubleCommandLinearPoke);
```

4.4.3. GetControllerSignalPokeMode()

Get the “poke” mode of the controller signal.

Declaration

```
_ControllerSignal_PokeMode GetControllerSignalPokeMode(DataHandle hSignalData);
```

Input Parameters

`DataHandle hSignalData`	Handle of the Controller SignalObject

Output Parameters

None

Return Value

The “poke” mode of the Controller SignalObject

4.4.4. GetControllerSignal_ArrayPtr()

Get the pointer of the specified internal memory from the Controller SignalObject

Declaration

```
void *GetControllerSignal_ArrayPtr(DataHandle hSignalData,
                                   _ControllerSignal_InternalArray arraySel);
```

Input Parameters

`DataHandle hSignalData`	Handle of the Controller SignalObject
`_ControllerSignal_InternalArray arraySel`	Specifies the internal memory. Available options are: `enum _ControllerSignal_InternalArray { ControllerSignal_Mask, ControllerSignal_CorrectionData, ControllerSignal_HomeCommands, ControllerSignal_Calibration_OffsetCommands, ControllerSignal_Calibration_StartCommands, ControllerSignal_Calibration_MaxCommands, };`

```
enum _ControllerSignal_InternalArray

{

        ControllerSignal_Mask,

        ControllerSignal_CorrectionData,

        ControllerSignal_HomeCommands,

        ControllerSignal_Calibration_OffsetCommands,

        ControllerSignal_Calibration_StartCommands,

        ControllerSignal_Calibration_MaxCommands,

};
```

Output Parameters

None

Return Value

The pointer to the internal memory in the Controller SignalObject

Example Code

```
double *offsetCommands = (double *)GetControllerSignal_ArrayPtr(hSignal,

        _ControllerSignal_InternalArray::ControllerSignal_Calibration_OffsetCommands);


double *maxCommands = (double *)GetControllerSignal_ArrayPtr(hSignal,
        _ControllerSignal_InternalArray::ControllerSignal_Calibration_MaxCommands);
```

4.4.5. ControllerSignalPoke()

Poke controller signal by specified element and command.

Declaration

```
void ControllerSignalPoke(DataHandle hSignalData,
                          int calibrationIndex,
                          int commandIndex,
                          double *command);
```

Input Parameters

`DataHandle hSignalData`	Handle of the Controller SignalObject
`int calibrationIndex`	Index of the Controller signal element
`int commandIndex`	Index of the “poke” command. For BMC deformable mirror, it is always 0; for Thorlabs DMP40 deformable mirror, when Zernike Coefficient goes to negavie max, the command index is 0, when it goes to positive max, the command index is 1
`double *command`	Value of the command (passed by pointer)

Output Parameters

None

Return Value

None

4.5. Coloring Data and Colored DataObject

4.5.1. CreateColoring()

Create coloring object to colorize the data object.

Declaration

```
ColoringHandle CreateColoring();
```

Input Parameters

None

Output Parameters

None

Return Value

The Handle of ColoringObject

4.5.2. SetColoringBoundaries()

Set the lower and upper boundaries of coloring.

Declaration

```
void SetColoringBoundaries(ColoringHandle hColoring,
                           double lowerBoundary,
                           double upperBoundary);
```

Input Parameters

`ColoringHandle hColoring`	Handle of the ColoringObject
`double lowerBoundary`	The lower boundary of the coloring. The pixel value will be set to lower boundar if it is smaller than this value
`double upperBoundary`	The upper boundary of the coloring. The pixel value will be limit to upper boundary if it is larger than this value

Output Parameters

None

Return Value

None

4.5.3. CreateColoredData()

Create colored data object.

Declaration

```
ColoredDataHandle CreateColoredData();
```

Input Parameters

None

Output Parameters

None

Return Value

The handle of the created ColoredDataObject

4.5.4. ColorizeDataARGB()

Colorize the data.

Declaration

```
void ColorizeDataARGB(ColoringHandle hColoring,
                      DataHandle hInputData,
                      ColoredDataHandle hOutputColoredData,
                      unsigned char alpha);
```

Input Parameters

`ColoringHandle hColoring`	Handle of the ColoringObject
`DataHandle hInputData`	Handle of the input DataObject
`ColoredDataHandle hOutputColoredData`	Handle of the output ColoredDataObject
`unsigned char alpha`	Alpha value

Output Parameters

None

Return Value

None

4.5.5. SaveColoredDataToImage()

Save the colored data to the specified image.

Declaration

```
void SaveColoredDataToImage(ColoredDataHandle hColoredData,
                            _ExportImageType imageType,
                            const char *imageName);
```

Input Parameters

`ColoredDataHandle hColoredData`	Handle of the ColoredDataObject
`ExportImageType imageType`	Sets the output image format. Valid values are:
`const char *imageName`	Output image file name

```
enum _ExportImageType
{
    Image_bmp = 1,
    Image_tiff,
    Image_png,
    Image_jpg
};
```

Output Parameters

None

Return Value

None

4.5.6. ClearColoredData()

Clear colored data object.

Declaration

```
void ClearColoredData(ColoredDataHandle hColoredData);
```

Input Parameters

`ColoredDataHandle hColoredData`	Handle of the ColoredDataObject

Output Parameters

None

Return Value

None

4.5.7. CloseColoring()

Close coloring object.

Declaration

```
void CloseColoring(ColoringHandle hColoring);
```

Input Parameters

`ColoringHandle hColoring`	Handle of the ColoringObject

Output Parameters

None

Return Value

None

4.6. File Operations

4.6.1. CreateCtrlSysFile()

Create control system file.

Declaration

```
FileHandle CreateCtrlSysFile(_FileOpenMode ctrlSysFileOpenMode,
                             const char *ctrlSysFilePath);
```

Input Parameters

`_FileOpenMode ctrlSysFileOpenMode` Sets the ControlSystemFileObject open mode. The valid values are:

```
enum _FileOpenMode
{
    Write,
    Read
};
```

`const char *ctrlSysFilePath` Specifies the full path of the ControlSystemFileObject

Output Parameters

None

Return Value

The Handle of the created ControlSystemFileObject

4.6.2. CloseCtrlSysFile()

Close the control system file.

Declaration

```
void CloseCtrlSysFile(FileHandle hFile);
```

Input Parameters

`FileHandle hFile`	Handle of the ControlSystemFileObject

Output Parameters

None

Return Value

None

4.6.3. AppendData()

Add new data to the file.

Declaration

```
void AppendData(FileHandle hFile, const char *dataName, DataHandle hData);
```

Input Parameters

`FileHandle hFile`	Handle of the ControlSystemFileObject
`const char *dataName`	Specifies the name of the input DataObject
`DataHandle hData`	Handle of the input DataObject

Output Parameters

None

Return Value

None

Example Code

After running the example code below, the wavefront data will be saved into file "Wavefront.dat" and added to the ControlSystemFileObject:

```
AppendData(hFile, "Wavefront", hWavefront);
```

4.6.4. AddFile()

Add a file and zipped it into the specified control system file.

Declaration

```
void AddFile(FileHandle hFile, const char *zippedFilePath);
```

Input Parameters

`FileHandle hFile`	Handle of the ControlSystemFileObject
`const char *zippedFilePath`	The full path of the file to be added

Output Parameters

None

Return Value

None

Example Code

```
AddFile(hFile, "C:\\Users\\AOKit\\Example\\WavefrontSensorStatistics.txt");
```

4.6.5. UnzipFile()

Unzip specified file to specified path.

Declaration

```
bool UnzipFile(FileHandle hFile,
               const char *zippedFileName,
               const char *unzippedFilePath);
```

Input Parameters

`FileHandle hFile`	Handle of the ControlSystemFileObject
`const char *zippedFileName`	The name of the data file zipped in the ControlSystemFileObject. No path information needed
`const char *unzippedFilePath`	The full target path for unzipping the data file

Output Parameters

None

Return Value

None

Example Code

```
UnzipFile(hFile, “Wavefront.data”, "C:\\Users\\AOKit\\Example\\Wavefront.dat");
```

4.6.6. SaveCtrlSysFile()

After calling `AppendData` or `AddFile`, this function creates a control system file and zips all the data or files into it.

Declaration

```
void SaveCtrlSysFile(FileHandle hFile);
```

Input Parameters

`FileHandle hFile`	Handle of the ControlSystemFileObject

Output Parameters

None

Return Value

None

Example Code

```
FileHandle hFile = CreateCtrlSysFile(_FileOpenMode::Write, fullFilePath);


AppendData(hFile, "Wavefront", hWavefront);

AddFile(hFile, "C:\\Users\\AOKit\\Example\\WavefrontStatistics.txt");


SaveCtrlSysFile(hFile);

CloseCtrlSysFile(hFile);
```

4.6.7. GetNumZippedFiles()

Get the number of zipped files in specified ControlSystemFileObject. This includes data files, header file, parameter files and others.

Declaration

```
int GetNumZippedFiles(FileHandle hFile);
```

Input Parameters

`FileHandle hFile`	Handle of the ControlSystemFileObject

Output Parameters

None

Return Value

The number of total files in the ControlSystemFileObject

4.6.8. GetZippedFilename()

Get the name of zipped file specified by the index.

Declaration

```
void GetZippedFilename(FileHandle hFile,
                       unsigned int zippedFileIndex,
                       char *strBuffer, int *bytesOfStrBuffer);
```

Input Parameters

`FileHandle hFile`	Handle of the ControlSystemFileObject
`unsigned int zippedFileIndex`	Index of the zipped file. Valid range is from 0 to the number of total zipped file - 1
`int *bytesOfStrBuffer`	Input valueis the size of the `char` array. Returned value is the size of the file name string.

Output Parameters

`char *strBuffer`	`char` array returns the file name

Return Value

None

4.6.9. GetNumDataFiles()

Get the number of zipped data files in specified ControlSystemFileObject. This only includes data files (.dat).

Declaration

```
int GetNumDataFiles(FileHandle hFile);
```

Input Parameters

`FileHandle hFile`	Handle of the ControlSystemFileObject

Output Parameters

None

Return Value

The number of total data files (.dat) zipped in the ControlSystemDataObject

4.6.10. ExportDataFileToData()

Export the data file specified by the index to the data object.

Declaration

```
void ExportDataFileToData(FileHandle hFile,
                          DataHandle hData,
                          unsigned int dataFileIndex);
```

Input Parameters

`FileHandle hFile`	Handle of the ControlSystemFileObject
`DataHandle hData`	Handle of the DataObject to receive the data exported from the ControlSystemFileObject
`unsigned int dataFileIndex`	Index of the data file in the ControlSystemFileOjbect. The valid range is 0 to the number of total data files – 1.

Output Parameters

None

Return Value

None

4.6.11. ExportDataFileToFile()

Export the data file specified by the index in ControlSystemFileObject to the specified file.

Declaration

```
void ExportDataFileToFile(FileHandle hFile,
                          _ExportFileType exportFileType,
                          const char *exportFilePath,
                          unsigned int dataFileIndex);
```

Input Parameters

`FileHandle hFile`	Handle of the ControlSystemFileObject
`_ExportFileType exportFileType`	Specifies the output file type
`const char *exportFilePath`	Specifies the output file full path
`unsigned int dataFileIndex`	Index of the data file to be exported

Output Parameters

None

Return Value

None

4.6.12. ExportDataFileToImage()

Export the data file specified by the index in the ControlSystemFileObject to the image in specified format.

Declaration

```
void ExportDataFileToImage(FileHandle hFile,
                           _ExportImageType exportImageType,
                           const char *exportFilePath,
                           unsigned int dataFileIndex);
```

Input Parameters

`_ExportImageType exportImageType`	Specifies the output image format
`FileHandle hFile`	Handle of the ControlSystemFileObject
`const char *exportFilePath`	Specifies the output image full path
`unsigned int dataFileIndex`	Index of the data file to be exported

Output Parameters

None

Return Value

None

4.6.13. GetZippedDataPropertyInt()

Get the value of the specified property from the zipped data specified by the index.

Declaration

```
int GetZippedDataPropertyInt(FileHandle hFile,
                             unsigned int dataFileIndex,
                             _DataPropertyInt propertySel);
```

Input Parameters

`FileHandle hFile`	Handle of the ControlSystemFileObject
`unsigned int dataFileIndex`	Index of the data file
`_DataPropertyInt propertySel`	Specifies the property

Output Parameters

None

Return Value

The value of the specified property

4.6.14. GetZippedDataPropertyString()

Get the string of the specified property from the zipped data specified by the index.

Declaration

```
void GetZippedDataPropertyString(FileHandle hFile,
                                 unsigned int dataFileIndex,
                                 _DataPropertyString propertySel,
                                 char *strBuffer,
                                 int *bytesOfStrBuffer);
```

Input Parameters

`FileHandle hFile`	Handle of the ControlSystemFileObject
`unsigned int dataFileIndex`	Index of the data file
`_DataPropertyString propertySel`	Specifies the property
`int *bytesOfStrBuffer`	Input value is the size of the `char` array. Returned value is the size of the property string.

Output Parameters

`char *strBuffer`	`char` array returns the property string

Return Value

None

4.6.15. GetZippedDataFileInfoString()

Get the string of the specified property from the data file specified by the index.

Declaration

```
void GetZippedDataFileInfoString(FileHandle hFile,
                                 unsigned int dataFileIndex,
                                 _DataFilePropertyString propertySel,
                                 char *strBuffer,
                                 int *bytesOfStrBuffer);
```

Input Parameters

`FileHandle hFile`	Handle of the ControlSystemFileObject
`unsigned int dataFileIndex`	Index of the data file
`_DataFilePropertyString propertySel`	Specifies the property. Possible value are: `enum _DataFilePropertyString { DataFile_Name, DataFile_CreationTime };`
`int *bytesOfStrBuffer`	Input value is the size of the `char` array. Returned value is the size of the property string.

Output Parameters

`char *strBuffer`	`char` array returns the property string

Return Value

None

4.7. Extension Operations

4.7.1. SetExtensionNumericParameter()

Set the value of the parameter specified by the name.

Declaration

```
void SetExtensionNumericParameter(ExtensionHandle hExtension,
                                  const char *fieldName,
                                  void *value);
```

Input Parameters

`ExtensionHandle hExtension`	Handle of the plugin extension
`const char *fieldName`	Parameter name
`void *value`	Value passed by the pointer

Output Parameters

None

Return Value

None

4.7.2. SetExtensionBooleanParameter()

Set the value of boolean parameter specified by the name.

Declaration

```
void SetExtensionBooleanParameter(ExtensionHandle hExtension,
                                  const char *fieldName,
                                  bool value);
```

Input Parameters

`ExtensionHandle hExtension`	Handle of the plugin extension
`const char *fieldName`	Boolean parameter name
`bool value`	Boolean value

Output Parameters

None

Return Value

None

4.7.3. SetExtensionStringParameter()

Set the string to the parameter specified by the parameter name.

Declaration

```
void SetExtensionStringParameter(ExtensionHandle hExtension,
                                 const char *fieldName,
                                 char *value);
```

Input Parameters

`ExtensionHandle hExtension`	Handle of the plugin extension
`const char *fieldName`	String parameter name
`char *value`	String passed by a `char` array

Output Parameters

None

Return Value

None

4.7.4. GetExtensionNumericParameter()

Get the numeric value of the parameter specified by the parameter name.

Declaration

```
void GetExtensionNumericParameter(ExtensionHandle hExtension,

                                  _NumericParameterProperty parameterProperty,

                                  const char *fieldName,

                                  void *value);
```

Input Parameters

`ExtensionHandle hExtension`	Handle of the plugin extension
`_NumericParameterProperty parameterProperty`	Specifies the property
`const char *fieldName`	Parameter name

Output Parameters

`void *value`	Parameter value returned by the pointer

Return Value

None

4.7.5. GetExtensionBooleanParameter()

Get the boolean value of the parameter specified by the parameter name.

Declaration

```
bool GetExtensionBooleanParameter(ExtensionHandle hExtension,
                                  const char *fieldName);
```

Input Parameters

`ExtensionHandle hExtension`	Handle of the plugin extension
`const char *fieldName`	Parameter name

Output Parameters

None

Return Value

Boolean value

4.7.6. GetExtensionStringParameter()

Get the string of the parameter specified by the parameter name.

Declaration

```
void GetExtensionStringParameter(ExtensionHandle hExtension,
                                 const char *fieldName,
                                 char *strBuffer,
                                 int *bytesOfStrBuffer);
```

Input Parameters

`ExtensionHandle hExtension`	Handle of the plugin extension
`const char *fieldName`	Parameter name
`int *bytesOfStrBuffer`	Input value is the size of the `char` array. Returned value is the size of the string.

Output Parameters

`char *strBuffer`	`char` array returns the parameter string

Return Value

None

4.7.7. GetExtensionDataObjectHandle()

Get the Handle of the DataObject specified by the data name from the extension.

Declaration

```
DataHandle GetExtensionDataObjectHandle(ExtensionHandle hExtension,
                                        const char *fieldName);
```

Input Parameters

`ExtensionHandle hExtension`	Handle of the plugin extension
`const char *fieldName`	DataObject name

Output Parameters

None

Return Value

Handle of the specified DataObject

4.7.8. SetExtensionOutputByName()

Set output specified by the name from the plugin extension.

Declaration

```
void SetExtensionOutputByName(ExtensionHandle hExtension, const char *fieldName);
```

Input Parameters

`ExtensionHandle hExtension`	Handle of the plugin extension
`const char *fieldName`	Output name

Output Parameters

None

Return Value

None

4.7.9. ClearExtensionOutputByName()

Clear specified output from the extension.

Declaration

```
void ClearExtensionOutputByName(ExtensionHandle hExtension,
                                const char *fieldName);
```

Input Parameters

`ExtensionHandle hExtension`	Handle of the plugin extension
`const char *fieldName`	Output name

Output Parameters

None

Return Value

None

4.7.10. CatchExtensionOutput()

Get the output data from the extension.

Declaration

```
void CatchExtensionOutput(ExtensionHandle hExtension);
```

Input Parameters

`ExtensionHandle hExtension`	Handle of the plugin extension

Output Parameters

None

Return Value

None

4.8. Control System

4.8.1. CreateControlSystem()

Create control system.

Declaration

```
CtrlSysHandle CreateControlSystem(const char *ctrlSysName);
```

Input Parameters

`const char *ctrlSysName`	Name of the control system to be created

Output Parameters

None

Return Value

Handle of the created control system

4.8.2. GetRootControlNode()

Get control system root node.

Declaration

```
CtrlNodeHandle GetRootControlNode(CtrlSysHandle hCtrlSys);
```

Input Parameters

`CtrlSysHandle hCtrlSys`	Handle of the control system

Output Parameters

None

Return Value

Handle of the root control node from the control system

4.8.3. NewControlNode()

Create new control node.

Declaration

```
CtrlNodeHandle NewControlNode(CtrlSysHandle hCtrlSys,
                              CtrlNodeHandle hParentCtrlNode,
                              const char *ctrlNodeName);
```

Input Parameters

`CtrlSysHandle hCtrlSys`	Handle of the control system
`CtrlNodeHandle hParentCtrlNode`	Handle of the parent control node
`const char *ctrlNodeName`	Name of the new control node

Output Parameters

None

Return Value

Handle of the newly created control node

4.8.4. DeleteControlNode()

Delete a control node.

Declaration

```
void DeleteControlNode(CtrlNodeHandle hCtrlNode);
```

Input Parameters

`CtrlNodeHandle hCtrlNode`	Handle of the control node to be deleted

Output Parameters

None

Return Value

None

4.8.5. AddControlElement()

Create new control element.

Declaration

```
CtrlElementHandle AddControlElement(CtrlNodeHandle hCtrlNode,
                                    void *hPlugin,
                                    const char *ctrlElementName);
```

Input Parameters

`CtrlNodeHandle hCtrlNode`	Handle of the control node to which the new control element to be added
`void *hPlugin`	Handle of the plugin device
`const char *ctrlElementName`	Name of the control element to be added

Output Parameters

None

Return Value

Handle of the newly created control element

4.8.6. RemoveControlElement()

Remove plugin from control system.

Declaration

```
void RemoveControlElement(CtrlNodeHandle hCtrlNode,
                          CtrlElementHandle hCtrlElement);
```

Input Parameters

`CtrlNodeHandle hCtrlNode`	Handle of the control node from which the control element to be removed
`CtrlElementHandle hCtrlElement`	Handle of the control element to be removed

Output Parameters

None

Return Value

None

4.8.7. SetUserExtension()

Set user extension.

Declaration

```
void SetUserExtension(CtrlElementHandle hCtrlElement, ExtensionHandle
hExtension);
```

Input Parameters

`CtrlElementHandle hCtrlElement`	Handle of the control element to which the plugin extension to be added
`ExtensionHandle hExtension`	Handle of the plugin extension to be added

Output Parameters

None

Return Value

None

Example Code

```
CtrlSysHandle hCtrlSys = NULL;

CtrlNodeHandle hRootNode = NULL;

CtrlNodeHandle hCtrlNode = NULL;

CtrlElementHandle hProcessorElement = NULL;

CtrlElementHandle hSensorElement = NULL;

CtrlElementHandle hControllerElement = NULL;

CtrlElementHandle hControllerElement1 = NULL;


hCtrlSys = CreateControlSystem("WT_LagrangeBased_ControlSystem");

hRootNode = GetRootControlNode(hCtrlSys);

hCtrlNode = NewControlNode(hCtrlSys, hRootNode, " WT_AO_ControlNode ");
```

```
hProcessorElement = AddControlElement(hCtrlNode, hProcessor, "Processor");
SetUserExtension(hProcessorElement, hProcessorExt);


hSensorElement = AddControlElement(hCtrlNode, hSensor, "Sensor");
hControllerElement = AddControlElement(hCtrlNode, hWoofer, "Woofer");
hControllerElement1 = AddControlElement(hCtrlNode, hTweeter, "Tweeter");
```

4.8.8. ClearUserExtension()

Clear user extension.

Declaration

```
void ClearUserExtension(CtrlElementHandle hCtrlElement);
```

Input Parameters

`CtrlElementHandle hCtrlElement`	Handle of the control element whose extension to be removed

Output Parameters

None

Return Value

None

4.8.9. SetNumericParameter()

Set the value to the parameter specified by the parameter name.

Declaration

```
void SetNumericParameter(CtrlElementHandle hCtrlElement,
                         const char *fieldName,
                         void *value);
```

Input Parameters

`CtrlElementHandle hCtrlElement`	Handle of the control element
`const char *fieldName`	Parameter name
`void *value`	Value passed by the pointer

Output Parameters

None

Return Value

None

4.8.10. SetStringParameter()

Set the string to parameter specified by the parameter name.

Declaration

```
void SetStringParameter(CtrlElementHandle hCtrlElement,
                        const char *fieldName,
                        char *value);
```

Input Parameters

`CtrlElementHandle hCtrlElement`	Handle of the control element
`const char *fieldName`	String parameter name
`char *value`	String passed by the pointer to a `char` array

Output Parameters

None

Return Value

None

4.8.11. SetBooleanParameter()

Set the value to a Boolean parameter specified by the name.

Declaration

```
void SetBooleanParameter(CtrlElementHandle hCtrlElement,
                         const char *fieldName,
                         bool value);
```

Input Parameters

`CtrlElementHandle hCtrlElement`	Handle of the control element
`const char *fieldName`	Boolean parameter name
`bool value`	Boolean value

Output Parameters

None

Return Value

None

4.8.12. GetNumericParameter()

Get the value of the specified numeric property from the specified parameter.

Declaration

```
void GetNumericParameter(CtrlElementHandle hCtrlElement,
                         _NumericParameterProperty parameterProperty,
                         const char *fieldName,
                         void *value);
```

Input Parameters

`CtrlElementHandle hCtrlElement`	Handle of the control element
`_NumericParameterProperty parameterProperty`	Sets the numeric property of the parameter. The value of selected property will be returned. Available properties are: `enum _NumericParameterProperty` `{` `Value,` `Minimum,` `Maximum,` `Step` `};`
`const char *fieldName`	Parameter name

Output Parameters

`void *value`	Value returned by a pointer

Return Value

None

4.8.13. GetStringParameter()

Get the string from the specified string parameter.

Declaration

```
void GetStringParameter(CtrlElementHandle hCtrlElement,
                        const char *fieldName,
                        char *strBuffer,
                        int *bytesOfStrBuffer);
```

Input Parameters

`CtrlElementHandle hCtrlElement`	Handle of the control element
`const char *fieldName`	String parameter name
`int *bytesOfStrBuffer`	Input value is the size of the `char` array. Returned value is the size of the parameter string.

Output Parameters

`char *strBuffer`	`char` array returns the parameter string

Return Value

None

4.8.14. GetBooleanParameter()

Get the value of the specified Boolean parameter.

Declaration

```
bool GetBooleanParameter(CtrlElementHandle hCtrlElement, const char *fieldName);
```

Input Parameters

`CtrlElementHandle hCtrlElement`	Handle of the control element
`const char *fieldName`	Boolean parameter name

Output Parameters

None

Return Value

Boolean value

4.8.15. GetEnumParameterElements()

Get all the enumeration elements. Each element is followed by char ';'.

Declaration

```
void GetEnumParameterElements(CtrlElementHandle hCtrlElement,
                              const char *fieldName,
                              char *strBuffer,
                              int *bytesOfStrBuffer);
```

Input Parameters

`CtrlElementHandle hCtrlElement`	Handle of the control element
`const char *fieldName`	Enumeration parameter name
`int *bytesOfStrBuffer`	Input value is the size of the `char` array. Returned value is the size of the string, which contains all enumeration elements.

Output Parameters

`char *strBuffer`	`char` array returns all the enumeration elements.

Return Value

None

Example Code

In the example below, the str will be "um;waves;"

```
#define STRING_LENGTH_BUFFER_512    512
char str[STRING_LENGTH_BUFFER_512];
int strLength = STRING_LENGTH_BUFFER_512;
GetEnumParameterElements(hWavefront, "WavefrontUnit", str, &strLength);
```

4.8.16. SetAllOutput()

Output all the data from the specified control element.

Declaration

```
void SetAllOutput(CtrlElementHandle hCtrlElement);
```

Input Parameters

`CtrlElementHandle hCtrlElement`	Handle of the control element

Output Parameters

None

Return Value

None

4.8.17. ClearAllOutput()

Clear data output from specified control element.

Declaration

```
void ClearAllOutput(CtrlElementHandle hCtrlElement);
```

Input Parameters

`CtrlElementHandle hCtrlElement`	Handle of the control element

Output Parameters

None

Return Value

None

4.8.18. SetOutputByName()

Set data output specified by the name from control element.

Declaration

```
void SetOutputByName(CtrlElementHandle hCtrlElement, const char *fieldName);
```

Input Parameters

`CtrlElementHandle hCtrlElement`	Handle of the control element
`const char *fieldName`	Output name

Output Parameters

None

Return Value

None

4.8.19. ClearOutputByName()

Clear output specified by the name from the control element.

Declaration

```
void ClearOutputByName(CtrlElementHandle hCtrlElement, const char *fieldName);
```

Input Parameters

`CtrlElementHandle hCtrlElement`	Handle of the control element
`const char *fieldName`	Output name

Output Parameters

None

Return Value

None

4.8.20. GetDataObjectHandle()

Get the handle of the DataObject from the control element.

Declaration

```
DataHandle GetDataObjectHandle(CtrlElementHandle hCtrlElement,
                               const char *fieldName);
```

Input Parameters

`CtrlElementHandle hCtrlElement`	Handle of the control element
`const char *fieldName`	DataObject name

Output Parameters

None

Return Value

The handle of the specified DataObject

4.8.21. GetElementSignalObjectHandle()

Get the handle of the SignalObject from the control element.

Declaration

```
DataHandle GetElementSignalObjectHandle(CtrlElementHandle hCtrlElement);
```

Input Parameters

`CtrlElementHandle hCtrlElement`	Handle of the control element

Output Parameters

None

Return Value

The handle of the SignalObject from the control element

4.8.22. LockElement()

Lock element for multi-threading operation safe.

Declaration

```
void LockElement(CtrlElementHandle hCtrlElement);
```

Input Parameters

`CtrlElementHandle hCtrlElement`	Handle of the control element

Output Parameters

None

Return Value

None

Example Code

```
LockElement(hBMC_DM);
```

```
for (int i = 0; i < iBMC_DM_DataSize1 * iBMC_DM_DataSize2; i++)
{
        dmData[i] = 0.5;
        SetDataContent(hArrayActuatorVoltages, dmData);
        ExecuteApiFunction(hBMC_DM, "BMCSetArray");
}

UnlockElement(hBMC_DM);
```

4.8.23. ExecuteCommand()

Set command to the specified control element.

Declaration

```
void ExecuteCommand(CtrlElementHandle hCtrlElement, const char *command);
```

Input Parameters

`CtrlElementHandle hCtrlElement`	Handle of the control element
`const char *command`	Command name

Output Parameters

None

Return Value

None

4.8.24. ExecuteApiFunction()

Execute the API function of the control element.

Declaration

```
void ExecuteApiFunction(CtrlElementHandle hCtrlElement, const char
*apiFunction);
```

Input Parameters

`CtrlElementHandle hCtrlElement`	Handle of the control element
`const char *apiFunction`	API function name

Output Parameters

None

Return Value

None

Example Code

```
LockElement
```

4.8.25. UnlockElement()

Unlock control element.

Declaration

```
void UnlockElement(CtrlElementHandle hCtrlElement);
```

Input Parameters

`CtrlElementHandle hCtrlElement`	Handle of the control element

Output Parameters

None

Return Value

None

Example Code

See example code in `LockElement`

4.8.26. InitControlNode()

Initialize the control node.

Declaration

```
void InitControlNode(CtrlNodeHandle hCtrlNode);
```

Input Parameters

`CtrlNodeHandle hCtrlNode`	Handle of the control node

Output Parameters

None

Return Value

None

4.8.27. InitControlElement()

Initialize the control element.

Declaration

```
void InitControlElement(CtrlElementHandle hCtrlElement);
```

Input Parameters

`CtrlElementHandle hCtrlElement`	Handle of the control element

Output Parameters

None

Return Value

None

4.8.28. SetElementControlMode()

Set element control mode.

Declaration

```
void SetElementControlMode(CtrlElementHandle hCtrlElement,
                           _ElementControlMode controlMode);
```

Input Parameters

`CtrlElementHandle hCtrlElement`	Handle of the control element
`_ElementControlMode controlMode`	Sets the element control mode. Available options are:

```
enum _ElementControlMode
{
      Realtime,
      Calibration
};
```

Output Parameters

None

Return Value

None

4.8.29. RunElement()

Start element.

Declaration

```
void RunElement(CtrlElementHandle hCtrlElement);
```

Input Parameters

`CtrlElementHandle hCtrlElement`	Handle of the control element

Output Parameters

None

Return Value

None

4.8.30. GetElementDeviceStatus()

Get element device status.

Declaration

```
_DeviceStatus GetElementDeviceStatus(CtrlElementHandle hCtrlElement,
                                     char *strBuffer,
                                     int *bytesOfStrBuffer);
```

Input Parameters

`CtrlElementHandle hCtrlElement`	Handle of the control element
`int *bytesOfStrBuffer`	Input value is the size of the `char` array. Returned value is the size of the deive status message.

Output Parameters

`char *strBuffer`	`char` array returns the device status message

Return Value

The device status. Possible status are:

```
enum _DeviceStatus

{

        Ready = 1

        DeviceOpen = 2

        InitializationRequired = 4,

        ExceptionOrError = 8,

        DeviceNotOperable = 128

};
```

4.8.31. CalibrateControlNode()

Calibrates the specified control node.

Declaration

```
void CalibrateControlNode(CtrlNodeHandle hCtrlNode,
                          _ProcessingStatus *processingStatus,
                          int *remainingSteps,
                          int *remainingCommandsAtCurrentStep,
                          char *processingName,
                          int *processingNameStrBufferLength,
                           char *elementsInProcessing,
                           int *elementsInProcessingStrBufferLength);
```

Input Parameters

`CtrlNodeHandle hCtrlNode`	Handle of the control node
`int *processingNameStrBufferLength`	Input value is the size of the `char` array receiving the processing name. Returned value is the size of the processing name.
`int *elementsInProcessingStrBufferLength`	Input value is the size of the `char` array receiving the involved elements' names Returned value is the size of the string, which contains involved elements' names.

Output Parameters

`_ProcessingStatus *processingStatus`	Returns the calibration status. `enum _ProcessingStatus { Update, OK, Calibration_Failed, Calibration_Finished, End_ControlPhase, End_ControlLoop, };`
`int *remainingSteps`	Indicates the remaining calibration steps
`int *remainingCommandsAtCurrentStep`	Indicates the remaining commands of current step
`char *processingName`	`char` array returns the processing name
`char *elementsInProcessing`	`char` array returns the involved elements’ names

Return Value

None

Example Code

Please check ThorAOControlSDK_WinConsole_Demo project for using `CalibrateControlNode`

4.8.32. SetControlNodeLoopMode()

Set the loop control mode.

Declaration

```
void SetControlNodeLoopMode(CtrlNodeHandle hCtrlNode,
                            _LoopControlMode loopControlMode);
```

Input Parameters

`CtrlNodeHandle hCtrlNode`	Handle of the control node
`_LoopControlMode loopControlMode`	Sets the loop control mode. Valid values are: `enum _LoopControlMode { open = 0, closed };`

Output Parameters

None

Return Value

None

4.8.33. StartNodeLoopControl()

Start running the spcified control node.

Declaration

```
void StartNodeLoopControl(CtrlNodeHandle hCtrlNode,
                          _ProcessingStatus *processingStatus,
                          char *processingName,
                          int *processingNameStrBufferLength,
                          char *elementsInProcessing,
                          int *elementsInProcessingStrBufferLength);
```

Input Parameters

`CtrlNodeHandle hCtrlNode`	Handle of the control node
`int *processingNameStrBufferLength`	Input value is the size of the `char` array. Returned value is the size of the processing name.
`int *elementsInProcessingStrBufferLength`	Input value is the size of the `char` array. Returned value is the size of the string, which contains the names of all the involved elements

Output Parameters

`_ProcessingStatus *processingStatus`	Returns the processing status
`char *processingName`	`char` array returns the processing name
`char *elementsInProcessing`	`char` array returns the names of all the involved elements

Return Value

None

Example Code

Please check ThorAOControlSDK_WinConsole_Demo project for using `StartNodeLoopControl`

4.8.34. StopNodeLoopControl()

Stop running the control node.

Declaration

```
void StopNodeLoopControl(CtrlNodeHandle hCtrlNode);
```

Input Parameters

`CtrlNodeHandle hCtrlNode`	Handle of the control node

Output Parameters

None

Return Value

None

4.8.35. GetNumericOutput()

Get the numeric value of the output specified.

Declaration

```
void GetNumericOutput(CtrlElementHandle hCtrlElement,
                      const char *outputName,
                      void *value);
```

Input Parameters

`CtrlElementHandle hCtrlElement`	Handle of the control element
`const char *outputName`	Output name

Output Parameters

`void *value`	Value returned by the pointer

Return Value

None

4.8.36. GetStringOutput()

Get the string of the output specified by the name.

Declaration

```
void GetStringOutput(CtrlElementHandle hCtrlElement,
                     const char *outputName,
                     char *strBuffer,
                     int *bytesOfStrBuffer);
```

Input Parameters

`CtrlElementHandle hCtrlElement`	Handle of the control element
`const char *outputName`	Output name
`int *bytesOfStrBuffer`	Input value is the size of the `char` array Returned value is the size of the string output

Output Parameters

`char *strBuffer`	`char` array returns the string output

Return Value

None

4.8.37. CatchElementOutput()

Get the output data of the control element.

Declaration

```
void CatchElementOutput(CtrlElementHandle hCtrlElement);
```

Input Parameters

`CtrlElementHandle hCtrlElement`	Handle of the control element

Output Parameters

None

Return Value

None

Example Code

```
RunElement(hWavefrontSensor);

CatchElementOutput(hWavefrontSensor);
```

4.8.38. CloseControlSystem()

Close control system handle.

Declaration

```
void CloseControlSystem(CtrlSysHandle hCtrlSys);
```

Input Parameters

`CtrlSysHandle hCtrlSys`	Handle of the control system

Output Parameters

None

Return Value

None

Example Code

Please see the VS 2017.NET (C++) demo program ThorAOControlSDK_WinConsole_Demo for example code that accompanies this documentation.

Chapter 5 Regulatory

As required by the WEEE (Waste Electrical and Electronic Equipment Directive) of the European Community and the corresponding national laws, Thorlabs offers all end users in the EC the possibility to return “end of life” units without incurring disposal charges.

- This offer is valid for Thorlabs electrical and electronic equipment:
- Sold after August 13, 2005
- Marked correspondingly with the crossed out “wheelie bin” logo (see right)
- Sold to a company or institute within the EC
- Currently owned by a company or institute within the EC
- Still complete, not disassembled and not contaminated

Wheelie Bin Logo

As the WEEE directive applies to self-contained operational electrical and electronic products, this end of life take back service does not refer to other Thorlabs products, such as:

- Pure OEM products, that means assemblies to be built into a unit by the user (e. g. OEM laser driver cards)
- Components
- Mechanics and optics
- Left over parts of units disassembled by the user (PCB’s, housings etc.).

If you wish to return a Thorlabs unit for waste recovery, please contact Thorlabs or your nearest dealer for further information.

Waste Treatment is Your Own Responsibility

If you do not return an “end of life” unit to Thorlabs, you must hand it to a company specialized in waste recovery. Do not dispose of the unit in a litter bin or at a public waste disposal site.

Ecological Background

It is well known that WEEE pollutes the environment by releasing toxic products during decomposition. The aim of the European RoHS directive is to reduce the content of toxic substances in electronic products in the future.

The intent of the WEEE directive is to enforce the recycling of WEEE. A controlled recycling of end of life products will thereby avoid negative impacts on the environment.

Chapter 6 Thorlabs Worldwide Contacts

For technical support or sales inquiries, please visit us at **www.thorlabs.com/contact** for our most up-to-date contact information.

USA, Canada, and South America
Thorlabs, Inc.
sales@thorlabs.com
techsupport@thorlabs.com

Europe
Thorlabs GmbH
europe@thorlabs.com

France
Thorlabs SAS
sales.fr@thorlabs.com

Japan
Thorlabs Japan, Inc.
sales@thorlabs.jp

UK and Ireland
Thorlabs Ltd.
sales.uk@thorlabs.com
techsupport.uk@thorlabs.com

Scandinavia
Thorlabs Sweden AB
scandinavia@thorlabs.com

Brazil
Thorlabs Vendas de Fotônicos Ltda.
brasil@thorlabs.com

China
Thorlabs China
chinasales@thorlabs.com

THORLABS
www.thorlabs.com