Photon etc | User Manual

XPHySpec

ActiveX API Specification

Version 1.19.0 Revision 2012-02-27

Table of Contents

1. About This Manual

This manual describes how to use XPHySpec, the ActiveX control giving access to many of PHySpec's capabilities, with each instrument in Photon etc.'s catalogue. Given that every instrument does not have the same functionalities, some XPHySpec functions might not be relevant for your instrument. XPHySpec gives access to many low-level functionalities of PHySpec in order to provide more flexibility to the user. For more information about PHySpec and its uses, consult its User Manual.

2. Customer Service

Business hours:	Monday to Friday from 9:00 to 17:00 Eastern Standard Time (GMT -5)
Email:	info@photonetc.com
Phone:	(514) 385-9555
Address:	**Photon etc. Inc.** 5795 De Gaspé Avenue, #222 Montréal, Québec H2S 2X3 Canada

To obtain more information regarding our products: sales@photonetc.com

To communicate with one of our experts regarding specific queries or applications: expert@photonetc.com

For technical support: support@photonetc.com

3. Introduction

XPHySpec is intended to be used by advanced users that need to integrate Photon etc. instrument in their process. XPHySpec is an ActiveX control and therefore can only work on a Windows platform. It is possible to access XPHySpec functions through scripting languages (VBscript) or software that support ActiveX interface (LabView, MATLAB). XPHySpec does not provide a graphical interface, which is the role of PHySpec, but rather offer access to many low-level functions of PHySpec. These functions consist of controlling Photon etc. instruments, managing memory of a cube and correcting images with various techniques. The XPHySpec control is registered in Windows at the installation process of the PHySpec software.

For those unfamiliar with ActiveX, it is useful to know that an ActiveX object, or COM object, has methods, properties and events. A method is a function, which can take both input and output parameters, executing a complex action. A property is a value that can be read-only, write-only or read-write. An event is an asynchronous message sent by the object to inform the user of a change. In XPHySpec, properties are used to get or set various states of an object, whether turning on and off a lamp or knowing the current position.

The following sections are ordered by ActiveX object accessible to the user and their hierarchical dependency. Each section describes the methods and properties of the specific object. Events are only mentioned in the description of the function that generates them. Finally, there is an example section that describes some of the most commonly used operations.

4. XPHySpec Control

XPHySpec is the control by which all methods, properties and sub-objects can be accessed. When connecting to the XPHySpec control, calling `InitializeSystem` is mandatory before using any other methods or properties.

4.1. XPHySpec Methods

The XPHySpec interface provides methods to operate all of Photon etc.'s instruments. As such, many methods will be unused in a particular context. Some XPHySpec methods are synchronous and others are asynchronous, meaning that they will return when they have not finished processing and the user has to wait for a specific event. This has the advantage of giving the user the opportunity to do some other processing while waiting for the event. On the other hand, the user has to know which events are sent by which methods. Also, managing events is somewhat more complicated than calling a synchronous method. XPHySpec tries to keep the majority of methods synchronous, but some especially laborious methods are asynchronous. The description for such methods explicitly states what the user needs to do in order to know that the processing is finished. Typically, two events can occur for an asynchronous method: an error event or a completion event. The signature of the error event is `sigError(int p_ErrorCode, BSTR p_ErrMessage)`. It is always possible to catch this event to have a description of the error, whether it comes from a synchronous or asynchronous method.

4.1.1. AbortAcquisition

If an acquisition is running, abort the process and stay at the current position, else take no action.

```
void AbortAcquisition()
```

Parameters

None.

Return value

None.

4.1.2. Acquire

Start and perform the acquisition process with a given device and camera to store the result in a cube. This method generates an `AcquisitionFinished` event upon completion.

```
int Acquire(XPHySpecTunableFilter *p_device
            XPHySpecCamera *p_camera,
            BSTR p_cubeName,
            double p_lowerLambda,
            double p_higherLambda,
            double p_step,
            double p_exposureTime,
            VARIANT_BOOL p_coverGradient)
```

Parameters

`[in] p_device:`	handle to a hyperspectral imager
`[in] p_camera:`	handle to a camera used to acquire images
`[in] p_cubeName:`	new cube that will be created at the end of the acquisition
`[in] p_lowerLambda:`	lower boundary of the wavelength range in nanometres
`[in] p_higherLambda:`	higher boundary of the wavelength range in nanometres
`[in] p_step:`	sampling resolution for the given range in nanometres
`[in] p_exposureTime:`	each image will have the given exposure time in seconds
`[in] p_coverGradient:`	set to `true` if additional images to cover the gradient are needed

Return value

Returns 0 if successful, otherwise the return value is an XPHySpec error code.

4.1.3. AddCameraToSystem

Select a connected camera and add it to the working environment. The camera must be detected first and then the property `ListOfAvailableCameras` gives the connected cameras from which to choose. This method generates a `CameraAdded` event when a camera is successfully added to the system.

```
int AddCameraToSystem(BSTR p_CameraName)
```

Parameters

`[in] p_CameraName:`	internal name for the camera, not necessarily its commercial name

Return value

Returns 0 if successful, otherwise the return value is an XPHySpec error code.

4.1.4. AddCube

Create an empty raw cube which will be referenced by its name in the XPHySpec system. This method generates a NewCubeAdded event, but it is only to be consistent with the LoadCube method. When returning from this method, a new cube is present in the system.

```
int AddCube(BSTR p_cubeName,
            int p_imageWidth,
            int p_imageHeight)
```

Parameters

[in] p_cubeName:	identifier for the new cube, must not already exists in the system
[in] p_imageWidth:	image width, usually taken from the camera dimensions
[in] p_imageHeight:	image height, usually taken from the camera dimensions

Return value

Returns 0 if successful, otherwise the return value is an XPHySpec error code.

4.1.5. AddCurrentImage

Add the last image taken by the specified camera to the cube. This method is similar to AddImage, it only fills the information from the device and camera for the user.

```
int AddCurrentImage(BSTR p_cubeName,
                    XPHySpecTunableFilter *p_device,
                    XPHySpecCamera *p_camera)
```

Parameters

[in] p_cubeName:	name of the cube in the system
[in] p_device:	handle to a hyperspectral imager
[in] p_camera:	handle to a camera with an image already taken

Return value

Returns 0 if successful, otherwise the return value is an XPHySpec error code.

4.1.6. AddImage

Add an image to the specified cube. The `p_wavelenght`, `p_braggAngle` and `p_gratingPeriod` parameters are only used in the rectifying process, so the user can pass zero values if he does not intend to rectify the cube.

```
int AddImage(BSTR p_cubeName,
             SAFEARRAY(BYTE) p_data,
             double p_wavelength,
             double p_braggAngle,
             double p_gratingPeriod)
```

Parameters

`[in] p_cubeName:`	name of the cube in the system
`[in] p_data:`	raw image, must be the same dimensions as the cube
`[in] p_wavelength:`	wavelength at which the image was taken
`[in] p_braggAngle:`	position of the grating in degree
`[in] p_gratingPeriod:`	period of the grating used in nanometres

Return value

Returns 0 if successful, otherwise the return value is an XPHySpec error code.

4.1.7. CalibrateHImagerGrating

Calibrate a specific grating of a hyperspectral imager automatically given an input file of calibration rays. This method generates a `CalibrationDone` event upon completion.

```
int CalibrateHImagerGrating(XPHySpecTunableFilter *p_HImager,
                            XPHySpecCamera *p_Camera,
                            double p_ExposureTime,
                            BSTR p_RaysFilename,
                            unsigned int p_GratingNumber)
```

Parameters

`[in] p_HImager:`	handle to a hyperspectral imager
`[in] p_Camera:`	handle to a camera used to acquire images
`[in] p_ExposureTime:`	each image will have the given exposure time in seconds
`[in] p_RaysFilename:`	location of the file containing rays for the calibration
`[in] p_GratingNumber:`	index of the grating to be calibrated

Return value

Returns 0 if successful, otherwise the return value is an XPHySpec error code.

4.1.8. CalibrateTSrcGrating

Calibrate a specific grating of a tunable source automatically. This method generates a `CalibrationDone` event upon completion.

```
int CalibrateTSrcGrating(XPHySpecTunableFilter *p_TunableSrc,
                         unsigned int p_GratingNumber)
```

Parameters

`[in] p_TunableSrc:`	handle to a tunable source
`[in] p_GratingNumber:`	index of the grating to be calibrated

Return value

Returns 0 if successful, otherwise the return value is an XPHySpec error code.

4.1.9. DeleteCube

Free the memory used by a cube in the system. The raw cube and its rectified version, if any, are deleted together.

```
int DeleteCube(BSTR p_cubeName)
```

Parameters

`[in] p_cubeName:`	name of the cube in the system

Return value

Returns 0 if successful, otherwise the return value is an XPHySpec error code.

4.1.10. DetectCameras

Populate an internal list of available cameras for a given driver. The supported driver names are available as properties and one of them should be used when calling this method.

```
int DetectCameras(BSTR p_DriverName)
```

Parameters

`[in] p_DriverName:`	driver to search cameras for

Return value

Returns 0 if successful, otherwise the return value is an XPHySpec error code.

4.1.11. DetectDevices

Populate an internal list of devices connected to the system. This method generates a `DeviceDetectionFinished` event upon completion.

```
int DetectDevices()
```

Parameters

None.

Return value

Returns 0 if successful, otherwise the return value is an XPHySpec error code.

4.1.12. GetCameraInterface

Give a handle to the camera sub-object matching the given name. The camera must be added with `AddCameraToSystem` before calling this method.

```
XPHySpecCamera *GetCameraInterface(BSTR p_CameraName)
```

Parameters

`[in] p_CameraName:`	internal name for the camera, not necessarily its commercial name

Return value

Return a handle to a camera sub-object, null if no valid camera is found.

4.1.13. GetDeviceInterface

Give a handle to the device sub-object matching the given serial number. To know which devices are available to control, the `DetectDevices` method must be called first.

```
XPHySpecTunableFilter *GetDeviceInterface(BSTR p_SerialNumber)
```

Parameters

`[in] p_SerialNumber:`	serial number written on the instrument

Return value

Return a handle to a device sub-object, null if no valid device is found.

4.1.14. InitializeSystem

Create a PHySpec backbone which XPHySpec interfaces to give access to a limited set of method and properties. This method must be called before any other one, else they will fail.

```
int InitializeSystem()
```

Parameters

None.

Return value

Returns 0 if successful, otherwise the return value is an XPHySpec error code.

4.1.15. LoadCube

Load a cube file from a given location in system memory. The file must be in the FITS format to be read by XPHySpec. This method generates a `NewCubeAdded` event upon completion.

```
int LoadCube(BSTR p_filename)
```

Parameters

`[in] p_filename:`	location of the cube FITS file

Return value

Returns 0 if successful, otherwise the return value is an XPHySpec error code.

4.1.16. RectifyCube

Rectify an existing raw cube which have been added by the `LoadCube` method or created by the `AddCube` method. The selected device must be the one used to create the cube for the results to make sense. This method generates a `CubeRectificationDone` event upon completion.

```
int RectifyCube(XPHySpecTunableFIlter *p_Device,
                BSTR p_CubeName,
                double p_WavelengthStart,
                double p_WavelengthEnd,
                double p_Resolution)
```

Parameters

`[in] p_Device:`	handle to a hyperspectral imager
`[in] p_CubeName:`	name of the cube in the system
`[in] p_WavelengthStart:`	lower boundary of the wavelength range in nanometres
`[in] p_WavelengthEnd:`	higher boundary of the wavelength range in nanometres
`[in] p_Resolution:`	sampling resolution for the given range in nanometres

Return value

Returns 0 if successful, otherwise the return value is an XPHySpec error code.

4.1.17. RectifyCubeFromCalibFile

Rectify an existing raw cube which have been added by the `LoadCube` method or created by the `AddCube` method. A previously saved calibration matrix must be used to rectify the cube, as the matrix must match the cube. This method generates a `CubeRectificationDone` event upon completion.

```
int RectifyFromCalibFile(BSTR p_MatrixFileName,
                         BSTR p_CubeName,
                         double p_WavelengthStart,
                         double p_WavelengthEnd,
                         double p_Resolution)
```

Parameters

`[in] p_MatrixFileName:`	location of the calibration matrix
`[in] p_CubeName:`	name of the cube in the system
`[in] p_WavelengthStart:`	lower boundary of the wavelength range in nanometres
`[in] p_WavelengthEnd:`	higher boundary of the wavelength range in nanometres
`[in] p_Resolution:`	sampling resolution for the given range in nanometres

Return value

Returns 0 if successful, otherwise the return value is an XPHySpec error code.

4.1.18. SaveCube

Save the given cube to a file in FITS format. The `p_cubeType` parameter can take two values: `1` to designate the raw version or `2` to designate the rectified version.

```
int SaveCube(BSTR p_cubeName,
             BSTR p_filename,
             int p_cubeType)
```

Parameters

`[in] p_cubeName:`	name of the cube in the system
`[in] p_filename:`	location where to save the cube
`[in] p_cubeType:`	type of cube

Return value

Returns 0 if successful, otherwise the return value is an XPHySpec error code.

4.1.19. SetCubeMetaInfo

Modify the current meta-information in the given cube with the new value specified. The change is made permanent when the cube is saved. The table below describes each meta-information available, note that the names are case sensitive.

Name	Description
`author`	author name
`description`	brief description of the cube content
`exposure`	exposure time used for the images in the cube
`height`	height of the images
`lambdaHigh`	last wavelength of the cube in nanometres
`lambdaLow`	first wavelength of the cube in nanometres
`lambdaStep`	sampling resolution for the given range in nanometres
`serial`	serial number of the device used to make the cube
`width`	width of the images

```
int SetCubeMetaInfo(BSTR p_cubeName,
                    BSTR p_name,
                    BSTR p_value)
```

Parameters

`[in] p_cubeName:`	name of the cube in the system
`[in] p_name:`	name of the meta-information to modify
`[in] p_value:`	new value to assign

Return value

Returns 0 if successful, otherwise the return value is an XPHySpec error code.

4.1.20. ToAndorIxon

Give access to methods only available for the AndorIxon camera through a specialized interface.

```
XPHySpecAndorIxon *ToAndorIxon(XPHySpecCamera *p_CameraInterface)
```

Parameters

`[in] p_CameraInterface:`	generic camera interface

Return value

Return a handle to a specialized camera sub-object, null if no valid camera is found.

4.1.21. ToApogee

Give access to methods only available for the Apogee camera through a specialized interface.

```
XPHySpecApogee *ToApogee(XPHySpecCamera *p_CameraInterface)
```

Parameters

`[in] p_CameraInterface`: generic camera interface

Return value

Return a handle to a specialized camera sub-object, null if no valid camera is found.

4.1.22. ToHI

Give access to methods only available for the hyperspectral imager through a specialized interface.

```
XPHySpecHImager *ToHI(XPHySpecTunableFilter *p_DeviceInterface)
```

Parameters

`[in] p_DeviceInterface`: generic device interface

Return value

Return a handle to a specialized device sub-object, null if no valid device is found.

4.1.23. ToLLTF

Give access to methods only available for the laser line tunable filter through a specialized interface.

```
XPHySpecLLTunableFilter *ToLLTF(XPHySpecTunableFilter *p_DeviceInterface)
```

Parameters

`[in] p_DeviceInterface`: generic device interface

Return value

Return a handle to a specialized device sub-object, null if no valid device is found.

4.1.24. ToPCO

Give access to methods only available for the PCO camera through a specialized interface.

```
XPHySpecPCO *ToPCO(XPHySpecCamera *p_CameraInterface)
```

Parameters

`[in] p_CameraInterface`: generic camera interface

Return value

Return a handle to a specialized camera sub-object, null if no valid camera is found.

4.1.25. ToTLS

Give access to methods only available for the tunable laser source through a specialized interface.

```
XPHySpecTunableLaserSource *ToTLS(XPHySpecTunableFilter *p_DeviceInterface)
```

Parameters

`[in] p_DeviceInterface`: generic device interface

Return value

Return a handle to a specialized device sub-object, null if no valid device is found.

4.1.26. ToTS

Give access to methods only available for the tunable source through a specialized interface.

```
XPHySpecTunableSource *ToTS(XPHySpecTunableFilter *p_DeviceInterface)
```

Parameters

`[in] p_DeviceInterface`: generic device interface

Return value

Return a handle to a specialized device sub-object, null if no valid device is found.

4.2. XPHySpec Properties

XPHySpec provides properties to query the state of the system at any time. The user should not rely on the progress properties to know when a process is completed, but should wait for the appropriate event instead.

Type	Name	Description
unsigned int	AcquisitionProgress	progress of the acquisition given in percentage
BSTR	AndorDriverName	supported Andor camera driver name
BSTR	ApogeeDriverName	supported Apogee camera driver name
unsigned int	AutoCalibProgress	progress of the calibration given in percentage
VARIANT_BOOL	IsAcquiring	indicate the state of the acquisition
SAFEARRAY(BSTR)	ListOfAvailableCameras	list of detected cameras currently in the system
SAFEARRAY(BSTR)	ListOfAvailableDevices	list of detected devices currently in the system
SAFEARRAY(VARIANT)	ListOfAvailableRawCubes	list of cube currently in the system
BSTR	PCODriverName	supported PCO camera driver name
unsigned int	RecitficationProgress	progress of the rectification given in percentage
VARIANT_BOOL	SerialPostIsOpen	state of the serial port

5. XPHySpecTunableFilter Object

This object is not directly creatable by the user; it is obtained through the `GetDeviceInterface` method. All methods and properties of this object are copied for each specialized sub-object: `XPHySpecHImager`, `XPHySpecLLTunableFilter`, `XPHySpecTunableLaserSource` and `XPHySpecTunableSource`. The user should always convert to the more specialized sub-object corresponding to its instrument when he needs to use a method, even if the method is available in a more generic sub-object.

5.1. XPHySpecTunableFilter Methods

5.1.1. CalibrateGrating

Calibrate a grating manually by giving two stage positions for which the user matches two wavelengths.

```
int CalibrateGrating(double p_lambda1,
                     double p_realAngle1,
                     double p_lambda2,
                     double p_realAngle2)
```

Parameters

`[in] p_lambda1:`	first theoretical wavelength in nanometres
`[in] p_realAngle1:`	real stage angle in degrees for the first wavelength
`[in] p_lambda2:`	second theoretical wavelength in nanometres
`[in] p_realAngle2:`	real stage angle in degrees for the second wavelength

Return value

Returns 0 if successful, otherwise the return value is an XPHySpec error code.

5.1.2. CalibrateMotorizedLens

Calibrate the offset of the motorized lens according to a wavelength for a given grating.

```
int CalibrateMotorizedLens(int p_gratingIndex,
                           double p_position,
                           double p_wavelength)
```

Parameters

`[in] p_gratingIndex:`	grating to apply the offset to
`[in] p_position:`	motorized lens position in micrometres
`[in] p_wavelength:`	matching wavelength for the position in nanometres

Return value

Returns 0 if successful, otherwise the return value is an XPHySpec error code.

5.1.3. EnableFollowWavelength

Enable the motorized lens to automatically follow the wavelength to keep its focus once calibrated.

```
void EnableFollowWavelength(VARIANT_BOOL p_automatic)
```

Parameters

`[in] p_automatic:`	mode in which to operate

Return value

None.

5.1.4. IsCalibrated

Give the calibration state of a grating.

```
VARIANT_BOOL IsCalibrated(int p_gratingIndex)
```

Parameters

`[in] p_gratingIndex:`	index of the grating to check

Return value

True if the grating is calibrated, false otherwise.

5.1.5. MoveMotorizedLens

Move the motorized lens to the given position.

```
int MoveMotorizedLens(int p_position)
```

Parameters

`[in] p_position:`	motorized lens position in micrometres

Return value

Returns 0 if successful, otherwise the return value is an XPHySpec error code.

5.1.6. MoveToAngle

Move the device to the given angle. This method generates a `sigStageMoved` event upon completion.

```
int MoveToAngle(double p_TargetAngle)
```

Parameters

`[in] p_TargetAngle:`	stage angle in degrees

Return value

Returns 0 if successful, otherwise the return value is an XPHySpec error code.

5.1.7. MoveToWavelength

Move the device to the given wavelength. This method generates a `sigStageMoved` event upon completion.

```
int MoveToWavelength(double p_TargetWavelength)
```

Parameters

`[in] p_TargetWavelength:`	desired wavelength in nanometres

Return value

Returns 0 if successful, otherwise the return value is an XPHySpec error code.

5.1.8. MoveToWavelengthInGrating

Move the device to the given wavelength for a specific grating. The selection of the grating permits access to wavelengths normally unreachable otherwise. This method generates a `sigStageMoved` event upon completion.

```
int MoveToWavelengthInGrating(int p_gratingIndex,
                              double p_wavelength)
```

Parameters

`[in] p_gratingIndex:`	index of the grating to move
`[in] p_wavelength:`	desired wavelength in nanometres

Return value

Returns 0 if successful, otherwise the return value is an XPHySpec error code.

5.1.9. SaveCalibrationMatrices

Save the calibration matrix of each grating. Each grating has its matrix saved in a file following this pattern: `p_Filename.gratingKit[X].grating[Y].fits`.

```
int SaveCalibrationMatrices(BSTR p_Filename)
```

Parameters

`[in] p_Filename:`	location where to save the matrix

Return value

Returns 0 if successful, otherwise the return value is an XPHySpec error code.

5.1.10. Step

Move the device by the given step size. This method generates a `sigStageMoved` event upon completion.

```
int Step(double p_StepWavelength)
```

Parameters

`[in] p_StepWavelength:` desired step in nanometres

Return value

Returns 0 if successful, otherwise the return value is an XPHySpec error code.

5.2. XPHySpecTunableFilter Properties

Type	Name	Description
double	BraggAngle	angle of incidence with the Bragg in degrees
double	CurrentWavelength	current wavelength in nanometres
int	GratingIndex	index of the current grating
double	GratingPeriod	period of the current grating in nanometres
int	MotorizedLensPosition	current position of the motorized lens in micrometres
int	NumberOfGratings	total number of gratings present in the device
BSTR	SerialNumber	serial number of the device
double	StageAngle	current position of the stage in degrees
int	Status	indicate if the device is initialized (`IDLE = 0`)
double	Temperature	internal device temperature in degree Celsius

6. XPHySpecHImager Object

This specialized object inherits all of `XPHySpecTunableFilter` methods and properties and adds the ones described below.

6.1. XPHySpecHImager Methods

6.1.1. CalibrateFromFile

Calibrate a hyperspectral imager with the given calibration matrix file.

```
int CalibrateFromFile(BSTR p_filename,
                      int p_width,
                      int p_height)
```

Parameters

`[in] p_filename:`	location of the calibration matrix to use
`[in] p_width:`	width of the given matrix
`[in] p_height:`	height of the given matrix

Return value

Returns 0 if successful, otherwise the return value is an XPHySpec error code.

7. XPHySpecLLTunableFilter Object

This specialized object inherits all of `XPHySpecTunableFilter` methods and properties and adds the ones described below.

7.1. XPHySpecLLTunableFilter Methods

7.1.1. CalibrateFromFile

Calibrate a laser line tunable filter with the given calibration vector file.

```
int CalibrateFromFile(BSTR p_filename)
```

Parameters

`[in] p_filename:`	location of the calibration vector to use

Return value

Returns 0 if successful, otherwise the return value is an XPHySpec error code.

7.1.2. CalibrateGrating_1

Calibrate a grating manually by giving a stage position that matches a known wavelength.

```
int CalibrateGrating_1(int p_gratingIndex,
                       double p_lambda,
                       double p_realAngle)
```

Parameters

`[in] p_gratingIndex:`	grating to apply the offset to
`[in] p_lambda:`	theoretical wavelength in nanometres
`[in] p_realAngle:`	real stage angle in degrees

Return value

Returns 0 if successful, otherwise the return value is an XPHySpec error code.

8. XPHySpecTunableLaserSource Object

This specialized object inherits all of `XPHySpecLLTunableFilter` methods and properties and adds the ones described below.

8.1. XPHySpecTunableLaserSource properties

Type	Name	Description
`VARIANT_BOOL`	`LampState`	state (on or off) of the lamp

9. XPHySpecTunableSource Object

This specialized object inherits all of `XPHySpecLLTunableFilter` methods and properties and adds the ones described below.

9.1. XPHySpecTunableSource Methods

9.1.1. MoveSrcLampMirror

Move the mirror to choose the source lamp. The positions are in the configuration file. This method generates a `SrcLampMirrorMoved` event upon completion.

```
int MoveSrcLampMirror(double p_MirrorAngle)
```

Parameters

`[in] p_MirrorAngle:` angle in degrees

Return value

Returns 0 if successful, otherwise the return value is an XPHySpec error code.

9.1.2. RequestPhotodiodeAmplifierValue

Gives the value read by the photodiode. This method generates a `PhotodiodeAmplifierValue` event with the current value in parameter.

```
int RequestPhotodiodeAmplifierValue()
```

Parameters

None.

Return value

Returns 0 if successful, otherwise the return value is an XPHySpec error code.

9.1.3. setDeviceFan

Start or stop the fan of the device to eliminate vibration.

```
void setDeviceFan(VARIANT_BOOL p_State)
```

Parameters

`[in] p_State:`	fan state

Return value

None.

9.1.4. setInternalMirror

Choose if the source lamp will be directed to the calibration photodiode or the output.

```
void setInternalMirror(VARIANT_BOOL p_State)
```

Parameters

`[in] p_State:`	true to direct to the calibration photodiode

Return value

None.

9.2. XPHySpecTunableSource properties

Type	Name	Description
`SAFEARRAY(BSTR)`	`AvailableLamps`	list of lamps available to the system
`BSTR`	`CurrentLamp`	lamp currently used as source
`double`	`CurrentLampMirrorAngle`	position of the mirror in degrees
`double`	`LampPosition`	position of the lamp in degrees
`VARIANT_BOOL`	`LampState`	state (on or off) of the lamp

10. XPHySpecCamera Object

This object is not directly creatable by the user; it is obtained through the `GetCameraInterface` method. All methods and properties of this object are copied for each specialized sub-object: `XPHySpecApogee` and `XPHySpecPCO`. The user should always convert to the more specialized sub-object corresponding to its camera when he needs to use a method, even if the method is available in a more generic sub-object.

10.1. XPHySpecCamera Methods

10.1.1. GetBinning

Give the binning size for both axes.

```
void GetBinning(int *p_horizontal,
                int *p_vertical)
```

Parameters

`[out] p_horizontal:`	horizontal binning in pixels
`[out] p_vertical:`	vertical binning in pixels

Return value

None.

10.1.2. GetDetectorSize

Give the detector size.

```
void GetDetectorSize(int *p_width,
                     int *p_height)
```

Parameters

`[out] p_width:`	detector width in pixels
`[out] p_height:`	detector height in pixels

Return value

None.

10.1.3. GetExposureRange

Give the supported range of exposure times.

```
void ExposureValidRange(double *p_minimum,
                        double *p_maximum)
```

Parameters

`[out] p_minimum:`	lower boundary in seconds
`[out] p_maximum:`	higher boundary in seconds

Return value

None.

10.1.4. GetPixelSize

Give the pixel physical size on the detector.

```
void GetPixelSize(double *p_width,
                  double *p_height)
```

Parameters

`[out] p_width:`	pixel width in micrometres
`[out] p_height:`	pixel height in micrometres

Return value

None.

10.1.5. GetROISize

Get the size of the selected region of interest.

```
void GetROISize(int *p_ROIWidth,
                int *p_ROIHeight)
```

Parameters

`[out] p_ROIWidth:`	width in pixels
`[out] p_ROIHeight:`	height in pixels

Return value

None.

10.1.6. GetROIStart

Give the origin of the region of interest.

```
void GetROIStart(int *p_x,
                 int *p_y)
```

Parameters

`[out] p_x:`	abscissa in pixels
`[out] p_y:`	ordinate in pixels

Return value

None.

10.1.7. SetBinning

Modify the current binning with the given values. Not all binning combinations are supported for a specific camera. See the user manual of the camera for more information.

```
void SetBinning(int p_horizontal,
                int p_vertical)
```

Parameters

`[in] p_horizontal:`	horizontal binning in pixels
`[in] p_vertical:`	vertical binning in pixels

Return value

None.

10.1.8. SetReadOutResolution

Modify the current bit depth with the given value. Not all bit depths are supported for a specific camera, see the `AvailableReadoutResolutions` property for a list.

```
void SetReadOutResolution(int p_resolution)
```

Parameters

`[in] p_resolution:`	bit depth

Return value

None.

10.1.9. TakeSinglePicture

Acquire an image with the given exposure time.

```
int TakeSinglePicture(double p_ExposureTime)
```

Parameters

`[in] p_ExposureTime:` exposure time in seconds

Return value

Returns 0 if successful, otherwise the return value is an XPHySpec error code.

10.2. XPHySpecCamera Properties

Type	Name	Description
`SAFEARRAY(VARIANT)`	`ReadoutResolutions`	list of all possible bit depths
`SAFEARRAY(VARIANT)`	`ReadoutSpeeds`	list of all possible readout speeds in MHz
`BSTR`	`CameraName`	internal name of the camera
`VARIANT_BOOL`	`CoolerAvailable`	indicate the presence of a cooler
`SAFEARRAY(BYTE)`	`LastImageTaken`	contain the last image taken by the camera (16-bit aligned)
`int`	`Status`	indicate if the camera is exposing (`IDLE = 0`)
`double`	`Temperature`	Internal camera temperature in degree Celsius

11. XPHySpecApogee Object

This specialized object inherits all of `XPHySpecCamera` methods and properties and adds the ones described below.

11.1. XPHySpecApogee Methods

11.1.1. CloseShutter

Close the shutter.

```
int CloseShutter()
```

Parameters

None.

Return value

Returns 0 if successful, otherwise the return value is an XPHySpec error code.

11.1.2. OpenShutter

Open the shutter.

```
int OpenShutter()
```

Parameters

None.

Return value

Returns 0 if successful, otherwise the return value is an XPHySpec error code.

11.1.3. SetCoolerOn

Control the cooling status.

```
void SetCoolerOn(VARIANT_BOOL p_On)
```

Parameters

`[in] p_On:`	true if cooler is active

Return value

None.

11.2. XPHySpecApogee Properties

Type	Name	Description
`double`	`CoolerTemperature`	current temperature of the camera cooler in degree Celsius
`double`	`CurrentReadOutSpeeds`	current readout speed in MHz
`unsigned int`	`Gain`	current gain
`unsigned int`	`Offset`	current offset

12. XPHySpecPCO Object

This specialized object inherits all of `XPHySpecCamera` methods and properties and adds the ones described below.

12.1. XPHySpecPCO Properties

Type	Name	Description
`VARIANT_BOOL`	`CoolerOn`	state of the cooler
`double`	`CoolerTemperature`	current temperature of the camera cooler in degree Celsius
`VARIANT_BOOL`	`IRSensitivity`	activate the PCO IR sensitivity mode

13. Error Codes

The table below describes all errors that can occur when a method fails. It is always possible to see a more detailed description of the error in the `physpec.log` file located in PHySpec root directory.

Code	Name	Description
-1001	PERROR_FAILURE	general failure
-1002	PERROR_NO_MEMORY	object could not be created because of lack of memory
-1003	PERROR_BAD_PARAM	bad parameter was given to a method or property
-1004	PERROR_BAD_STATE	system is in bad state
-1005	PERROR_OPEN	error occurs while opening a file
-1006	PERROR_READ	error occurs while reading a file
-1007	PERROR_WRITE	error occurs while writing into a file
-1008	PERROR_CLOSE	error occurs while closing a file
-1009	PERROR_SEEK	error occurs while seeking information in a file
-1010	PERROR_NOT_FOUND	object is not available or does not exist
-1011	PERROR_OUT_OF_BOUNDS	over the allowed boundaries
-1012	PERROR_NOT_IMPLEMENTED	method is not supported by the device
-1013	PERROR_REFERENCED	attempt to dispose of something that is still in use
-1014	PERROR_BAD_FORMAT	bad format
-1015	PERROR_NULL_VALUE	null value parameter detected
-1016	PERROR_BAD_ALLOC	bad memory allocation
-1017	PERROR_EMPTY	attempt to access data in an empty structure
-1018	PERROR_NOT_INITIALIZED	attempt to access a non initialized object
-1019	PERROR_NOT_CALIBRATED	attempt to access a non calibrated object
-1021	PERROR_ALREADY_EXIST	attempt to add something that already exists
-1022	PERROR_HOLE_FOUND	specified spectral range is not accessible
-1023	PERROR_PLUGIN_FAILURE	general failure associated with plug-ins
-1024	PERROR_CONNECT_FAILURE	general failure while connecting events
-1501	PERROR_DEVICE_FAILURE	general device error
-1502	PERROR_DEVICE_NOT_FOUND	device is not detected
-1503	PERROR_DEVICE_COMM_FAILED	failed to communicate with the device
-1504	PERROR_DEVICE_TYPE_UNKNOWN	device type unknown
-1505	PERROR_DEVICE_SOURCE_NOT_FOUND	source module is not found
-1601	PERROR_CAMERA_NOT_INIT	camera is not initialized
-1602	PERROR_CAMERA_NO_MEMORY	there is no memory left in the camera

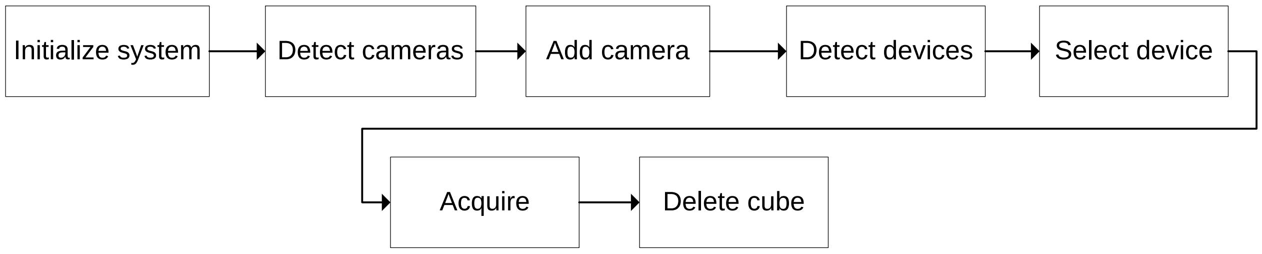
Code	Name	Description
-1603	PERROR_CAMERA_FAILURE	camera general failure
-1604	PERROR_CAMERA_NOT_FOUND	cannot find something requested to the camera
-1605	PERROR_CAMERA_BAD_PARAM	bad input parameter
-1606	PERROR_CAMERA_NOT_READY	camera is busy
-1620	PERROR_CAMERA_COOLER	general error with camera cooler
-1621	PERROR_CAMERA_COOLER_UNSTABILIZED	cooler of the camera is not stabilized
-1622	PERROR_CAMERA_SHUTTER	general camera shutter error
-1623	PERROR_CAMERA_IMAGING_FAILED	camera failed to retrieve an image or to start exposure
-1624	PERROR_CAMERA_OUT_OF_BOUNDS	parameters given to the camera are out of bounds

14. Examples

The following sections describe five typical procedures to execute with Photon etc.'s instruments. They are not meant to be exhaustive, but to help the user understand what can be done with the XPHySpec Interface. It is important to note that XPHySpec ActiveX must be installed and registered on the target computer before any example can be run. Before describing the examples, it is necessary to define the terminology used in XPHySpec. A "**device**" is the filter with Photon etc.'s logo, which can be controlled via USB. When using a hyperspectral imager, the term "**camera**" is referring only to the device put behind the filter. The examples below are described in a generic manner, but there is also concrete implementation made in LabView. All the examples are located in the LabView folder in the PHySpec installation directory.

14.1. Xacquisition

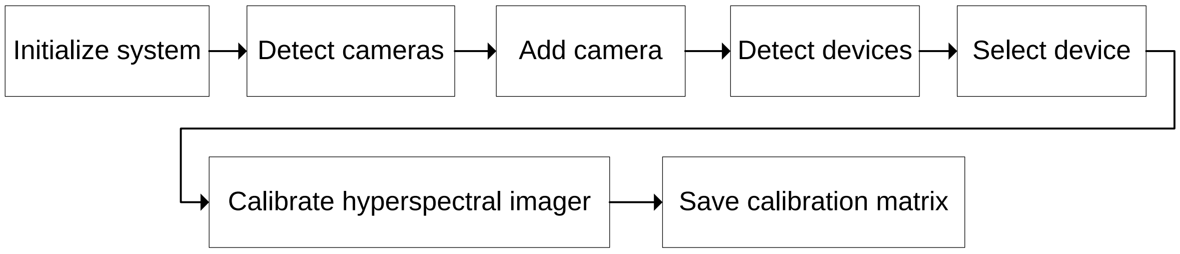
To execute an acquisition sequence, the user must have a hyperspectral imager. The figure below shows the sequence of steps to execute a successful acquisition. An acquisition is the process of taking several images, one at each wavelength, to have a resulting spectral analysis of the scene, which is called "**cube**" in XPHySpec.



These steps assume an `XPhySpec` control is created prior to calling any function. First, the system needs to be initialized to setup internal states and logging facilities. When the system has initialized properly, it is possible to detect and add a camera or to detect and select a device. When both the camera and the device are found and accessible through their objects, respectively `XPHySpecCamera` and `XPHySpecTunableFilter`, an acquisition can be started. An acquisition can take several minutes depending on the exposition time and the wavelength range. The acquisition process generates a cube which is kept in memory until it is deleted or until XPHySpec is closed. The cube can be saved to a file in the FITS format. When the cube is no longer needed, it is good practice to delete it in order to save memory space.

14.2. Xcalibration

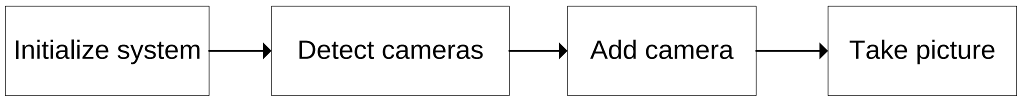
To execute an automatic calibration sequence, the user must have a hyperspectral imager. The figure below shows the sequence of steps to execute a successful calibration. An automatic calibration is the process of taking several images of a known light source to calibrate the response of the hyperspectral imager. The automatic calibration needs a file in which the known emission bands of a calibration light source are described. Each grating needs to be calibrated separately.



The only notable distinctions with the above example are the two last steps. The automatic calibration function is called from the `XPHySpec` control and can take quite some time in order to find a match between the measured values and the ones in the emission bands file. It is important to be aware that an explicit conversion of the `XPHySpecTunableFilter` object to the `XPHySpecHImager` sub-object is needed before calling the `SaveCalibrationMatrices` function. This operation is done by calling `ToHI` from the `XPHySpec` control.

14.3. Xcamera

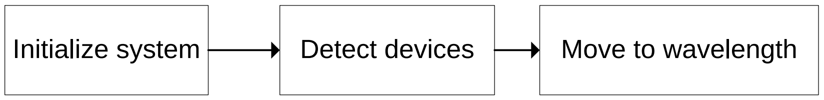
To capture a single image with a hyperspectral imager, the figure below shows the correct sequence.



Taking a single image at a time offers greater flexibility to the user. It gives the user a complete control over his acquisition process and offers him the possibility to do additional work between the snapshots. It is essential to mention that the `TakeSinglePicture` function must be called from the appropriate camera interface. For example, if the camera used is a model from the PCO manufacturer, the interface to use would be `XPHySpecPCO`. Conversion from the generic `XPHySpecCamera` interface to the specific ones can be done with one of the following functions: `ToApogee` or `ToPCO`.

14.4. Xdevice

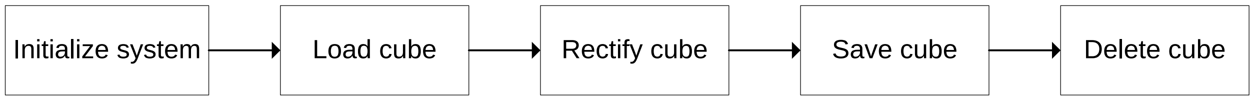
If the user wants a manual control over his filter, the interfaces of each specific device can give him control over the position. The position can be controlled in nanometres or in degrees whichever is more convenient.



The specific device interface offers much information relating to its current state, like its temperature or its position.

14.5. Xrectification

A rectification process can be done with or without an active device; it can be used as a post-processing step. This example uses an existing cube loaded from a file in FITS format.



The rectification is the process of correcting every image in a raw cube to obtain a rectified cube which can then be analysed. It is useful to know that when rectifying a cube, both the raw and the rectified version of the cube exist in memory.