
SynerJY™
SDK

Contents

Introduction

The SynerJY® Software Development Kit (SDK) was created to simplify customers' programming efforts relating to HORIBA Jobin Yvon instrumentation. The SDK is a collection of COM (Component Object Model) components that allows access to instrument data acquisition and control functions across multiple Windows-based development environments. The implementation and access to the COM components will vary due to development environment requirements. The details of implementation in each environment are covered in the Development Environment Overview section.

Each component is designed to interact with a specific class of device. The component classes that are currently implemented are the Monochromator, CCD, Single Channel Detector, and Filter Wheel (type of accessory). Each of these components operates independently. Configuration information for these devices is established on installation. All of the configuration information is stored in the registry and is accessible via programmatic means.

Supported Development Environments

The SynerJY SDK can be used with a variety of programming environments including Microsoft Visual C++, Microsoft Visual Basic, and Microsoft Visual Basic for Applications.

Example programs are contained for each programming language. These programs can be modified to best suite the requirements of your system.

- Microsoft Visual C++ (VC++)
- Microsoft Visual Basic (VB)
- Microsoft Visual Basic for Applications (VBA)

Terminology

COM – Component Object Model is the Microsoft standard for defining software building blocks. See http://www.microsoft.com for a more detailed description.

ActiveX – ActiveX components are primarily components with a visual user interface. All ActiveX components ARE COM components, but all COM components ARE NOT necessarily ActiveX components. The components in this SDK ARE NOT ActiveX components. See http://www.microsoft.com/com/tech/ActiveX.asp.

HRESULT – Standard return value from a COM interface call that indicates either success or failure of a given function. In C++, you can use the Macros SUCCEEDED(hr) and FAILED(hr) to determine success or failure of the given function.

Interface – An interface defines how to call a function within a component. The component implements the functionality that is specified by the interface, and is said to "support" the given interface. This is a basic COM concept; see http://www.microsoft.com/com.

Enumeration - The concept of enumeration is applied on multiple components within the framework. This is a way to provide a simple interface which gives the user access to a list of values. These functions are named GetFirstXXX and GetNextXXX. To initialize access to the list, the user can call the GetFirst function. To access subsequent list values, the user can call GetNext. The list is terminated when the value "-1" is returned for the token.

Token String Pairs – Some functions which enumerate values have output parameters for both an enum value (which is a long value) and a string. The enum/long value is the "Token" and the string is the string representation of the value. Token string pairs provide a string for the user interface and a token for the caller to send back to the component when requesting that an item from the list be selected.

VB – Microsoft Visual Basic

VBA – Microsoft Visual Basic for Applications (included with Excel and other MS Office Applications)

VC++ - Microsoft Visual C++

Getting Started

Installation Requirements

To successfully install the SynerJY SDK, your computer system should be equipped with the following:

Software

Windows 2000 or Windows XP

Hardware

Meets requirements for Windows 2000 or Windows XP

128 MB RAM (256 MB recommended)

200 MB disk space

One free USB port for installation of SynerJY SDK USB hardware key

Installing SynerJY SDK

To install SynerJY SDK:

1. Start Windows if you have not already done so. Make sure all programs are closed.
2. Insert the CD labeled **SynerJY SDK** into your CD-ROM drive. If Autorun is enabled installation will begin automatically. If Autorun is not enabled, execute the **setup.exe** file.
3. If you have a previous version of SynerJY SDK installed, a question dialog box appears. Select **Yes** to uninstall the previous version.
4. The InstallShield Wizard dialog box appears. Click **Next** to display the License Agreement.
5. Read the License Agreement carefully, then click **Yes** to agree to the terms and conditions of the agreement. You must agree to install SynerJY SDK.
6. Enter your name and the name of the company for which you work. Click **Next**.
7. Select a destination location to install SynerJY SDK or click **Next** to accept the default location (**C:\Program Files\Jobin Yvon**) and continue the installation.
8. At the prompt, unplug the SynerJY hardware key (if inserted) from the appropriate USB port of your computer. Click **Next**.
9. Select a restart option. Remove the SynerJY SDK CD from the CD-ROM drive then click **Finish**.

Note: It is necessary to restart your computer following installation. When your computer restarts, the InstallShield Wizard automatically opens the device configuration dialog box, which allows you to create a hardware configuration. You must insert the SynerJY hardware key into a free USB port of your computer in order to start SynerJY SDK.

Common Programming Requirements

The following is a general overview of the programming requirements common to all of the components in the SDK. The example code is nearly identical for all components. For simplicity, Visual Basic is used for all of the example code in the documentation. Functioning examples in VBA and VC++ are also provided for your reference when working in those environments. Getting Started Guides for each development environment have also been provided to assist you.

Declaring and Creating Your Object

When declaring your object in VB, you need to use the **WithEvents** keyword to ensure that VB sets up the object with events enabled.

```
' JY device definition for CCD
Private WithEvents ccdObject As JYMCD

Set ccdObject = CreateObject("JYCCD.JYMCD")
```

Loading the Configuration and Initializing the Component

1. The unique id is the id of the device you are attempting to load. The example programs provide more detail on how to obtain the available/installed id's. 'ccdObject.Uniqueid = "CCD1"
2. 'ccdObject.Load loads the configuration for the specified id.
3. Attempt to establish a connection to the hardware. The SDK will throw exception if the hardware is not present (See the example program for details on how to handle exception and use emulation). ccdObject.OpenCommunications
4. Initialize the component (This command will return immediately. You will receive an "Initialized" event when the device initialization is complete) ccdObject.Initialize

Installation of CCD Hardware Files

All HORIBA Jobin Yvon spectrometers can be integrated with a number of detectors and accessories. Detectors and accessories can be added or removed from the hardware configuration to accommodate a variety of experiment types. Hardware configurations that include CCD detectors require the installation of hardware files before creating a hardware configuration. These files are contained on the Initialization and Setup CD or disk that is provided with your CCD.

Symphony CCD Hardware Files

To install the Symphony CCD hardware files:

1. Start Windows if you have not already done so. Make sure all programs are closed.
2. Insert the CD labeled **Symphony** into your CD-ROM drive. If Autorun is enabled installation will begin automatically. If Autorun is not enabled, execute the **setup.exe** file.
3. The InstallShield Wizard dialog box appears. Click **Next** to display the License Agreement.
4. Read the License Agreement carefully, then click **Yes** to agree to the terms and conditions of the agreement. You must agree to install the Symphony hardware files.
5. Enter your name and the name of the company for which you work. Click **Next**.
6. Select a destination location or click **Next** to accept the default location (**C:\Program Files\Jobin Yvon\Symphony**).
7. Review the current settings then click **Next** to continue the installation.
8. Click **Finish**. Remove the Symphony CD from the CD-ROM drive.

Note: When the Symphony controller is powered on, the system status LEDs located on the front panel of the Symphony Controller blink at a 1 Hz rate, indicating that the Symphony CCD Detection System is awaiting initialization. It is necessary to install the Symphony hardware files, which include the Symphony Hardware Initialization program, in order to create a hardware configuration that includes a Symphony CCD.

CCD 3000, CCD 3500 or IGA 3000 Hardware Files

To install your detector's hardware files:

1. Insert the **Initialization and Setup** disk.
2. From the **Start** menu click **Run**.
3. Type **A:\install.exe** and press **Enter**.
4. A description of the component for which the initialization files are being installed appears; press any key to continue.
5. Using the **Arrow Keys**, select **User Defined** and press **Enter**.
6. Select **Drive C:** and press **Enter**.
7. Type **JobinYvon\CCD** and press **Enter**.
8. Press any key to continue. The files are saved to **Drive C>Program Files>Jobin Yvon**.

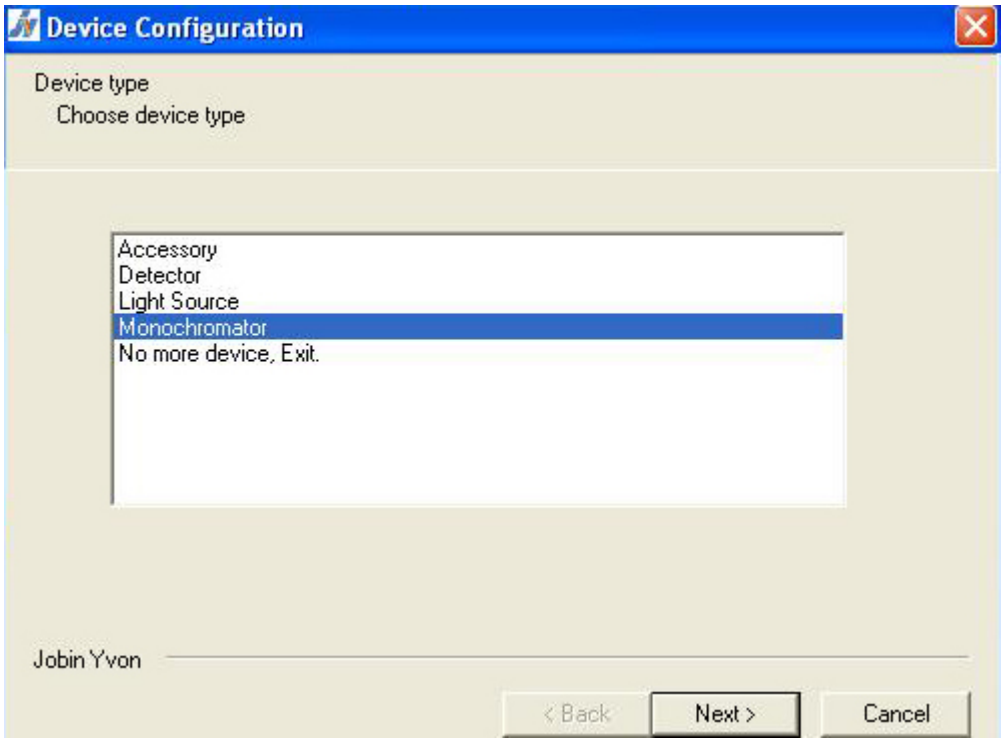
Hardware Configuration

SynerJY SDK allows you to set up many different hardware configurations. Hardware configurations are created and edited via the JY Device Configuration dialog box. Following SynerJY SDK installation, the InstallShield Wizard automatically opens the JY Device Configuration dialog box, which allows you to create a hardware configuration. You can also access the JY Device Configuration dialog box by selecting **Jobin Yvon>SDK>Configure Device** from the **Start** menu.

To create a hardware configuration:

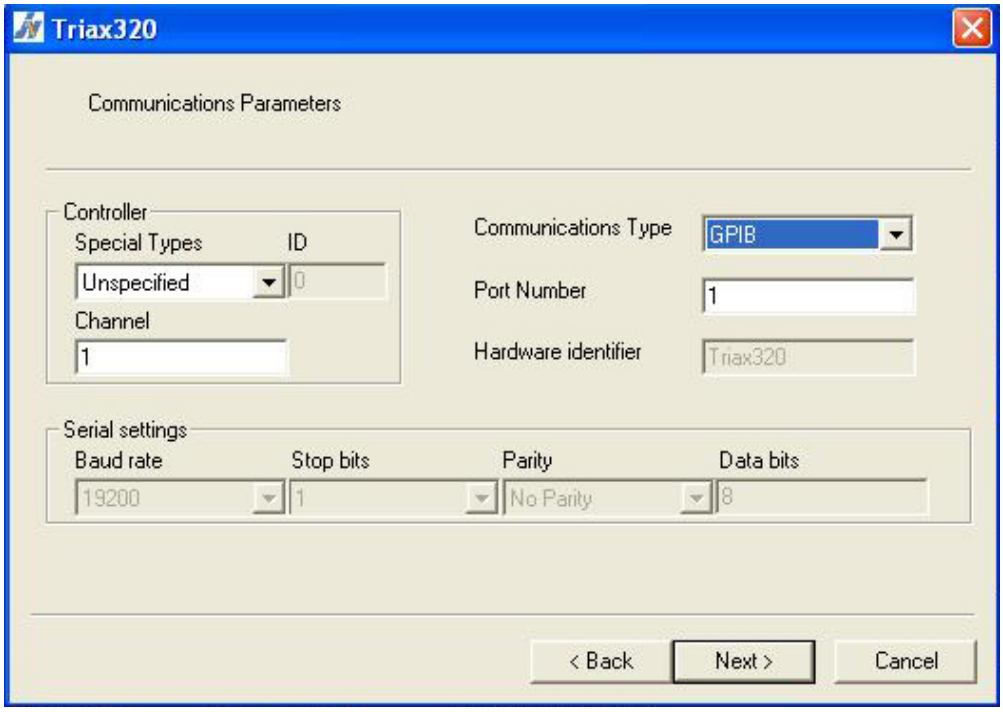
1. Click the **Go** button from the **JY Device Configuration** dialog box.

2. Select a **Device type** from the **Device Configuration** window. Click **Next**.



3. Select the specific type (spectrometer model, detector type, etc.) of device that you are configuring. Click **Next**.

4. Enter the **Communications Parameters** for the device being added. Refer to the Device Configuration Table for specific configuration information. After adding all parameters, click **Next**.



5. A summary of the device configuration displays. Click **Apply** to continue. You will return to the **Device type** list. Select another device if necessary. Once all additions are made, select **No more device, exit**.

6. Click **Yes** to end the device configuration session.

Your hardware should now be configured.

Device Configuration Table

Detector	Controller Type	Communications Type	Port Number	Channel	Baud Rate	Stop Bits	Parity	Data Bits
Lockin	Lockin	GPIB or	8 (Default)	1				
		Serial	Com port #		19,200	2	No Parity	8
CCD3000/3500	CCD3000	GPIB	5	0				
Symphony	Symphony	TCPIP	4321	0				
IGA3000	IGA	GPIB	6	0				
SpectrAcq2	Unspecified	GPIB or	3	1 or 0*				
		Serial	Com port #		19,200	1	No Parity	8

* Channel 0 used for photon counting

Monochromator	Controller Type	Communications Type	Port Number*	Channel	Baud Rate	Stop Bits	Parity	Data Bits
Triax 180/190	Unspecified	GPIB or	1-32	0				
		Serial	Com port #		19,200	1	No Parity	8
Triax 320/322	Unspecified	GPIB or	1-32	0				
		Serial	Com port #		19,200	1	No Parity	8
Triax 500/550	Unspecified	GPIB or	1-32	0				
		Serial	Com port #		19,200	1	No Parity	8
MicroHR	Unspecified	Default	Default	0				
750S	Unspecified	GPIB or	1-32	0				
		Serial	Com port #		19,200	1	No Parity	8
750M	Unspecified	GPIB or	1-32	0				
		Serial	Com port #		19,200	1	No Parity	8
750MI	Unspecified	GPIB or	1-32	0				
		Serial	Com port #		19,200	1	No Parity	8
1000M	Unspecified	GPIB or	1-32	0				
		Serial	Com port #		19,200	1	No Parity	8
1250M	Unspecified	GPIB or	1-32	0				
		Serial	Com port #		19,200	1	No Parity	8

* GPIB default port number = 1

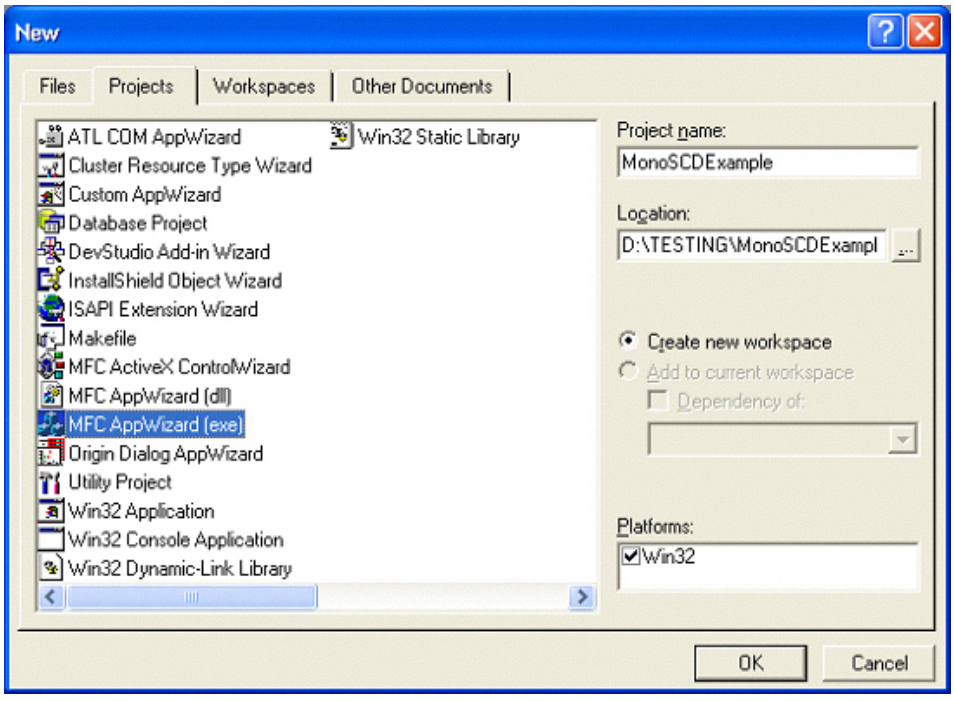
Getting started with Visual C++ 6.0

The following steps are to get you started with Visual C++ 6.0:

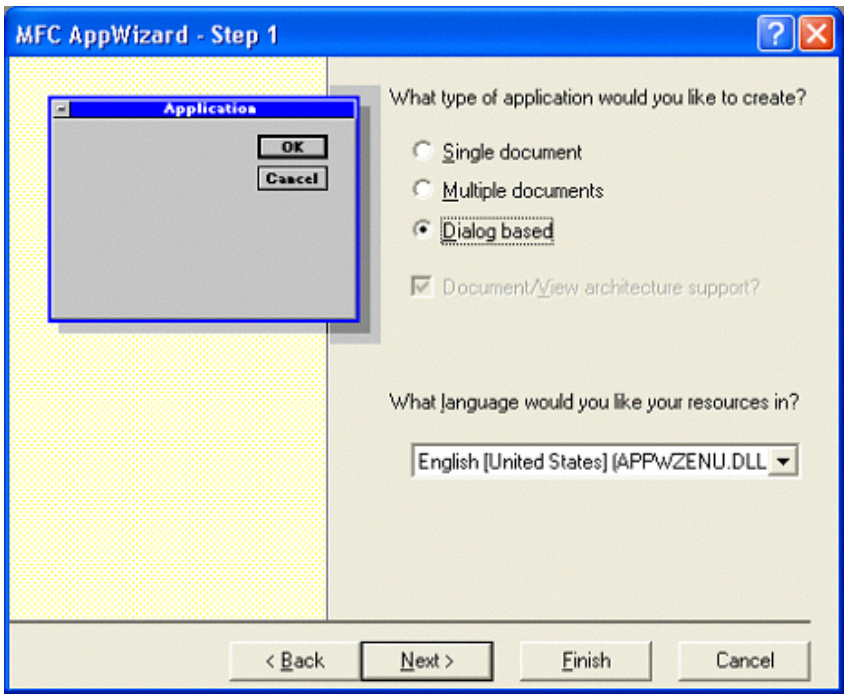
- Step 1. Create a new Dialog-based MFC project
- Step 2. Create an instance of a JY component
- Step 3. Create device sink to handle the events fired by the object
- Step 4. Use JY component in your application

Create a new Dialog-based MFC project

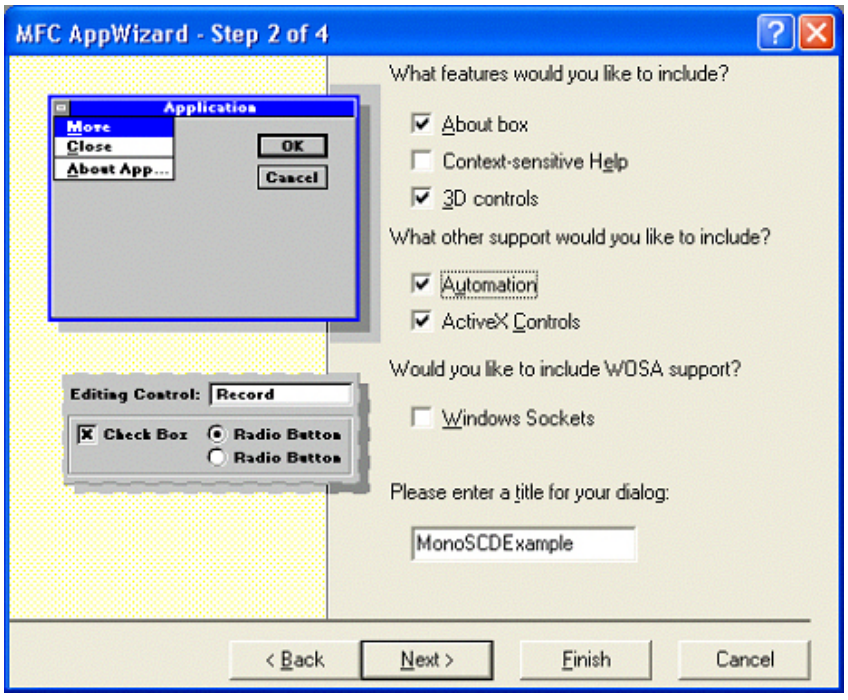
1. Close any currently open workspace by selecting **Close Workspace** from the **File** menu.
2. Select **New** from the **File** menu.
3. Select **MFC AppWizard (exe)** and enter the **Project name** as **MonoSCDExample**.
4. Click **OK** to continue.



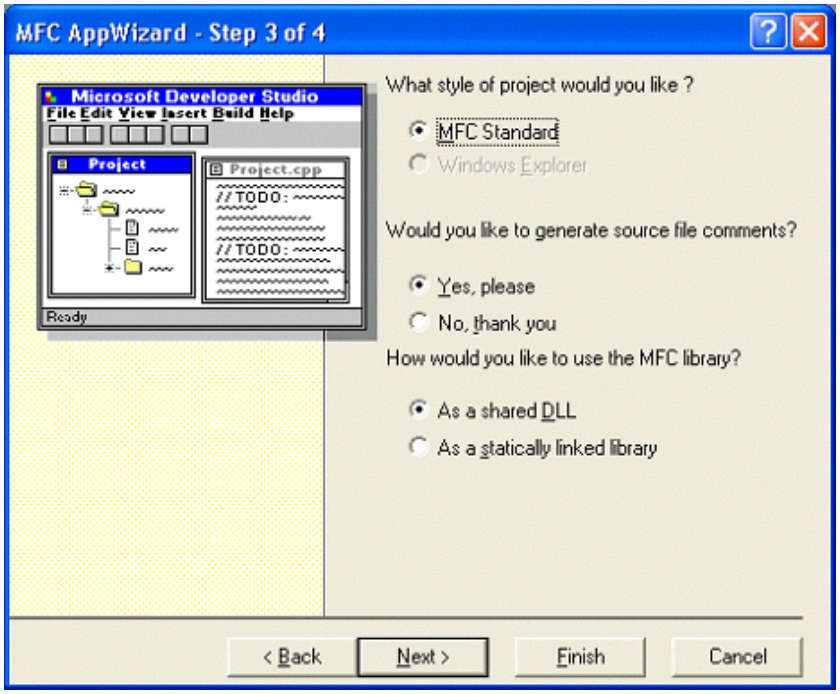
5. Select a **Dialog based** type of application.



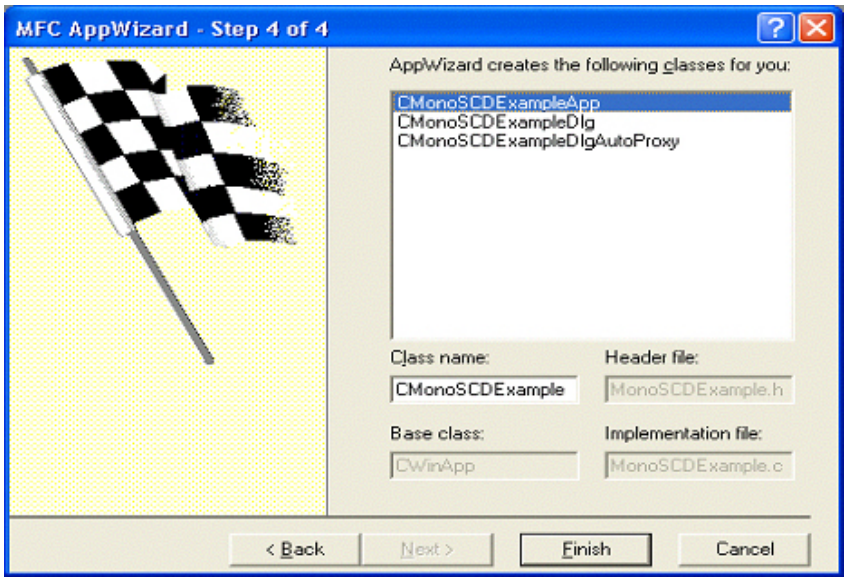
6. Click **Next** to continue.
7. Select **Automation.**



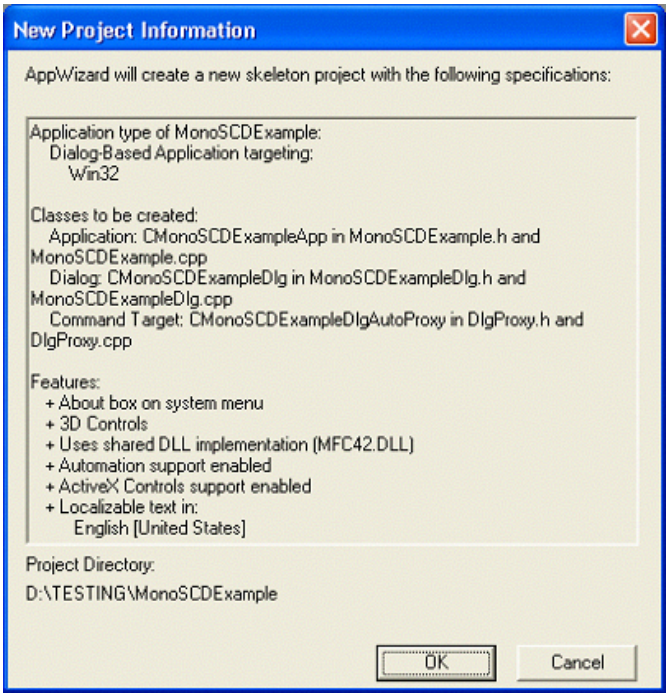
8. Click **Next** to continue.
9. Keep the defaults as they appear in the MFC AppWizard.



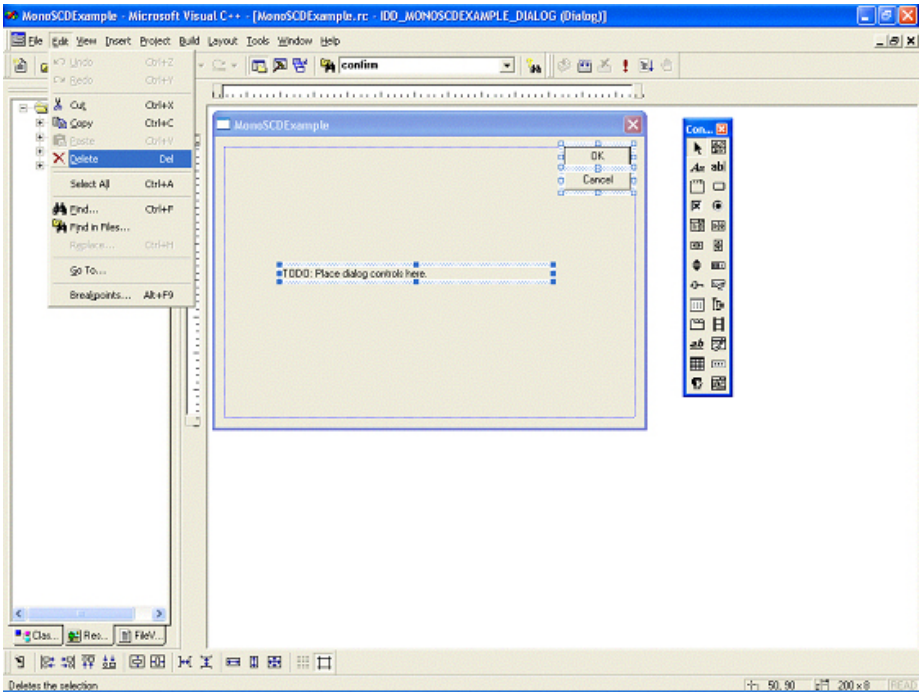
10. Click **Next** to go to MFC AppWizard; keep the defaults in this window as shown below.



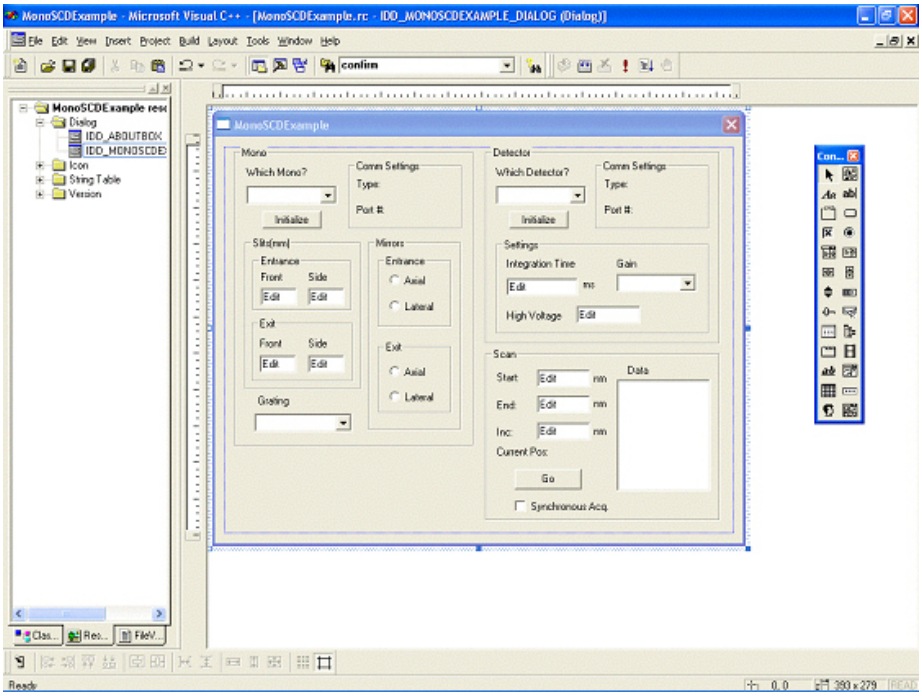
11. Click **Finish** to create the application.
12. Click **OK** to close the information dialog box.



Delete the label and two buttons that were added by the VC++ App Wizard, as shown below.



Add your own controls to the dialog box as you need. For example, to acquire a single channel detector within certain wavelength range, the dialog box may look like the following:



Now you have created your own dialog based MFC application. Next, you need to create instances of JY components in your application and setup device sinks for the objects created.

Create an Instance of a Jobin Yvon (JY) Component in Your Application

1. Import JY type libraries that define the required interfaces.

   In "MonoSCDExampleDlg.h" file, add the following two lines of code right above the dialog class.

```
// Iport the jy type libraries that defines the required interfaces...
#define PATH_TO_JYSYSTEMLIB
"..\..\..\..\JYTypeLibraries\JYSystemLib\Debug\jysystemlib.dll"
#import PATH_TO_JYSYSTEMLIB raw_interfaces_only, raw_native_types,
no_namespace, named_guids

class CMonoSCDExampleDlg : public CDialog

{

….

};
```

Note: You need to find the correct path where JYSystemLib.dll was installed on your computer and replace "PATH_TO_JYSYSTEMLIB" in the #define statement by the correct path.

You can also put these two lines of code into "StdAfx.h" instead of "MonoSCDExampleDlg.h" so that you can have access to JYSystemLib DLL everywhere in this application.

1. Create an instance of the JY component

   For example, to create an instance of JYMono Object, you need to do the following.

- Add a variable in "MonoSCDExampleDlg.h" file, and initialize it to NULL in the constructor of the dialog class.

```
class CMFC_MonoSCDExampleDlg : public CDialog
{
…
private:
IJYMonoReqd *m_jyMono;
…
};
CMFC_MonoSCDExampleDlg::CMFC_MonoSCDExampleDlg(CWnd* pParent
/*=NULL*/)
: CDialog(CMFC_MonoSCDExampleDlg::IDD, pParent)
{
…
m_jyMono = NULL;
…
}
```

- In the "MonoSCDExampleDlg.cpp", create an instance of the JY component at the place where the component is first used.

```
if ( m_jyMono == NULL)
{
hr= CLSIDFromProgID( L"JYMono.Monochromator", &clsid);
if ( FAILED(hr = CoCreateInstance( clsid, NULL, CLSCTX_ALL,
__uuidof( IJYMonoReqd ), (void **)&m_jyMono )))
{
TRACE( "Failed to create Mono Object. Err: %ld", hr );
return;
}
…
}
```

  After creating the instance of the JY component, you need to create a device sink for the object using CJYDeviceSink class.

Note: CJYDeviceSink class files "JYDeviceSink.cpp" and "JYDeviceSink.h" can be copied from any C++ example provided by JY.

Create a device sink

1. Add a variable of type "CJYDeviceSink" in "MonoSCDExampleDlg.h".
2. Initialize it right after the object is created.

   This provides a facility for handling events fired by the object. To do this, you need to add ATL support to your MFC project

Add ATL support to MFC project as follows:

1. From the "Class View" of the MFC project you created, right-click on "MonoSCDExample classes" and select "New ATL Object.
2. Answer "Yes" to the prompt about supporting ATL.
3. Click "Cancel" in the resulting dialog box (object selection).

In order to provide the appropriate callbacks in your Dialog class, the "CJYDeviceSink" class must also be modified to support the appropriate "Parent" of this sink by modifying the constructor to take the Dialog class as the input parameter.

Next, you need to implement the functions required to listen to the device events. There are four callback functions implemented as part of the CJYDeviceSink implementation:

void ReceivedDeviceInitialized (long status, IJYEventInfo *eventInfo);

void ReceivedDeviceUpdate (long status, IJYEventInfo *eventInfo);

void ReceivedDeviceStatus (long status, IJYEventInfo *eventInfo);

void ReceivedDeviceCriticalError (long status, IJYEventInfo *eventInfo);

Each of them takes two parameters. One is event-depended status. One is JYEventInfo object that contains robust information about the event.

You must provide implementations for all these four functions in your Dialog class. For example, when the sink received the "Initialized" event, it will call ReceivedDeviceInitialized(…) function in your Dialog class. This is where you would handle any post-initialization setup of your application.

Now you are ready to use JY component in your application.

Use JY component in your application

For example, to initialize a Mono, you can simply do the following:

1. Tell the Mono object you just created the Unique ID
   m_jyMono->Put_Uniqueid((CComBSTR)monoID);
2. Load this Mono
   m_jyMono->Load();
3. Do OpenCommunications
   m_jyMono->OpenCommunications();

4. Initialize the mono
   m_jyMono->Initialize();

Since you have the device sink setup already, after the mono has been initialized, the device sink will receive “Initialized” event. Then the device sink will call the ReceivedDeviceInitialized(…) function in your dialog class. You can simply insert any post-initialization setup here, in this function. For example, after the mono has been initialized, you would like to know the current mono position.

Getting Started with VBA

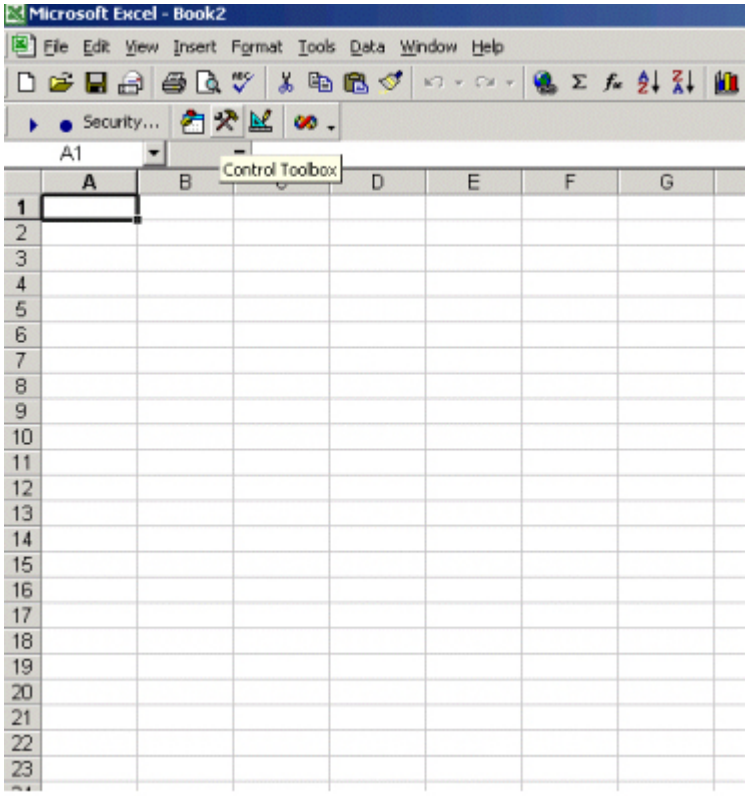
Use the following steps to get you started with VBA:

- Step 1. Create an Excel application user interface
- Step 2. Add an instance of a JY component
- Step 3. Use the JY component in your application

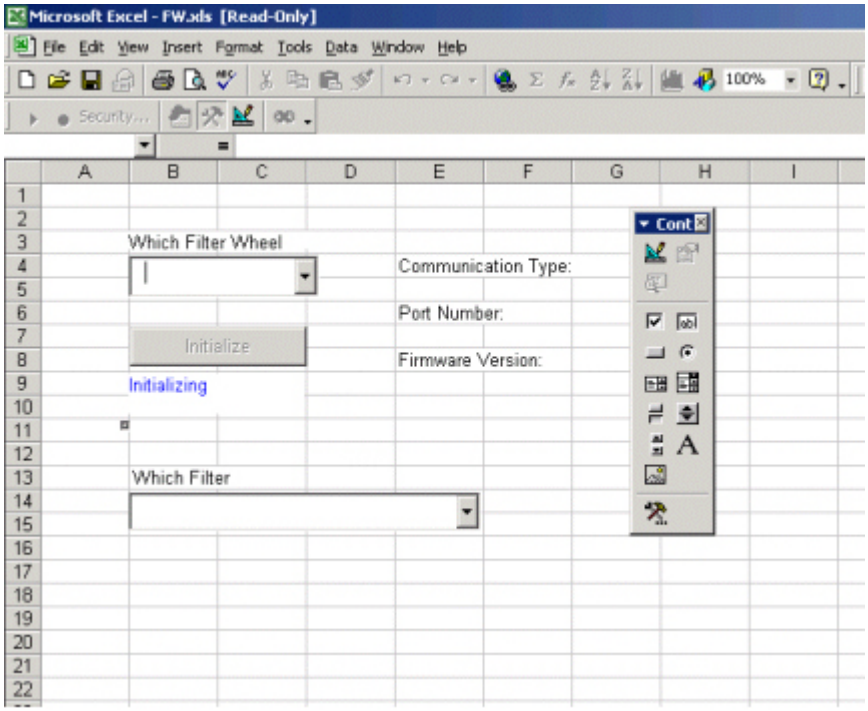
Create an Excel application user interface

1. Open Microsoft Excel. Right-click on the toolbar to check **Visual Basic**. The Visual Basic buttons will appear on the toolbar.

2. Click the **Control Toolbox** button..

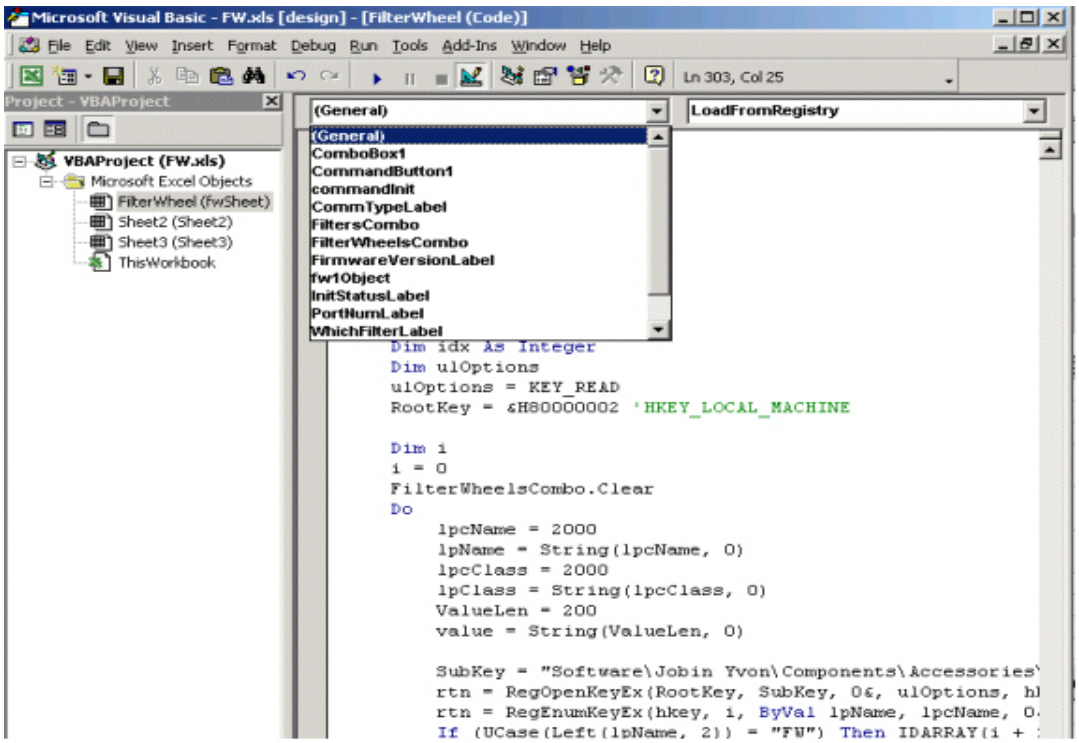


Use the Control Toolbox to **add** control items to the worksheet. The following is an example of a filter wheel application:



3. **Save** the applications as FW.xls.

4. Open **Visual Basic Editor** (VBE) by clicking the toolbar button. In VBE, add **JYSystemLib** and **JYFilterWheel** components using **Tools>References**. In this example JYFilterWheel is used as an example JY component. The interfaces for JYFilterWheel are defined in JYSystemLib.



5. After checking all components you will be using, click **OK**.

   In the declarations section of your code, add the following for each type of object that you will access:

   ```
   Private WithEvents Mono1 As Monochromator
   Private WithEvents SCD1 as JYSCD
   ```

   (This needs to be within the scope of the code from which you will access the objects).

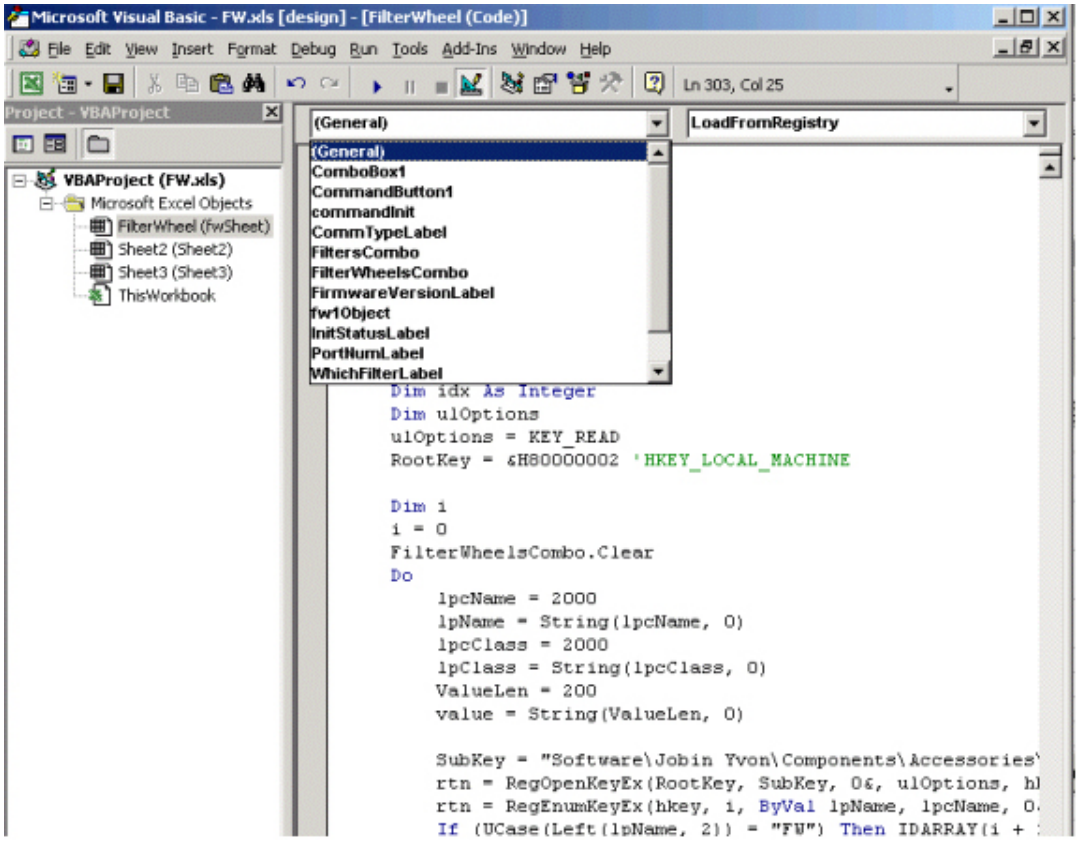
   In the **Form Load** section of your form, add the following for each type of object:

   ```
   Set Mono1 = CreateObject("JYMono.Monochromator")
   Set SCD1 = CreateObject("JYSCD.JYSCD")
   ```

Handling JY events in your application

There are some events that will be put into your Visual Basic code that you need to handle for each device object. They are:

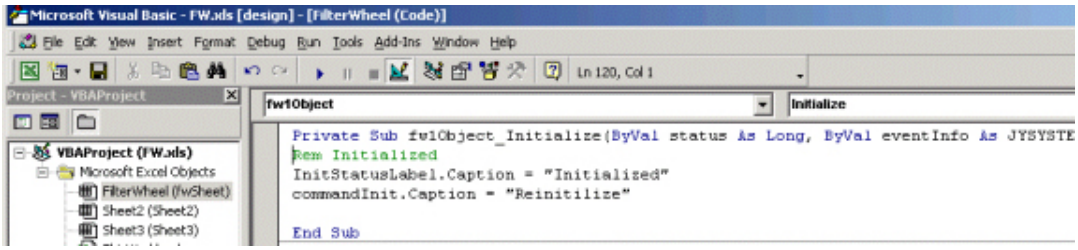
1. **User interface item events**



2. **Initialize** - When sending the initialize command to the device objects, you will get an immediate response from the command, but the actual initialization may take some time (depending on the hardware). It is important that you handle this event so that you know when the actual

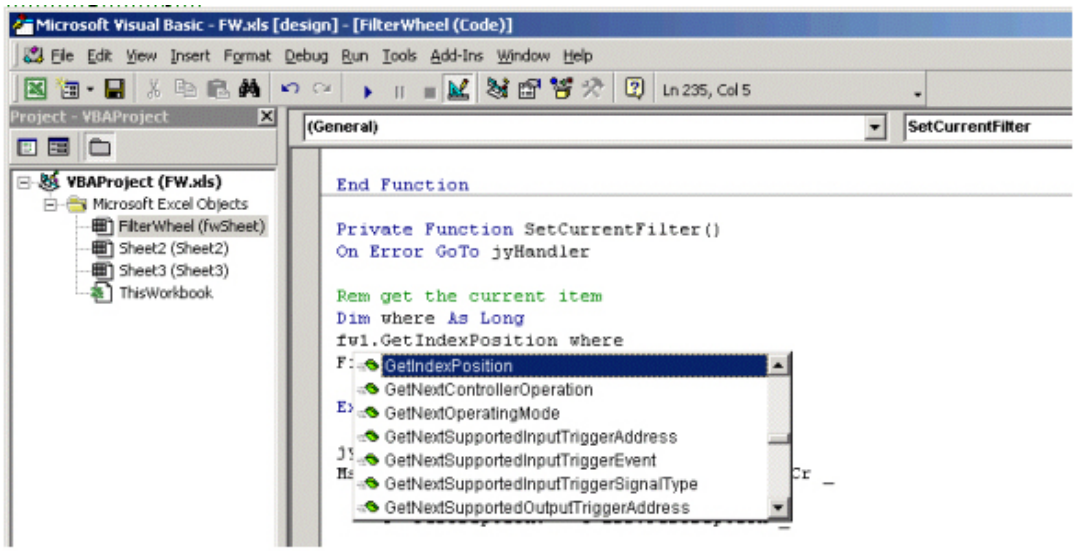
hardware initialization is complete. You can use this event to then enable other controls in your form. For example, you would not want **Mono Move** to be enabled until **Mono Initialize** is complete.

3. **Operation Status** – Gives information about current status of object.
4. **Update** – Asynchronous data collection must use this event to get data.



Use the JY component in your application

The component's interface is exposed in Visual Basic Editor. The methods and properties can be used by accessing the object.



Getting Started with Visual Basic 6.0

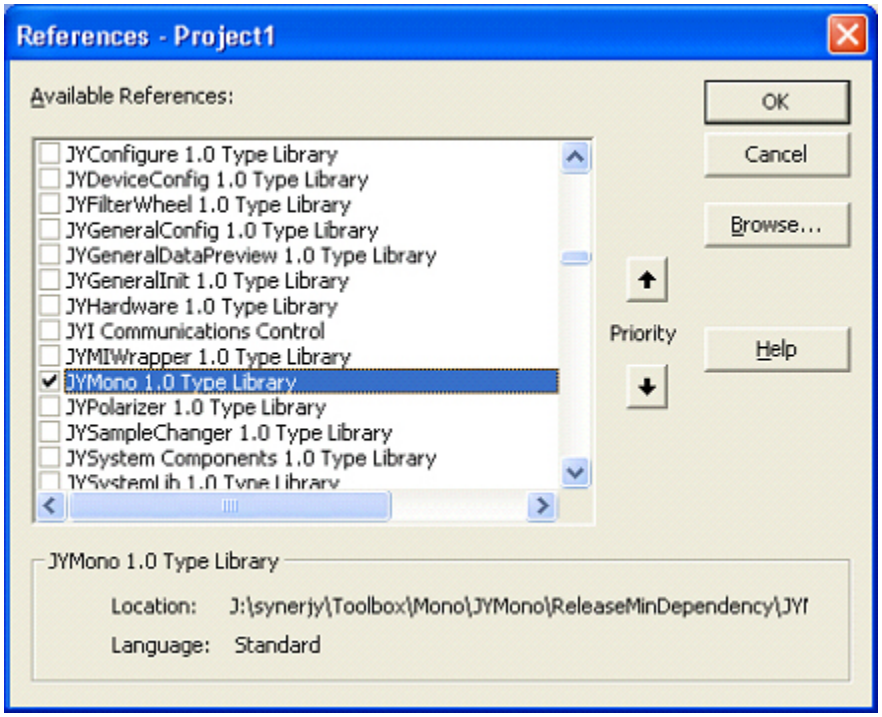
Use the following steps to get started with Visual Basic 6.0:

- Step 1. Create a Form
- Step 2. Add an instance of a JY component
- Step 3. Use the JY component in your application

Create a Form

1. Open **Visual Basic** and **Create a New Project** as a **Standard EXE** by clicking **File>New Project>Standard EXE**. Click **Open**.

2. Select **Project>References** to install Jobin Yvon (JY) components. Check all components that you will be using (this list can be added to later). In the example below, the Mono Component is going to be used. All Jobin Yvon components should start with the letters "JY."



3. After selecting all components that you will be using, click **OK**.
4. In the declarations section of your code, add the following for each type of object that you will access:

   ```
   Private WithEvents Mono1 As Monochromator
   Private WithEvents SCD1 as JYSCD
   ```

   (This needs to be within the scope of the code from which you will access the objects).
5. In the **Form Load** section of your form, add the following for each type of object:

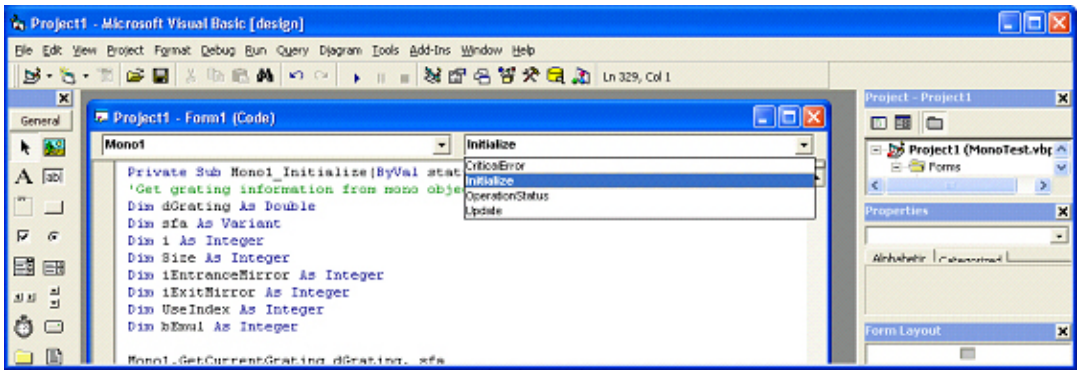
   ```
   Set Mono1 = CreateObject("JYMono.Monochromator")
   Set SCD1 = CreateObject("JYSCD.JYSCD")
   ```

Handling JY events in your application

There are some events that will be put into your Visual Basic code that you need to handle for each device object. They are:

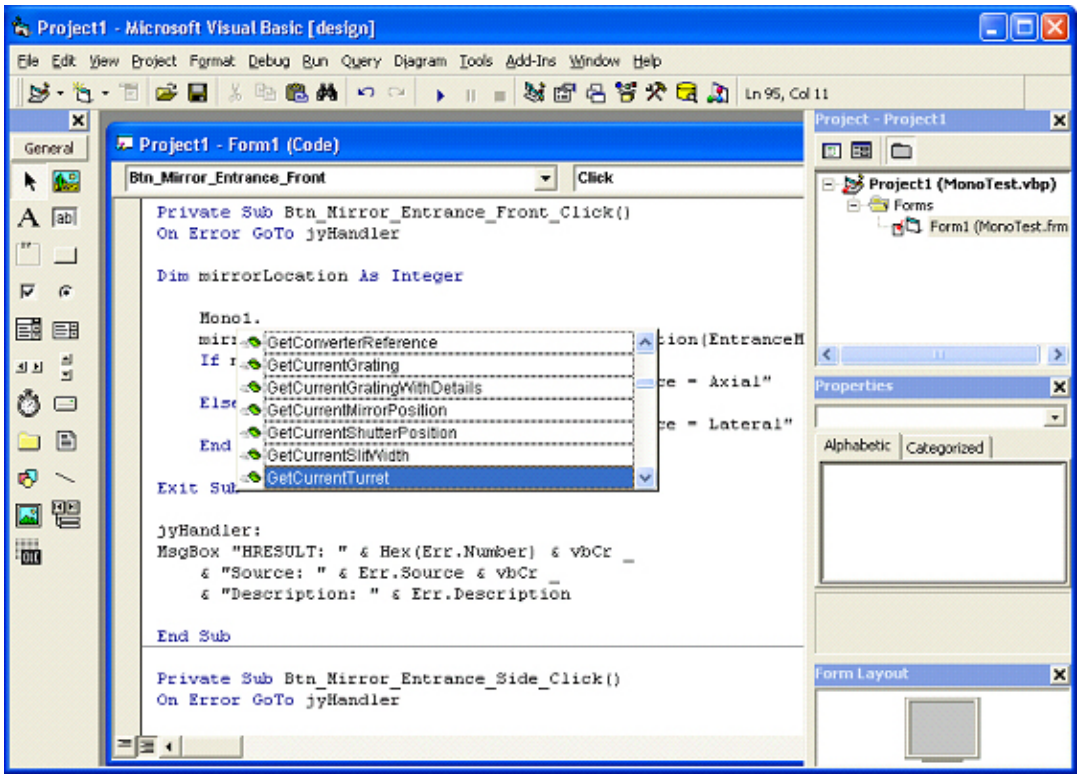
1. **Critical Error** – Any critical error would be reported through this event.
2. **Initialize -** When sending the initialize command to the device objects, you will get an immediate response from the command, but the actual initialization may take some time (depending on the hardware). It is important that you handle this event so that you know when the actual hardware initialization is complete. You can use this event to then enable other controls in your form. For example, you would not want "Mono Move" to be enabled until "Mono Initialize" is complete.
3. **OperationStatus** – Gives information about current status of object.

Update – Asynchronous data collection must use this event to get data.

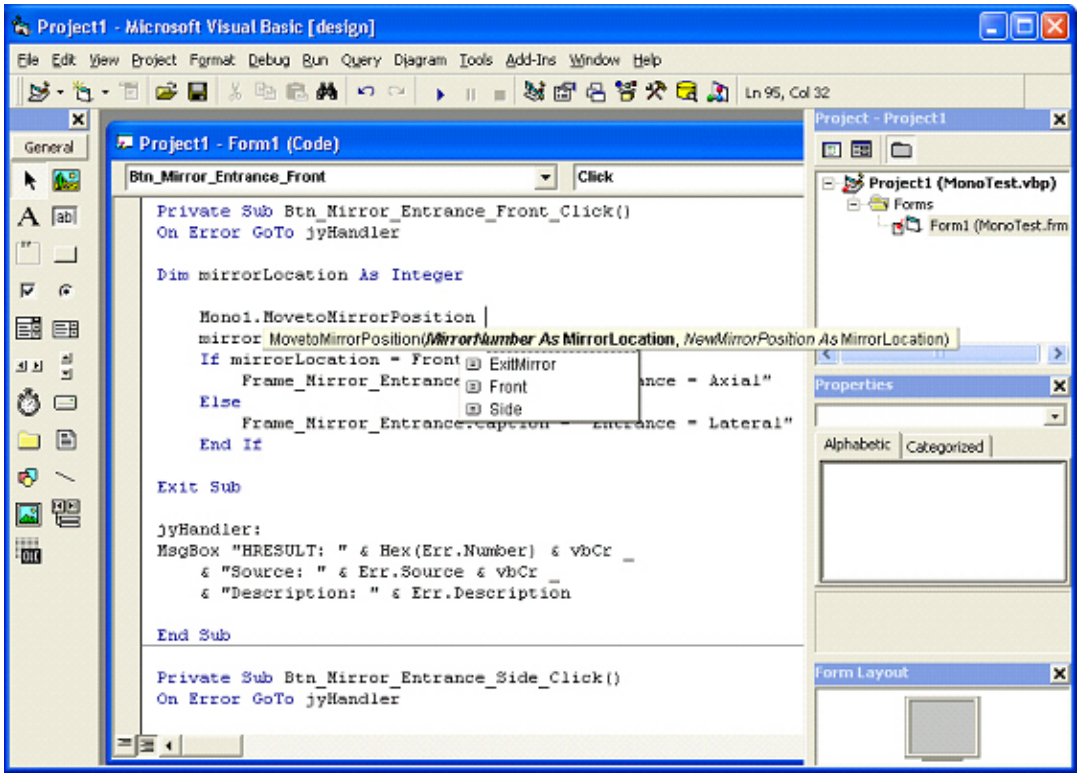


Use the JY component in your application

Now, you are ready to access the object. Once you type your object's name and a dot in Visual Basic, you will get a list of methods for that object.



After typing the `(` or space you will see the list of arguments and any enumerations for that method.



For example, to initialize a Mono, you can do the following…

1. Tell the Mono object you just created the Unique ID
   Mono1.Uniqueid = monoUID
2. Load this Mono
   Mono1.Load
3. Open Communications
   Mono1.OpenCommunications
4. Initialize the mono
   Mono1.Initialize False, False
5. Wait for Initialize Event to Occur
6. Enable other Controls in your form that allow other mono methods to be available.

Some Suggestions while using Visual Basic

- Always set "Option Explicit" in your declarations.
- After causing a hardware action, check .IsBusy to be sure that action is complete.
- If you are in a tight loop checking status (IsBusy), be sure to put in a DoEvents call. This allows windows to multitask.

Example:

```
Mono1.MovetoWavelength Val(Text_Position.Text)
While (Mono1.IsBusy = True)
DoEvents
Wend
Text_Position.Text = Mono1.GetCurrentWavelength
```

- Be cautious when setting the values of radio buttons, since they will also cause events to occur. You may end up calling methods when you do not expect to.

Running Example Programs

SynerJY SDK contains example programs that can be compiled and run in three different languages: Visual Basic (VB), Visual C++ (VC++), and Visual Basic for Applications (VBA). These example programs can be modified to meet the specific needs of your system. To run an example program select **Jobin Yvon>SDK** from the **Start** menu. Select the language you are working in and then select the component for which you want to run the example program.

Example Specifications are provided for the following:

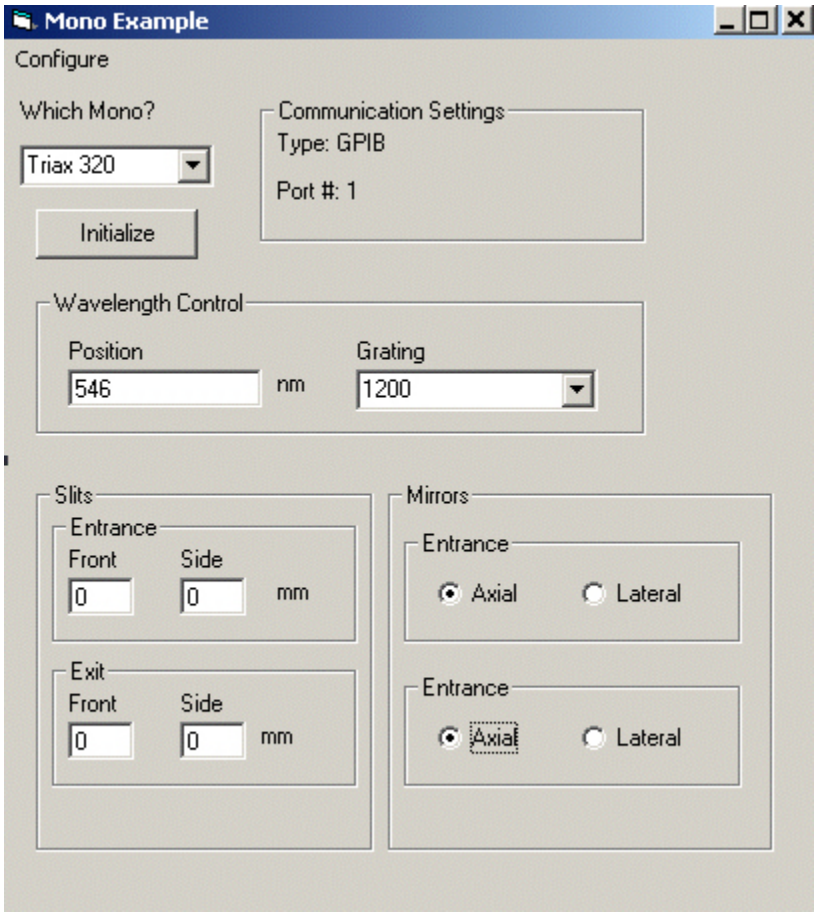
- Mono
- SCD
- CCD - Simple Acquisition
- CCD - Time Based Acquisition
- Mono + SCD
- Mono + CCD

Examples Specification

General

- All examples must NOT use any third party controls
- All source code must be clearly and fully documented
- Non-Component related code should be minimized as much as possible
- Separate component related code from non-component related code wherever possible
- KEEP IT SIMPLE
- Multiple Component Operations should be broken into VB functions/procedures wherever possible
- All functions/procedures and controls should be renamed to make sense (do not use "Button1" or other default identifiers)

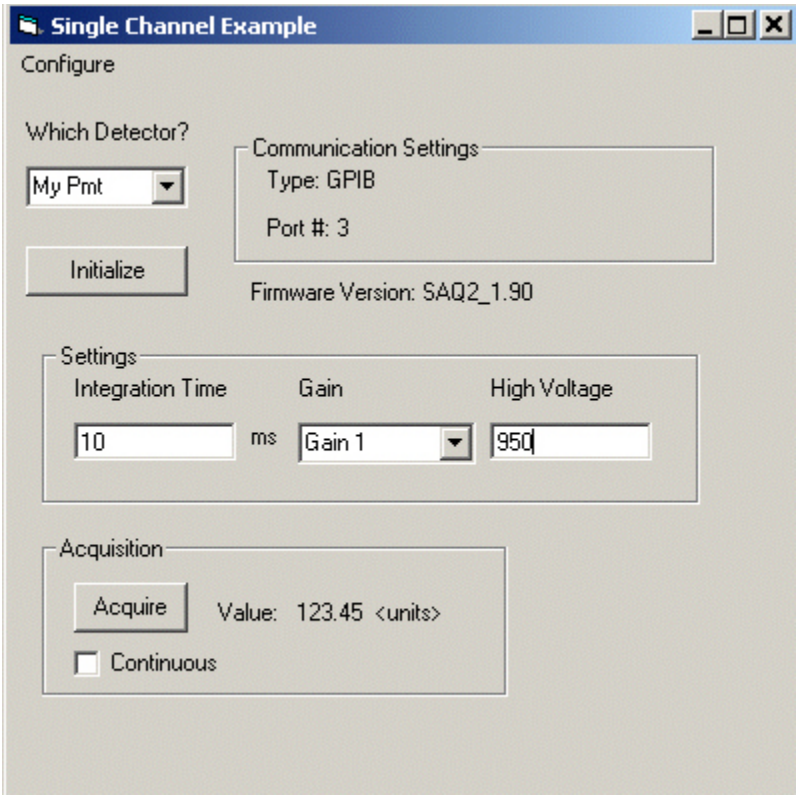
Mono Example



Features:

- Shows monos to select by looking in the registry
- Initializes the Mono, prompting for emulation if hardware is not found
- All controls are disabled until initialization is successful
- Initialize button changes to "Reinitialize" after one successful initialization (Reinitialize will use the "Force" flag to do a complete reinitialization)
- Devices that are not present (ie. – side exit slit) should be disabled/greyed
- Command should be sent for a control based on lost focus or Enter key pressed

SCD Example

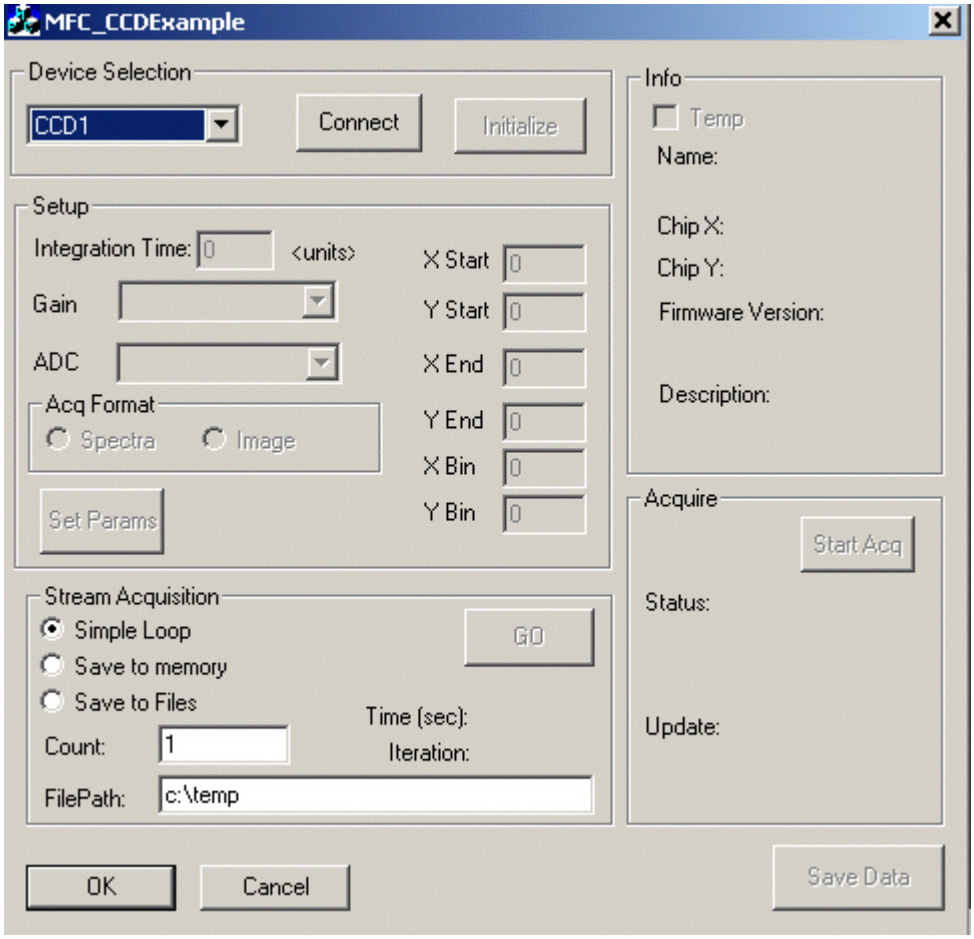


Features:

- Shows detectors to select by looking in the registry
- Initializes the detector, prompting for emulation if hardware is not found
- All controls disabled until initialization is successful
- Initialize button changes to "Reinitialize" after one successful initialization (Reinitialize will use the "Force" flag to do a complete reinitialization)
- Devices that are not present (ie. – voltage) should be disabled/greyed

- Command should be sent for a control based on lost focus or Enter key pressed
- Asynchronous acquisition mode should be used
- Code should give example of Synchronous acquisition

CCD Example – Simple Acquisition



Features:

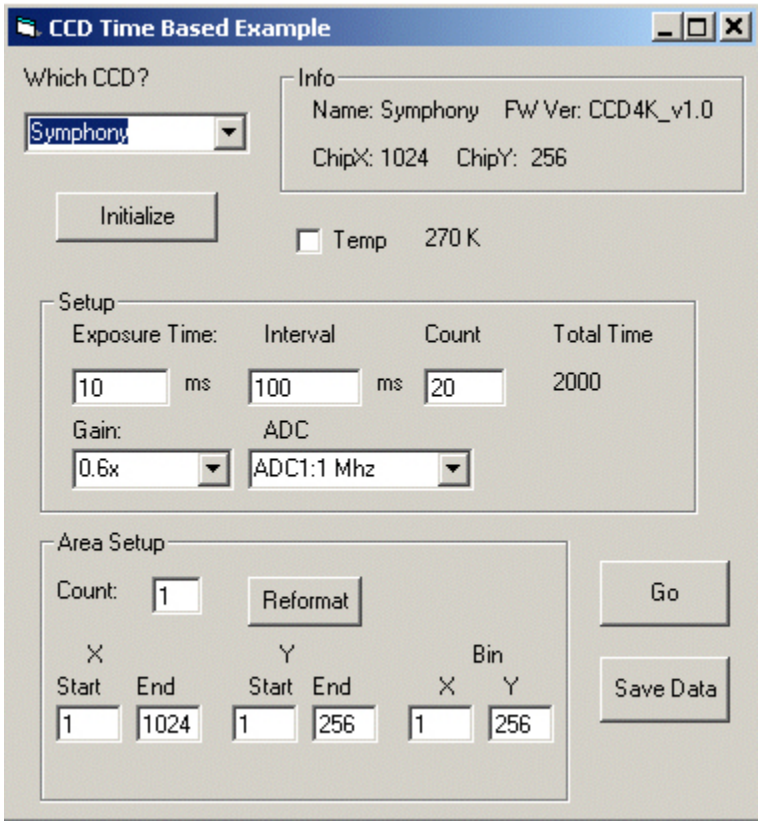
- Shows detectors to select by looking in the registry
- Initializes the detector, prompting for emulation if hardware not found
- All controls disabled until initialize is successful
- Initialize button changes to "Reinitialize" after one successful initialization (Reinitialize will use the "Force" flag to do a complete reinitialization)
- Command should be sent for a control based on lost focus or Enter key pressed
- Asynchronous acquisition mode should be used
- Code should give example of Synchronous acquisition

- In addition to what is shown, multiple areas should be supported

  If number of areas is > 1, chip will be split equally over entire chip

  If number of areas = 1, then the user will be able to specify all area parameters.

CCD Example – Time Based Acquisition



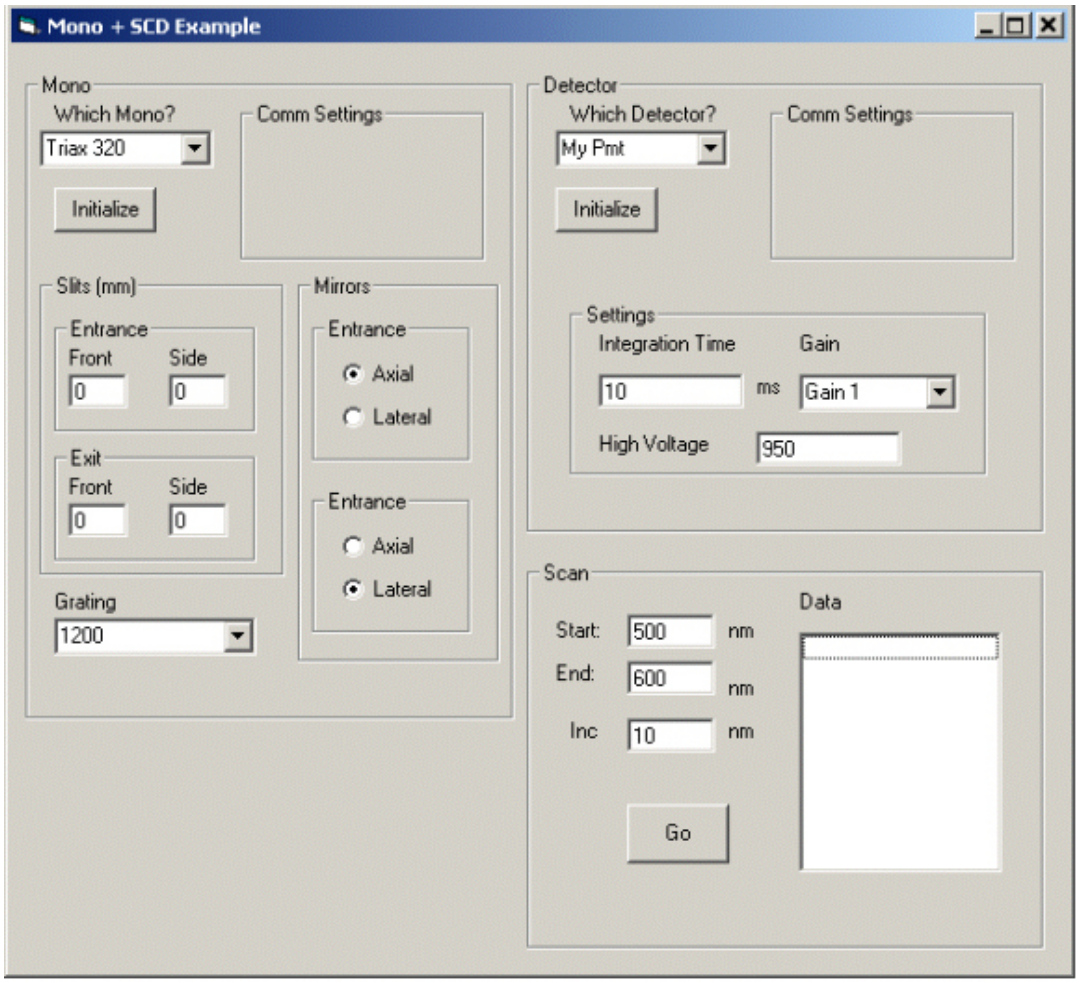
Features:

- Performs a time based acquisition
- Parameter settings are not interactive, but sent only on the Go
- Allows setting of all parameters presented
- Area Setup

  1 area → full control

  > 1 area → split chip
- Allows user to save the data

Mono + SCD Example



Features:

- Same features as individual examples (Mono,SCD)
- Changes are sent only on "Go" (not Real-time)
- Data is displayed in (x,y) format in the list control
- Data is cleared before each Scan

Mono + CCD Example

Features:

CCD Simple Acquisition features, plus ability to control:

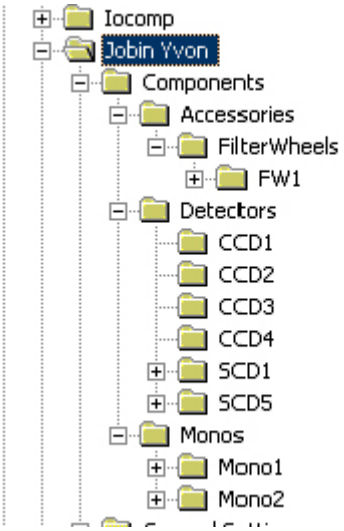
- Mono Selection/Initialization
- Mono Position
- Grating Selection

- Slit Positions
- Mirror Positions

SynerJY SDK Frequently Asked Questions (FAQ's)

How do I find the Unique Id of the instrument I want to program?

The Unique Ids are created by the Device Configuration program during installation or when the program is executed at a later time. A configured device has entries in the registry. To find out what devices are installed, you can look at the following registry keys:



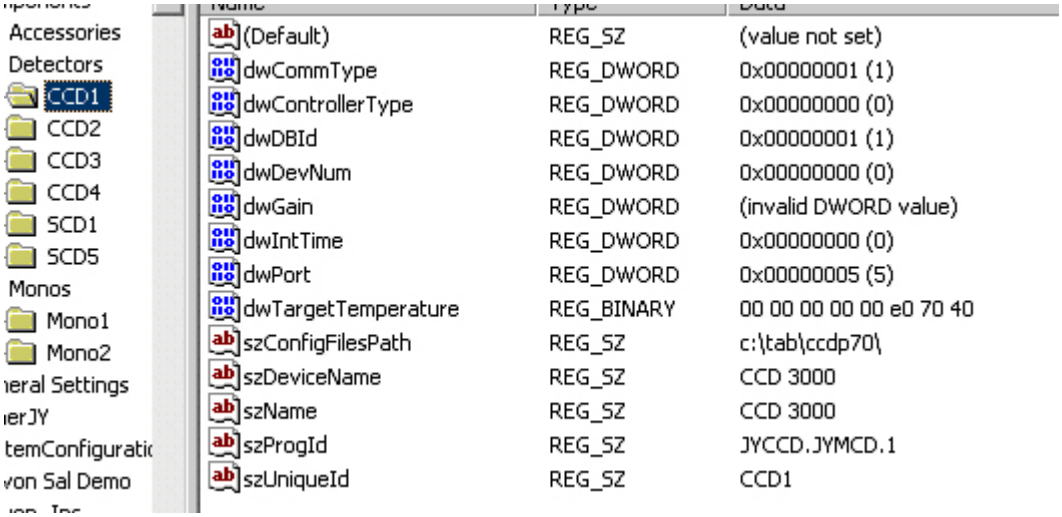
KEY: HKEY_LOCAL_MACHINE\SOFTWARE\Jobin Yvon\Components
This key is the root of all installed devices.

KEY: HKEY_LOCAL_MACHINE\SOFTWARE\Jobin Yvon\Components\Detectors
This key contains a list of all detectors. The Key is the Unique ID (i.e. – CCD1).

KEY: HKEY_LOCAL_MACHINE\SOFTWARE\Jobin Yvon\Components\Monos
This key contains a list of all monochromators. The Key is the Unique ID (i.e. – Mono1).

KEY: HKEY_LOCAL_MACHINE\SOFTWARE\Jobin Yvon\Components\Accessories
This key contains a list of types of accessories. Under each type is a list of installed accessories of that type.

Once you find the list of components of the type your looking for, you can view the values associated with the key to determine if it's the correct device. Each Key has a value szName associated with it. The szName gives the text representation of the name you entered during configuration.



Note: You should not directly modify any of the entries here (in Regedit), but should use the IJYSystemReqd::Configure interface call to get a user interface that allows for modifications of these parameters. Alternatively, you can make programmatic changes to the configuration and use the IJYDeviceReqd::Save operation to persist those changes to the registry. Manually modifying any of the parameters while in Regedit could potentially corrupt the configuration.

Uninstalling SynerJY SDK

To uninstall SynerJY SDK:

1. From the **Start** menu, select **Settings>Control Panel**.
2. Double-click on **Add/Remove Programs**.
3. Select **SynerJY SDK** from the list of programs.
4. Click **Change/Remove**.
5. At the prompt, click **Yes** to confirm that you want to remove SynerJY SDK. The uninstall program removes program files, folders, and registry entries.
6. When the files have been removed, the uninstall program indicates that the process is complete. Click **OK**.

Technical Support

If you are experiencing a technical problem, please consult the documentation supplied with the SynerJY SDK. If you are unable to solve the problem, please contact one of our customer service representatives:

In the United States:

Telephone:	1-732-494-8660 Ext. 185
Fax:	1-732-549-5125
World Wide Web:	www.jobinyvon.com/osd
Email:	osd@jobinyvon.com

Technical support hours are 8:00 A.M. - 5:00 P.M., Monday though Friday Eastern Standard/ Daylight Time.

In France:

Telephone:	+33 (0) 1 64 54 13 00
Fax:	+33 (0) 1 69 09 07 21
World Wide Web:	www.jobinyvon.fr

Worldwide:

China	+86 (0) 10 6849 2216
Germany	+49 (0) 89 462317-0
Italy	+39 0 2 57603050
Japan	+81 (0) 3 3861 8231
UK	+44 (0) 20 8204 8142

Enumeration Values

Enumeration Values (To be used for lookup of appropriate values for parameters)

```
enum jyTriggerEvent{
        jyTrigEventUndefined = -1,
        jyTrigEventOnStart,
        jyTrigEventOnEnd,
        jyTrigEventPerAccumulation,
        jyTrigEventPerCycle,
        jyTrigEventPerAcquisition,
        jyTrigEventPerRegion,
        jyTrigEventScanStart
};


enum jyTriggerSignalType{
        jyTrigSigTypeUndefined = -1,
        jyTrigSigTypeTTLHigh,
        jyTrigSigTypeTTLLow
};


enum jyDeviceOperatingMode{
        jyDevOpModeNormal = 0,
        jyDevOpModeAcqHWTimeBased,
        jyDevOpModeAcqSWTimeBased,
        jyDevOpModeAcqFKBurst,
        jyDevOpModeAcqFKBlast,
        jyDevOpModeAcqICCD,
        jyDevOpModeAcqAllowIntermediateRead,  // Used to disable intermediate
        reading of data while controller based acq is running
        jyDevOpModeLAST // USE To determine if we know this mode
};


enum jyKnownControllerSpecificOperations{
        jyControllerOpUnknown = -1,
        jyControllerOpLEDs,
};


enum jyDataAxisType{
        jyDATUndefined = 0,
        jyDATWavelength,
        jyDATPixel,
        jyDATMillimeters,
        jyDATMicrometers
};


// Enum: jyHardwareProperty
```

```
// The purpose of this enumeration is to expose hardware specific information that
// cannot be known at design time and to provide information required for
// inter-device calculations (i.e. - CCD Linearization needing mono properties
enum jyHardwareProperty{
jyUndefined = 0,

        // Mono
        jypMonoLinearDispersion, // The constant is for base grating  // nm per mm.
        jypMonoBacklashAmount, // The constant is for base grating
        jypMonoStepsPerUnit, // The constant is for base grating
        jypMonoBaseUnits,
        jypMonoBaseGrating,
        jypMonoCurrentGrating,
        jypMonoMaxLimit,
        jypMonoMinLimit,
        jypMonoTiltAngle,
        jypMonoIncludedAngle,
        jypMonoFocalLength,
        jypMonoOrder,
        jypMonoCurrentWavelength,
        jypMonoMCDWavelengthDirection, // -1 = high/low, 1 = low/high
        jypBandpassUnits, // By definition should be the current mono wavelength
        units

        // Slits
        jypSlitMaxLimit,
        jypSlitMinLimit,
        jypSlitBackLashAmount,
        jypSlitBaseUnits,
        jypSlitStepsPerUnit,

        // Light source
        jypLaserLine, // in Nanometers
        jypTotalHardwarePropertiesPlusOne
};


enum jyDeviceConfigProperty{
        jyDevConfigUndefined = 0,

        // General
        jyDevConfigDisplayName,
        jyDevConfigDBId,
        jyDevConfigDeviceNum,
        jyDevCongigControllerType,

        // Mono
        jyMonoDBFilePath,

        jyMonGratingDensity,
        jyDevGratingBlaze,
        jyDevGratingDescription,

        jyMonoSlitInstFEn,
        jyMonoSlitInstFEx,
```

```
        jyMonoSlitInstSEn,
        jyMonoSlitInstSEx,
        jyMonoSlitTypeFEn,
        jyMonoSlitTypeFEx,
        jyMonoSlitTypeSEn,
        jyMonoSlitTypeSEx,
        jyMonoSlitTypeInt1,
        jyMonoSlitTypeInt2,
        jyMonoMirrorInstEn,
        jyMonoMirrorInstEx,
        jyMonoShutterInstFEn,
        jyMonoShutterInstSEn,

        // Detectors
        jyDetectorGain,
        jyDetectorIntegrationTime,
        jyDetectorSCDType,

        // CCD
        jyDevConfigTabFilePath,
};


enum jyUnits{
        jyuUndefined ,
        jyuMillimeters,
        jyuMicrons,
        jyuNanometers,
        jyuAngstroms,
        jyuPicometers,
        jyuBandpass,
        jyuSteps,
        jyuWavenumbers,
        jyuDeltaWavenumbers,
        jyuElectronVolts,
// time
        jyuNanoseconds,
        jyuMicroseconds,
        jyuMilliseconds,
        jyuSeconds,
        jyuMinutes,
        jyuHours,
// voltage
        jyuVolts,
        jyuMillivolts,
        jyuMicrovolts,
// current
        jyuAmperes,
        jyuMilliAmps,
        jyuMicroAmps,
          // counts etc.
        jyuCounts,
        jyuCountsPerSecond,
```

```
// temperature
        jyuDegreesCelcius,
        jyuDegreesFahrenheit,
        jyuDegreesKelvin,
// CCD property
        jyuPixels,
        jyuTotalUnitsPlusOne
};


enum jyUnitsType{
        jyutUndefined,
        jyutWavelength,
        jyutSlitWidth,
        jyutTime,
        jyutTemperature,
        jyutDataUnits,
        jyutSpatial,
        jyutTotalTypesPlusOne
};


enum jyDataObjectDataFormat{
        jyDFUndefined = 0,
        jyDFInt8,
        jyDFUInt,
        jyDFInt16,
        jyDFUInt16,
        jyDFInt32,
        jyDFUInt32,
        jyDFFloat,
        jyDFDouble,
        jyDFLongDouble
};


enum jyFileFormat{
        jyFileFormatUndefined = 0,
        jyFileFormatSPC,
        jyFileFormatTabDelimitted
};


enum jyOperation{
        jyOperationUndefined = 0,
        jyAdd,
        jySubtract,
        jyMultiply,
        jyDivide,
        jyCosmicRemovalSingle,
        jyCosmicRemovalMultiple
};
```

```
enum jyAnalysisOperation{
        jyAnalysisUndefined = 0,
        jyAnalysisStdDev,
        jyAnalysisAvg,
        jyAnalysisMin,
        jyAnalysisMax,
        jyAnalysisSum,
        jyAnalysisSumSqrs
};


enum jyCCDDataType{
        JYMCD_ACQ_FORMAT_IMAGE = 0,
        JYMCD_ACQ_FORMAT_SCAN
};


enum jyCommType{
        no_comm=0,
        jyGPIB,
        jySerial,
        jyIP,
        jyUSB
};


enum jyDeviceClass{
        jyDevClassUndefined = 0,
        jyDevClassDetector,
        jyDevClassMono,
        jyDevClassAccessory,
        jyDevClassLightSource,
        jyDevClassOther // KEEP THIS AS THE LAST
};


enum jyDeviceType{
        jyDevTypeUndefined = 0,
        // Detector Types
        jyDevTypeMCD,
        jyDevTypeSCD,
        // Mono Types
        jyDevTypeMono,
        // Accessory Types
        jyDevTypeFW,
        jyDevTypePolarizer,
        jyDevTypeTempControl,
        jyDevTypeXY,
        jyDevTypeXYZ,
        jyDevTypeSC,
        jyDevTypeChopper
};
```

```
enum jySubDeviceType{
        jySubDevTypeUndefined = 0,
        // Mono Types
        jySubDevTypeSlit,
        jySubDevTypeMirror,
        jySubDevTypeTurret,
        jySubDevTypeShutter,
        // Detector Types'
        jySubDevTypeReference
};


// Supported file types for JY
enum jySupportedFileType{
        jyUndefinedFileType = 0,
        jyTabDelimitted,
        jySPC,
        jySPCAsBlob
};


// MultiAcq specifiers for when things should happen
enum jyMultiAcqState{
        jyMultiAcqStateNever, // invalid value
        jyMultiAcqStateBeforeFirstOnly, // before the first of multiple acqs only
        jyMultiAcqStateBetweenOnly, // from acq 2 to n only
        jyMultiAcqStateBeforeEach, // before each (including first acq)
};


enum jyDataItemProperty{
        jyDIPUndefined = 0,
        jyDIPDescription,
        jyDIPSourceName,

        jyDIPXAxisLabel,
        jyDIPXAxisUnitsVal,
        jyDIPXAxisUnitsString,

        jyDIPYAxisLabel,
        jyDIPYAxisUnitsVal, // enumeration val
        jyDIPYAxisUnitsString, // string

        jyDIPZAxisLabel,
        jyDIPZAxisUnitsVal,
        jyDIPZAxisUnitsString,

        jyDIPTAxisLabel,
        jyDIPTAxisUnitsVal, // enumeration val
        jyDIPTAxisUnitsString, // string

        jyDIPDataLabel,
        jyDIPDataUnitsVal,
        jyDIPDataUnitsString,
```

```
        jyDIPDataOverrange, // Bool val telling if data is overrange or not
        jyDIPHidden,
        jyDIPCosmicEnabled, // Tells data object whether or not to do cosmic ray
        removal (Only works for avg data !!!)
        jyDIPCosmicMethod, // Method to use for cosmic removal
        jyDIPCosmicParameter1, // parameter1 required by the method
        jyDIPCosmicParameter2, // parameter2 required by the method
        jyDIPFlippedX, // parameter indicating if the data has been flipped (not
        settable)
};


enum jyResultsProperty{
        jyRPUndefined = 0,
        jyRPDescription,
        jyRPHeader,
        jyRPTime,
        jyRPDate,
        jyRPSaveMode,
};


enum jySaveMode{
        jySMOverwriteAlways = 0,
        jySMSaveAsNew,
        jySMPrompt,
};


enum jyCommParamType{
        jyCPTUndefined = 0,
        jyCPTCommType,
        jyCPTPortNum,
        jyCPTDeviceName,
        jyCPTBaudRate,
        jyCPTDatabits,
        jyCPTParitybits,
        jyCPTStopbits

};
```

Components

Monochromator

Supported Interfaces

- IJYDeviceReqd
- IJYSystemReqd
- IJYMonoReqd

Examples

Example programs for Jobin Yvon monochromators, detectors, and accessories are contained in the SynerJY SDK and can be run in the following languages:

- Microsoft Visual C++ 6.0
- Microsoft Visual Basic 6.0
- Microsoft VBA

CCD

Supported Interfaces

- IJYDeviceReqd
- IJYSystemReqd
- IJYDetectorReqd
- IJYCCDReqd

Examples

Example programs for HORIBA Jobin Yvon monochromators, detectors, and accessories are contained in the SynerJY SDK and can be run in the following languages:

- Microsoft Visual C++ 6.0
- Microsoft Visual Basic 6.0
- Microsoft VBA

Single Channel Detector

Supported Interfaces

- IJYDeviceReqd
- IJYSystemReqd
- IJYDetectorReqd
- IJYSCDReqd

Examples

Example programs for HORIBA Jobin Yvon monochromators, detectors, and accessories are contained in the SynerJY SDK and can be run in the following languages:

- Microsoft Visual C++ 6.0
- Microsoft Visual Basic 6.0
- Microsoft VBA

Filter Wheel

Supported Interface: IJYAccessoryReqd

Derived from IJYSystemReqd

Overview

All Accessories are required to implement this interface. It contains basic functions to allow a caller to determine the type of accessory and whether or not the device is ready to accept commands.

interface IJYAccessoryReqd : IJYSystemReqd

HRESULT Configure() Launches a device-specific configuration dialog allowing customization of operating parameters.

AccessoryClass

Function: HRESULT AccessoryClass([out, retval] enum jyAccessoryClass *pVal);
Description: For type identification, each accessory needs to identify its class and type.
Parameters:

pVal – class of accessory {jyAccClassIndexed, jyAccClassStepper}

AccessoryType

Function: HRESULT AccessoryType([out, retval] enum jyAccessoryType *pVal);
Description: For type identification, each accessory needs to identify it's class and type.
Parameters:

pVal = type of accessory
{jyAccTypeFilterWheel, jyAccTypePolarizer, jyAccTypeXYStage, jyAccTypeSampleChanger, jyAccTypeTemperature, jyAccTypeChopper}

AccessoryBusy

Function: HRESULT AccessoryBusy([out, retval] VARIANT_BOOL *isBusy);
Description: Busy indicates that the accessory is actively doing something.

Note: The difference between "Busy" and "Ready": If a device is not busy, it MAY be ready. "Ready" means ready to perform another operation or receive another command. The difference is based on whether or not the controller has other devices under its control that are currently busy. For example, HORIBA Jobin Yvon produces a filter wheel and a monochromator that controls that filter wheel through common electronics. If the filter wheel is not busy, but the monochromator is busy, then the filter wheel is not ready. The difference is subtle, but important. You should use "Busy" to determine when an operation (i.e. –move) is complete. You should use "Ready" before you send the next command.

Parameters:

isBusy – whether or not the device in question is busy

Example:

```
myAccessory.SetIndexPosition( position );
while ( myAccessory.AccessoryBusy )
 `  continue to wait
wend
```

Accessory Ready

Function: HRESULT AccessoryReady([out, retval] VARIANT_BOOL *isReady);
Description: Ready indicates that the device is ready to accept another command.

Note: The difference between "Busy" and "Ready": If a device is not busy, it MAY be ready. "Ready" means ready to perform another operation or receive another command. The difference is based on whether or not the controller has other devices under its control that are currently busy. For example, HORIBA Jobin Yvon produces a filter wheel and a monochromator that controls that filter wheel through common electronics. If the filter wheel is not busy, but the monochromator is busy, then the filter wheel is not ready. The difference is subtle, but important. You should use "Busy" to determine when an operation (i.e. –move) is complete. You should use "Ready" before you send the next command.

Parameters:

isReady – whether or not the device is ready to receive another command

Example:

```
If ( myAccessory.AccessoryReady()) then
   ` Ready, so go ahead and move
   myAccessory.SetIndexPosition( position )
Endif
```

IJYAccIndexReqd Interface

Derived from IJYAccessoryReqd

Overview

Indexed devices are those which contain a set number of discrete positions (i.e. Filter Wheel). The IJYAccIndexReqd Interface provides the basic functionality to retrieve the device's minimum and maximum positions, set the device to a position, and read the device's current position.

SetIndexPosition

Function: HRESULT SetIndexPosition([in]long newVal,[in,optional]VARIANT devNum);
Description: Sends the indexed device to the specified position.
Parameters:
newVal – position to go to
devNum – optional variable to indicate a subaddress on an accessory.
For Example, xy stage needs to specify whether the position is x or y
Example:
myFW.SetIndexPosition(1)

GetIndexPosition

Function: HRESULT GetIndexPosition([out]long *pVal, [in,optional]VARIANT devNum);
Description: Retrieves the current position of the accessory.
Parameters:
pVal – position the device is at
devNum – optional variable to indicate a subaddress on an accessory.
For Example, xy stage needs to specify whether the position is x or y
Example:
myFW.GetIndexPosition(whichFilter)

GetIndexMinPosition

Function: HRESULT GetIndexMinPosition([out]long *pVal, [in, optional]VARIANT devNum);
Description: Retrieve the minimum valid position for the accessory.
Parameters:
pVal – minimum valid position
devNum – optional variable to indicate a subaddress on an accessory.
For Example, xy stage needs to specify whether the position is x or y

GetIndexMaxPosition

Function: HRESULT GetIndexMaxPosition([out]long *pVal, [in, optional]VARIANT devNum);\
Description: Retrieves the maximum valid position for this accessory.

Parameters:
pVal – minimum valid position
devNum – optional variable to indicate a subaddress on an accessory.
For Example, xy stage needs to specify whether the position is x or y

IJYFilterWheelReqd Interface

(Derived from IJYAccIndexReqd)

Overview

This interface specifically supports functions for the filter wheel.

interface IJYFilterWheelReqd : IJYAccIndexReqd

GetFilterInfo

Function: HRESULT GetFilterInfo([in]int index, [out]BSTR *description, [out]double *min_wl, [out]double *max_wl);
Description: Retrieves the information about the filter in a given position . INFORMATIONAL ONLY. Nothing is done based on this information.
Parameters:
index – which filter
Description – text description of the filter
Min_wl – the minimum wavelength for this filter
Max_wl – the maximum wavelength for this filter

SetFilterInfo

Function: HRESULT SetFilterInfo([in]int index, [in]BSTR description, [in]double min_wl, [in]double max_wl);
Description: Sets the information about the filter in a given position . INFORMATIONAL ONLY. Nothing is done based on this information.
Parameters:
index – which filter
Description – text description of the filter
Min_wl – the minimum wavelength for this filter
Max_wl – the maximum wavelength for this filter

Component Interfaces

IJYCCDReqd

Overview

The IJYCCDReqd interface deals with the specific functionality required exclusively by Multi Channel Detectors (CCDs).

Code Example

Setup the CCD for multiple area, spectral acquisition, splitting the chip into 2 areas

```
'Assumes CCD has been connected, initialized and in a good state
' Get the chip size
myCCD.GetChipSize( &x, &y )
' Setup the Acquisition Parameters
myCCD.DefineAcqFormat(  2, jyImage  )
' Setup 2 areas, each ½ chip in the y direction
myCCD.DefineArea( 1, 1, 1, x , y / 2, 1, 1 )
myCCD.DefineArea( 2, 1, (y/2)+1, x, y, 1, 1 )
myCCD.SetDefaultUnits( jyutTime, jyuMilliseconds )
myCCD.IntegrationTime = 10
' Get the first ADC and select it
myCCD.GetFirstADC( adcText, adcToken )
myCCD.SelectADC( adcToken )
myCCD.StartAcquisition( bShutterFlag )
while ( myCCD.AcquistionBusy )
  {
  ' do nothing
  }
' Here we are ready to read out the data
myCCD.GetResult( &myResult )
```

DefineAcquisitionFormat

Function:HRESULT DefineAcquisitionFormat([in]enum jyCCDDataType, [in]long numAreas);
Description: Sets up the acquisition mode and the number of areas for subsequent acquisitions. When setting up the Acquisition format, you should resend the area list.
Parameters:
jyCCDDataType – the format of the data being acquired (image or data)
Example:

```
' Prepare for 4 areas spectral acquisition
myCCD.DefineAcquisitionFormat(JYMCD_ACQ_FORMAT_SCAN, 4 )
```

See Also: DefineArea

DefineArea

Function:HRESULT DefineArea([in]int areaNum, [in]int xOrigin, [in]int yOrigin, [in]int xSize, [in]int ySize, [in]int xBin, [in]int yBin);
Description: This function sets the parameters for a specified area number. Parameters must fall within the parameters of the chip and must be called for each area (1-n) specified in call to DefineAcquisitionFormat.
Parameters:

areaNum – area number being setup

XOrigin	Start x coordinate of the area
YOrigin	Start y coordinate of the area
XSize	Size x of the area
YSize	Size y of the area
XBin	bin factor in x direction
YBin	bin factor in y direction

Note: ALL AREA INFORMATION IS 1 BASED

Example:

```
` Set up n areas from an array of areas (Note: n must match numAreas
parameter in
` DefineAcquisitionFormat command
for( areaNum = 1 to n )
   myCCD.DefineArea( areaNum, xOrigin[areaNum], yOrigin[areaNum],
xSize[areaNum],
ySize[areaNum], xBin[areaNum], yBin[areaNum] )
Next areaNum
```

See Also: DefineAcquisitionFormat

SelectADC

Function:HRESULT SelectADC([in]enum jyADCType adc);
Description: Selects the active ADC for the CCD and performs any necessary re-initialization. The adc value must be valid for the detector. This can be determined by using GetFirstADC, GetNextADC functions.
Parameters:

jyADCType – the token enumerated by the component for the desired adc

Example:

```
` Select from list box that had the itemdata set
myCCD.SelectADC( myADCListBox.ItemData(myADCListBox.ListIndex ) )
```

See Also: GetFirstADC, GetNextADC, CurrentADC

GetFirstADC/GetNextADC

Function:HRESULT GetFirstADC([out]BSTR *Description, [out, retval]long *adcToken);
HRESULT GetNextADC([out]BSTR *Description, [out, retval]long *adcToken);
Description: Enumerates a list of token-string pairs that should be used for displaying options to the user (the string) and sending values back to the controller (the token).

Parameters:

Description:
adcToken:

Example:

```
` fill a list box with the available adc's and put the tokens into the associated
itemdata
    Dim adcIdx As Integer
    Dim adcToken As Long
    Dim adcName As String
    Dim currentADC As Long

    ADCSelect.Clear
    'ADCSelect.ListIndex = 0
    currentADC = ccdObject.currentADC
    adcToken = ccd.GetFirstADC(sName)
    If (adcToken = -1) Then
        ADCSelect.AddItem "NONE"
        ADCSelect.Enabled = False
    Else
        While (Not adcToken = -1)
            ADCSelect.AddItem (adcName)
            ADCSelect.ItemData(ADCSelect.NewIndex) = adcToken
            If (adcToken = currentADC) Then
                ADCSelect.ListIndex = ADCSelect.NewIndex
            End If
            token = ccd.GetNextADC(adcName)
        Wend
        If (ADCSelect.ListIndex = -1) Then
            ADCSelect.ListIndex = 0
        End If
    End If
    If (ADCSelect.ListCount = 1) Then
        ADCSelect.Enabled = False
    End If
```

See Also: SelectADC

CurrentADC

Function:HRESULT CurrentADC([out,retval]long *pVal);
HRESULT CurrentADC([in]long val);
Description: Sets/Gets the Current ADC (Should use SelectADC function for selecting the ADC).
Parameters:

Value – the token for the adc to be selected

Example: See GetFirstADC
See Also: SelectADC

OpenShutter

Function:HRESULT OpenShutter();
CloseShutter();
Description: Open and Close the CCD-controlled shutter.

FlushCount

Function:HRESULT FlushCount([out, retval] long *pVal);
HRESULT FlushCount([in] long newVal);
Description: Sets/Gets the number of flushes to be performed before each acquisition in a single acquisition mode. If you are doing multiple accumulations, hardware based, then you should use the MultiAcqFlushCount function.
Parameters:

pVal – ptr to long value to receive the current flush count
newVal – the count of flushes to perform before each acquisition

Example:

```
` Setup the detector to flush/clean 2 times before each individual acquisition
myCCD.FlushCount = 2
```

Gain

Function:HRESULT Gain([out, retval] long *pVal);
HRESULT Gain([in] long newVal);
Description: Sets/Gets the current gain setting for the CCD.
Parameters:

pVal – ptr to long value to contain the token value for the current gain setting
newVal – token value to be used for selection of the new gain setting

Example:

```
` Select from list box that had the itemdata set
myCCD.Gain = myGainListBox.ItemData(myGainListBox.ListIndex )
```

See Also: GetFirstGain, GetNextGain

CurrentTemperature

Function:HRESULT CurrentTemperature([out, retval] double *pVal);
Description: Reads the current operating temperature of the CCD in current temperature units.
Parameters:

pVal – ptr to a double to contain the current temperature of the detector in the current default temperature units of the detector

Example:

```
If ( myCCD.CurrentTemperature > m_requiredTemp )
  MsgBox( "Detector Not at temperature )
Else
  MsgBox( "Detector at temperature")
Endif
```

See Also: GetDefaultUnits, SetDefaultUnits, TemperatureSetPoint

TemperatureSetPoint

Function:HRESULT TemperatureSetPoint([out, retval] double *pVal);
HRESULT TemperatureSetPoint([in] double newVal);
Description: Set/Get the current target temperature of the CCD. Temperature set point is limited by the capabilities of the associated hardware. Consult your hardware manual for achievable values.

Parameters:
pVal – double reference to contain the value of the temperature set point (in default units) on successful return
newVal – double value (in default units) to be used as the new temperature set point
Example:
myCCD.TemperatureSetPoint = m_goalTemp
See Also: GetDefaultUnits, SetDefaultUnits, CurrentTemperature

GetChipSize

Function:HRESULT GetChipSize([out]int *activeXPixels, [out]int *activeYPixels);
Description: Returns the chip size parameters in pixels.
Parameters:
activeXPixels
activeYPixels
Example:
myCCD.GetChipSize(x, y)
xDisplay.Text = x
yDisplay.Text = y
See Also: GetPixelSpacing

GetPixelSpacing

Function:HRESULT GetPixelSpacing([out]int *horizontalPixelSpacing, [out]int *verticalPixelSpacing);
Description: Returns the inter-pixel spacing in microns.
Parameters:
horizontalPixelSpacing
verticalPixelSpacing
Example: GetChipSize

GetResult

Function:HRESULT GetResult([out, retval]IJYResultsObject **resultObj);
Description: Retrieves the result object for the last completed acquisition
Parameters:
resultObj
Example:
myCCD.StartAcquisition(shutterFlag)
while (myCCD.AcquisiionBusy())
` Wait
wend
myResult = myCCD.GetResult()
MyProcessResultFunction(myResult)
See Also: StartAcquisition, IsBusy, DoAcquisition, Results and DataObjects

AcquisitionCount

Function:HRESULT AcquisitionCount([out,retval]long *pVal);
HRESULT AcquisitionCount([in]long val);
Description: Set/Get the number of accumulations to be performed on a single StartAcquisition or DoAcquisition.
Parameters:

pVal - gets the value of the acq count
val - sets the value of the acq count

See Also: AdvancedTopics:Multiple Accumulations

SetMono

Function: HRESULT SetMono([in]IJYSystemReqd * iMono);
HRESULT GetMono([out]IJYSystemReqd ** iMono);
Description: Set/Get the interface pointer to the mono object that is associated with the CCD. This is used primarily for extracting monochromator parameters. Alternatively, you can call SetMonoProperty function to set specific properties. Advanced Topic: See Linearizing Your Data.
Parameters:

iMono

Example:

myCCD.SetMono(myMonoObject) ' all linearization of data will be based on this associated mono

See Also: GetAxisAs, SetMonoProperty

SetMonoProperty

Function:HRESULT SetMonoProperty([in]enum jyHardwareProperty whichProperty, [in]VARIANT propVal);
Description: Sets individual properties for the mono. This should only be used in the absence of the availability of a IJYMono interface pointer, or to set a dynamic property for inquiry purposes.
Parameters:

whichProperty – the enumeration value for the property you want to set
propVal - the value to set the propery to

Example:

```
' In absence of a JYMono object, we can set specific properties to acquire
accurate linearization
    ccd.SetMonoProperty jypMonoCurrentWavelength, centerWavelength.Text
    ccd.SetMonoProperty jypMonoCurrentGrating, 1200
    ccd.SetMonoProperty jypMonoCurrentWavelength, 546
    ccd.SetMonoProperty jypMonoIncludedAngle, 24
    ccd.SetMonoProperty jypMonoFocalLength, 319.652
    ccd.SetMonoProperty jypMonoTiltAngle, 3.16
    ccd.SetMonoProperty jypMonoOrder, -1
```

GetAxisAs

Function:HRESULT GetAxisAs([in]IJYDataObject *dataObj, [in]long whichAxis, [in]enum jyDataAxisType axisType, [out,retval]IJYAxis **axis);
Description: This function retrieves an Axis object that contains information about the labels and types of data associated with the current data object. The assumption is that the data object has come from an acquisition on this ccd.
Parameters:

dataObj – The data object from which the axis will be extracted
whichAxis – with an 'n' dimensional object, need to specify which axis you want 1 = x, 2 = y
axisType – values → jyDATWavelength, jyDATPixel, jyDATMillimeters, jyDATMicrometers
axis - contains the actual axis object that can be used to extract labels, data, type, etc…

See Also: Results and Subcomponents, Axis Objects

SetAreaSubtimer/GetAreaSubtimer

Function:HRESULT SetAreaSubtimer([in]long areaNum, [in]double time);
HRESULT GetAreaSubtimer([in]long areaNum, [out]double *time);
Description: For Internal Use Only.

ChipDescription

Function:HRESULT ChipDescription([out,retval]BSTR *pVal);
HRESULT ChipDescription([in]BSTR val);
Description: Detailed description of the chip currently being used. For Future Use.
See Also: GetChipSize, GetPixelSpacing

DarkSubtract

Function:HRESULT DarkSubtract([out,retval]VARIANT_BOOL *doSubtraction);
HRESULT DarkSubtract([in]VARIANT_BOOL doSubtraction);
Description: Enables or disables automatic dark subtraction for subsequent acquisitions. The detector will do one acquisition without opening the shutter and then do one with the shutter open and subtract the first acquisition from the second. The result will have the dark subtracted.

Note: Only Works for DoAcquisition. If using StartAcquisition, you must perform the dark subtraction yourself.

Parameters:

doSubtraction – specifies whether to enable or disable subtraction

Example:

```
myCCD.DarkSubtract = True
myCCD.DoAcquisition( shutterFlag )
'Data in the UpdateEvent will have dark subtracted data
```

See Also: DoAcquisition, StartAcquisition

GetWavelengtCoverage

Function:HRESULT GetWavelengthCoverage([in]double position, [in]enum jyUnits units, [out]double *start, [out]double *end);
Description: Retrieves the coverage of the first area based on the provided position. Assumes either SetMono has been called to establish a JYMono object or SetMonoProperty has been used to provide the required parameters for the linearization.
Parameters:

Position – position in 'units' of the associated mono
Units – the units of the value in 'Position'
Start – On success, contains the start of coverage in 'units'
End – On success, contains the end of coverage in 'units'

Example:

```
` What is the wavelength coverage based on the current position
currentWL = myMono.GetCurrentWavelength
units = myMono.GetDefaultUnits( jyutWavelength )
myCCD.SetMono( myMono ) 'only if it hasn't been previously sent
myCCD.GetWavelengthCoverate( currentWL, units, startCoverage,
endCoverage )
```

See Also: SetMonoProperty, SetMono

ReadReferenceValue

Function:HRESULT ReadReferenceValue([out,retval] double* pRefVal);
Description: Returns the reference value (reference sum) for an acquisition. For a single acquisition one value will be returned. For a multi-acq acquisition the number of values returned will be equal to the number of acquisitions executed.
Parameters:

pRefVal – ptr to long value to contain the value of the ref sum

Example:

```
refVal = myCCD..ReadReferenceValue
```

SetReferenceGain

Function:HRESULT SetReferenceGain([in] long gainToken);
Description: Sets the gain to be used for the reference value.
Parameters:

gainToken – token value to be used for selection of the new reference gain setting

Example:

```
'Select from combo box that had the itemdata set
myCCD.SetReferenceGain
(myRefGainCombo.ItemData(myRefGainCombo.ListIndex)
```

See Also: GetFirstReferenceGain, GetNextReferenceGain

GetReferenceGain

Function:HRESULT GetReferenceGain([out, retval] long* pGainToken);
Description: Gets the token value representing the reference gain setting.

Parameters:

pGainToken – ptr to a long value containing the value of the token for the current reference
gain setting

Example:

```
gainToken = myCCD.GetReferenceGain
```

See Also: GetFirstReferenceGain, GetNextReferenceGain

GetFirstReferenceGain/GetNextReferenceGain

Function:HRESULT GetFirstReferenceGain([out]BSTR* gainDescription, [out, retval] long* pGainToken);
Function:HRESULT GetNextReferenceGain([out]BSTR* gainDescription, [out, retval] long* pGainToken);

Description: Enumerates a list of token-string pairs that should be used for displaying options to the user (the string) and sending values back to the controller (the token).

Parameters:

gainDescription – a BSTR value containing the description of the reference gain
pGainToken - ptr to a long containing the value of the token for the reference gain

Example:

```
'fill a combo box with the available reference gains and put the tokens into the associated itemdata
Dim token As Long
Dim sRefGainName As String
Dim nStrLen As Integer
Dim strGain As String
token = myCCD..GetFirstReferenceGain(sRefGainName)

 If cbRefGains.ListCount = 0 Then
 While (Not token = -1)
       'A returned string is in the format "AUXIN Gainxx"
      nStrLen = Len(sRefGainName)
     ' Strip the AUXIN from the returned string before displaying in combo box
      strGain = Right(sRefGainName, nStrLen - 6)
      cbRefGains.AddItem (strGain)
      cbRefGains.ItemData(cbRefGains.NewIndex) = token
      If (m_lastRefGain = token) Then
          cbRefGains.ListIndex = cbRefGains.NewIndex
      End If
      token = myCCD.GetNextReferenceGain(sRefGainName)
Wend
```

See Also: GetReferenceGain, SetReferenceGain

ReferenceMode

Function:HRESULT ReferenceMode([out, retval] long* pRefMode);
Function:HRESULT ReferenceMode([in] long refMode);

Description: Get/Set the reference mode of the controller.

Parameters:

pRefMode – ptr to a long containing the value of the token for the reference mode
refMode – long containing the value of the token for the reference mode

Example:

```
myCCD..ReferenceMode = refMode ‘set the reference mode
refMode = myCCD.ReferenceMode  ‘get the reference mode
```

AutoNormalize

Function:HRESULT AutoNormalize([out, retval] VARIANT_BOOL* pNormalize);
Function:HRESULT AutoNormalize([in] VARIANT_BOOL normalize);
Description: Get/Set the AutoNormalize mode of the controller. If the AutoNormalize mode is set the controller will scale the results by dividing the data object by the reference value.
Parameters:
pNormalize – ptr to VARIANT_BOOL that will receive the value of the AutoNormalize property
normalize – a VARIANT_BOOL containing the value for setting the AutoNormalize property
Example:

```
myCCD.AutoNormalize = True ‘set AutoNormalize property to True

Dim bAutoNormalize as Boolean
bAutoNormalize = myCCD.AutoNormalize ‘get the value of the AutoNormalize property
```

IJYDetectorReqd

Derived from IJYSystemReqd

Overview

This interface contains functionality related to all detectors. Derived classes include IJYCCDReqd (multi channel detector) and IJYSCDRequired (single channel detectors).

AcquisitionBusy

Function: HRESULT AcquisitionBusy([out,retval]VARIANT_BOOL *busy);
Description: Checks to see if the device is busy doing an acquisition. This is only required if you are doing serial data acquisition.
Parameters:

busy - either VARIANT_TRUE or VARIANT_FALSE.
True → detector is currently busy acquiring
False → detector is currently idle

Example:

```
` Polling Acquisition
detector.StartAcquisition
while ( detector.AcquisitionBusy )
`do nothing
wend
`Get the data
result = detector.GetResult()
```

See Also: StartAcquisition, DoAcquisition, GetResult

StartAcquisition

Function: HRESULT StartAcquisition([in]VARIANT_BOOL shutterOpen);
Description: Kicks off an acquisition with the current settings. This is only required if you are doing serial data acquisition.
Parameters:

shutterOpen – flag determines whether or not to open the shutter during the acquisition.

Example:

```
` Polling Acquisition
detector.StartAcquisition
while ( detector.AcquisitionBusy )
`do nothing
wend
`Get the data
result = detector.GetResult()
```

See Also: AcquisitionBusy, DoAcquisition, Update, GetData

GetData

Function: HRESULT GetData([in,out]VARIANT *dataPtr);
Description: Retrieves data from detector.

Note: Only functions for Single Channel Detector. Multi Channel Detector must use GetResult function. GetResult works for both detector types and is the recommended way to read the data.

Parameters:
dataPtr – Variant containing the data from the just completed acquisition
Example: See GetResult
See Also: StartAcquisition, AcquisitionBusy, GetResult

DoAcquisition

Function:HRESULT DoAcquisition([in]VARIANT_BOOL shutterOpen);
Description: Performs a complete threaded acquisition. When the acquisition is complete, the caller will receive a Windows Event containing the data. This function returns immediately.
Parameters:
shutterOpen – whether or not to open the shutter during acquisition
Example:

```
` Start the acquisition and then wait for the event
Public Sub StartMyAcquisition
myDetector.StartAcquisition( shutterFlag )
End Sub

` Handler implemented to "catch" the event
Private Sub myDetector_Update(ByVal updateType As Long, ByVal eventInfo
As JYSYSTEMLIBLib.IJYEventInfo)
'Data is encapsulated in the IJYEventInfo object
myResult = eventInfo.GetResult()
MyProcessResultFunction( myResult )
End Sub
```

See Also: StartAcquisition, AcquistionBusy, GetResult

DataSize

Function: HRESULT DataSize([out, retval] long *pVal);
Description: Retrieves the number of data points in a single acquisition as it is currently configured.
Parameters:
pVal – the long value to receive the number of data points to be returned
Example:

```
` Setup all the parameters and determine how many data points are involved
myDetector.IntegrationTime = 10
SetupMultipleAreas()
howManyDataPoints = myDetector.DataSize
```

IntegrationTime

Function: HRESULT IntegrationTime([out, retval] double *pVal);
HRESULT IntegrationTime([in] double newVal);
Description: Sets/Gets the per acquisition integration time for the detector.
Parameters: pVal – returned value for the integration time (in default time units)
newVal - sets the integration time (in default time units)
Example:

```
'SET DEFAULT UNITS ONLY ONCE AND SAVED WITH THE DETECTOR
CONFIGURATION
myDetector.SetDefaultUnits( jyutTime, jyuMilliseconds )
myDetector.IntegrationTime = 10
```

See Also: SetDefaultUnits

GetFirstGain/GetNextGain

Function: HRESULT GetFirstGain([out]BSTR *description, [out,retval]long *gainVal);
HRESULT GetNextGain([out]BSTR *description, [out,retval]long *gainVal);
Description: Enumerates the list of gains available for the selected device. The user can then use the token value (gainVal) to select a specific gain. The string can be used to present a human readable format of the gain list to the end user.
Parameters:
description – text description for the gain
gainVal – token to be used for selecting the gain
Example:

```
` Fill a list box with the available gains for this detector
   Dim gainToken As Long
   Dim gainName As String
   Dim lastGain As Long
   lastGain = ccd.Gain
   GainList.Clear
   gainToken = ccd.GetFirstGain(gainName)
   If (gainToken = -1) Then
      GainList.AddItem "NONE"
      GainList.Enabled = False
   Else
      While (Not gainToken = -1)
         GainList.AddItem (gainName)
         GainList.ItemData(GainList.NewIndex) = gainToken
         If (lastGain = gainToken) Then
            GainList.ListIndex = GainList.NewIndex
         End If
         gainToken = ccd.GetNextGain(gainName)
      Wend
   End If
   If (GainList.ListCount = 1) Then
      GainList.Enabled = False
   End If
```

See Also: Enumerations, Token String Pairs

AccumulationMode

Function:HRESULT AccumulationMode([out, retval] enum jyDetectorAccumulationMode *pVal);
HRESULT AccumulationMode([in] enum jyDetectorAccumulationMode newVal);
Description: Set the accumulation mode for the detector.
See Also: Advanced Topics: Multiple Accumulations

NumberofAccumulations

Function: HRESULT NumberOfAccumulations([out, retval] long *count);
HRESULT NumberOfAccumulations([in] long count);
Description: Sets the number of accumulation per acquisition start.

Note: This is used only if the controller is handling the multiple accumulations. Otherwise, if you are programmatically looping on number of accumulations, then this number should be set to 1.

Parameters:

count – set/gets the number of accumulations

Example:

```
If(myDetector.IsOperatingModeSupported((jyDevOpModeAcqHWTimeBased) )
    'Have controller do multiple accumulations
    myDetector.SetOperatingModeValue((jyDevOpModeAcqHWTimeBased,
True )
    myDetector.NumberOfAccumulations = 10
else
   myDetector.NumberOfAccumulations = 1
endIf
```

See Also: Advanced Topics: Multiple Accumulations, IsOperatingModeSupported, SetOperatingModeValue

TimeInterval

Function: HRESULT TimeInterval([out, retval] double *interval);
HRESULT TimeInterval([in] double interval);
Description: Time period between starts of multiple accumulations. Applies only for controller based acquisitions.
Parameters:

interval – time (in default time units) to wait between acquisition starts

Example:

```
'Take 10 acquisitions with 10 ms integration time, evenly spaced over ½
second
myDetector.NumberOfAccumulations = 10
myDetector.IntegrationTime = 10
myDetector.TimeInterval = 50
```

See Also: Advanced Topics: Multiple Accumulations

ReadyforAcquisition

Function: HRESULT ReadyForAcquisition([out,retval]VARIANT_BOOL *ready)**Description:** Called to verify that the current settings produce a valid acquisition setup.

Note: Call this function whenever acquisition parameters that could effect acquisition validity are changed (i.e. – area list in CCD)

Parameters:
ready – VARIANT_TRUE → Ready to acquire
VARIANT_FALSE → Not ready. Check acquisition parameters.
Example:
See Also: Advanced Topics: Multiple Accumulations

MultiAcqShutterMode

Function: HRESULT MultiAcqShutterMode([out,retval]enum jyMultiAcqState *onFirstonEach);
HRESULT MultiAcqShutterMode([in]enum jyMultiAcqState onFirstonEach);
Description: Sets the shutter operating mode for multiple accumulation acquisitions.

Note: These parameters only apply if number of accumulations > 1 and hardware-based accumulations are being used.

Parameters:
acqState – when to do the shutter
Example:
` Open the shutter before the first acquisition only
myDetector.MultiAcqShutterMode(jyMultiAcqStateBeforeFirstOnly)
See Also: Advanced Topics: Multiple Accumulations

SetMultiAcqDelay/GetMultiAcqDelay

Function: HRESULT SetMultiAcqDelay([in]enum jyMultiAcqState acqState, [in]double delayForMode);
HRESULT GetMultiAcqDelay([in]enum jyMultiAcqState acqState, [out,retval]double *delayForMode);
Description: Set/Get the delay setting between multiple accumulations for the specified acq state.

Note: These parameters only apply if number of accumulations > 1 and MultiAcqHardwareMode is set to VARIANT_TRUE.

Parameters:
acqState – when to delay
delayForMode – delay in working time units
Example:
` Setup delays so that there is a 10 ms delay before 1st acq and 5 ms delay between other acqs
myDetector.SetMultiAcqDelay(jyMultiAcqStateBeforeFirstOnly, 10)
myDetector.SetMultiAcqDelay(jyMultiAcqStateBetweenOnly, 5)
See Also: Advanced Topics: Multiple Accumulations

SetMultiAcqCleanCount/Get MultiAcqCleanCount

Function: HRESULT SetMultiAcqCleanCount([in]enum jyMultiAcqState acqState, [in]long cleansForMode);
HRESULT GetMultiAcqCleanCount([in]enum jyMultiAcqState acqState, [out,retval]long *cleansForMode);
Description: Set/Get the clean count for the given acq state.

Note: These parameters only apply if number of accumulations > 1 and hardware-based accumulations are being used.

Parameters:

acqState – when the cleaning should happen
cleansForMode – how many cleans to do

Example:

` Setup cleans so that there is a 5 cleans before 1st acq and 0 cleans between other acqs
myDetector.SetMultiAcqCleanCount(jyMultiAcqStateBeforeFirstOnly, 5)
myDetector.SetMultiAcqCleanCount (jyMultiAcqStateBetweenOnly, 0)

See Also: Advanced Topics: Multiple Accumulations

MultiAcqHardwareMode

Function: HRESULT MultiAcqHardwareMode([out,retval]VARIANT_BOOL *enabled);
HRESULT MultiAcqHardwareMode([in]VARIANT_BOOL enabled);
Description: Set/Get whether or not hardware based accumulations are enabled. This feature is hardware dependant. If the physical device you are communicating with and it's controller don't support multiple accumulations, then this flag should not be enabled. This function is a wrapper function around the SetOperatingModeValue function.
See Also: Advanced Topics: Multiple Accumulations, SetOperatingModeValue, IsOperatingModeSupported

MultiAcqTotalTime

Function: HRESULT MultiAcqTotalTime([out,retval]double *totalTime);
HRESULT MultiAcqTotalTime([in]double totalTime);
Description: Set/Get the total time for acquiring data in a multi-acq acquisition.

Note: These parameters only apply if number of accumulations > 1 and hardware-based accumulations are being used.

Parameters:

totalTime – time in default units for the total acquisition

See Also: Advanced Topics: Multiple Accumulations

IJYDeviceReqd

Instrument Setup

Load

Function: HRESULT Load( )
Description: Load should be called after the unique id has been set to the appropriate value. Load sets up the component with the saved parameters based on that unique id. After the component has been "Loaded", the next step is to call OpenCommunications and then Initialize.
Example:

```
` Set the unique id of the object
ccdObject.UniqueID = "CCD1"
` Load the persisted configuration
ccdObject.Load
```

See Also: Save, UniqueId, ReadCommSettings

Save

Function: HRESULT Save( )
Description: After changes have been made to a component (programmatically or through the IJYSystemReqd::Configure() operation), Save can be called to persist those changes.
Example:

```
` Change the comm setting Port number to GPIB:5 override the default settings
ccdObject.UpdateCommSettings(jyCPTPortNum, 5)
ccdObject.UpdateCommSettings(jyCPTCommType, jyGPIB)
` Persist this change
ccdObject.Save
```

See Also: Load, UpdateCommSettings

ReadCommSetting

Function: HRESULT ReadCommSetting([in]enum jyCommParamType, [out,retval]VARIANT* value);
Description: Reads individual communication settings from the current settings for the component based on the specified jyCommTypeParam.
Parameters:

jyCommParamType: which parameter
value: the value containing the parameter setting on successful completion

Example:

```
'Read the current Comm Type and port number
ccdObject.ReadCommSetting( jyCPTPortNum, portNum)
ccdObject.ReadCommSetting( jyCPTCommType, commType )
```

See Also: Save, Load, UpdateCommSetting

UpdateCommSetting

Function: HRESULT UpdateCommSetting([in]enum jyCommParamType, [in]VARIANT newVal);
Description: Updates individual communication settings from the current settings for the component based on the specified jyCommTypeParam.
Parameters:
jyCommParamType: which parameter
newValue: the value to be set to the parameter
Example:

```
` Change the comm setting Port number to GPIB:5 override the default settings
ccdObject.UpdateCommSettings(jyCPTPortNum, 5)
ccdObject.UpdateCommSettings(jyCPTCommType, jyGPIB)
ccd.Save  `Persist change for future use
```

See Also: Save, Load, UpdateCommSetting

SupportFilesPath

Function: HRESULT SupportFilesPath([out, retval] BSTR *path);
HRESULT SupportFilesPath([in] BSTR path);
Description: Sets/Gets the path used for storing files required for support of the device. (Example: for CCD, path for table and configuration files)
Example:

```
`Change the location of the supporting files for this ccd
ccd.SupportFilesPath = "C:\Sptwmax\ccd"
ccd.Save  `Persist change for future use
```

See Also: Save, Load

Establishing/Terminating Communications

OpenCommunications

Function: HRESULT OpenCommunications()
Description: Attempts to establish communication with the device based on the current communication settings.
Example:

```
ccd.UniqueId = "CCD1"
ccd.Load
ccd.OpenCommunications
` At this point, if successful, there is a connection to physical hardware and
you
` need to initialize it
```

See Also: Load, Save, Initialize, OpenCommunicationsEx

OpenCommunicationsEx

Function: HRESULT OpenCommunicationsEx([in]enum jyCommType, [in]int portNum, [in,optional]VARIANT devName, [in, optional]VARIANT bRate, [in, optional]VARIANT databits, [in, optional]VARIANT paritybits, [in, optional]VARIANT jstopbits);
Description: OpenCommunicationsEx allows the user to establish a connection to the device and overload the current communication parameter setting.
Parameters:

jyCommType - Select value from the enumeration
portNum - Port Number (i.e. – COM port # or GPIB address)
devName - Depending on commType, this value has different meanings
bRate - Applies to Serial communications only
dataBits - Applies to Serial communications only
stopBits - Applies to Serial communications only

Note: It is recommended that you use the regular OpenCommunications function.

Initialize

Function: HRESULT Initialize([in, optional]VARIANT forceInit, [in, optional]VARIANT emulate);
Description: This function will initialize the device to a known state. Communications should be established prior to this call using OpenCommunications. If you wish to emulate, simpley specify the emulate parameter as VARIANT_TRUE.
Parameters:

forceInit: Variant of type VT_BOOL specifying whether or not to force the initialization to happen. If this is not set to "VARIANT_TRUE" then initialization is not forced (if device was previously initialized, it's current state will be accepted as initialized)
emulate: Variant of type VT_BOOL. This parameter must be specified as VARIANT_TRUE to put the component into emulation mode (no hardware)

Note: This function is threaded and will return immediately and fire an event when the initialization is complete. Users should wait for successful initialization before interacting with the device.

Example:

```
ccd.UniqueId = "CCD1"
ccd.Load
ccd.OpenCommunications
ccd.Initialize ' Returns immediately.  Caller should wait for the event.
' Example of event handler
Private Sub ccdObject_Initialize(ByVal Status As Long, ByVal eventInfo As
JYSYSTEMLIBLib.IJYEventInfo)
MsgBox( "Hardware Initialized!")
End Sub
```

See Also: Load, OpenCommunications, Uninitialize

Uninitialize

Function: HRESULT Uninitialize( );
Description: Uninitializes the device by calling CloseCommunications and clearing all internal state information about the device. After this call, the device is not longer valid until it is again loaded, communication is established and initialized.
Example:

```
Sub Form_Unload()
ccd.CloseCommunications
ccd.Uninitialize
set ccd = Nothing
End Sub
```

See Also: Initialize, CloseCommunications

CloseCommunications

Function: HRESULT CloseCommunications();
Description: Closes the current communications connection.
Example:

```
Sub Form_Unload()
ccd.CloseCommunications
ccd.Uninitialize
set ccd = Nothing
End Sub
```

See Also: OpenCommunications, Uninitialize

Generic String Communication

SendString

Function: HRESULT SendString([in,string]BSTR stringToSend, [optional, in]VARIANT countToSend);
Description: Allows the user to send an arbitrary string to the device.
Parameters:

stringToSend – This unicode string will be sent to the device with the teminating character appended.
countToSend - The size (in characters) of the string that is being sent

Note: For Future Use. Not thoroughly tested or supported.

ReadString

Function: HRESULT ReadString([in,out]long *charCount, [out, retval]BSTR *stringRead);
Description: Allows the caller to read strings from the device.
Parameters:

charCount - number of characters to read
stringRead – variable to hold the return string

Note: For Future Use. Not thoroughly tested or supported.

PassThruSendTerminationCharacter

Property: HRESULT PassThruSendTerminationCharacter([out, retval] BSTR *pVal);
HRESULT PassThruSendTerminationCharacter([in] BSTR newVal);
Description: Sets/Gets the terminating character to be used for sending strings

Device Properties

FirmwareVerison

Function: HRESULT FirmwareVersion([out, retval] BSTR *pVal)
Description: Read only parameter used to retrieve the firmware version from the currently connected/initialized device.
Example:

```
myControllerVersion = ccd.FirmwareVersion
```

See Also: Name

Emulating

Function: HRESULT Emulating([out,retval]VARIANT_BOOL *pVal)
Description: Read only parameter for querying the device as to whether it is connected to hardware or not.
Example:

```
emulatingIndicator = ccd.Emulating
```

See Also: OpenCommunications, Initialize

Name

Property: HRESULT Name([out, retval]BSTR *name);
HRESULT Name([in]BSTR name);
Description: Sets/Gets the name of the device. This name is not persisted unless the user calls "Save" function.
Example:

```
'read it
devName = ccd.Name
' change and save it
ccd.Name = "My 1024x1024 CCD"
ccd.Save
```

See Also: FirmwareVersion

Miscellaneous Functions

CheckLightPath
Function: HRESULT CheckLightPath([in]long inPort, [in]long outPort, [out, retval] VARIANT_BOOL *ok);
Description: For internal use only.

LastError
Function: HRESULT LastError([out,retval]HRESULT *hr);
Description: For internal use only.

ErrDisplayModeIn
Function: HRESULT ErrDisplayMode([in] enum jyErrDisplayMode newVal);
Description: For internal use only.

ErrDisplayModeOut
Function: HRESULT ErrDisplayMode([out, retval] enum jyErrDisplayMode* pVal);
Description: For internal use only.

IJYEventInfo

Derived from IDispatch

Overview

The IJYEventInfo object is an abstract object that encapsulates the details of various windows events that are fired by components in the SDK.

Code Example

The following is a code example of how to extract information from an event. The initialize event does not have any Result or data, so that is not shown here. See GetResult for example code.

```
Private Sub myObj_Initialize(ByVal status As Long, ByVal eventInfo As
JYSYSTEMLIBLib.IJYEventInfo)
    Dim tmpObj As JYSYSTEMLIBLib.IJYSystemReqd
    Dim stringToAdd As String

    'Extract event description and display status
    stringToAdd = "Event: " + eventInfo.Description
    InfoList.AddItem stringToAdd
    stringToAdd = "Status: " + Str$(status)
    InfoList.AddItem stringToAdd

    ' Extract the source object from the event
    Set tmpObj = eventInfo.Source

    ' Query information from the interface
    stringToAdd = "Source ID: " + tmpObj.Uniqueid
    InfoList.AddItem stringToAdd
    stringToAdd = "Firmware Ver: " + tmpObj.FirmwareVersion
    If (tmpObj.DeviceClass = jyDevClassDetector) Then
        If (tmpObj.DeviceType = jyDevTypeMCD) Then
            stringToAdd = "Device Is Multi Channel Detector"
        Else
            stringToAdd = "Device Is Single Channel Detector"
        End If
    ElseIf (tmpObj.DeviceClass = jyDevClassMono) Then
        stringToAdd = "Device Is Mono"
    Else
        stringToAdd = "Not a mono or detector"
    End If
    InfoList.AddItem stringToAdd
End Sub
```

Source

Function: HRESULT Source([out, retval] IJYDeviceReqd **pVal);
HRESULT Source([in] IJYDeviceReqd *newVal);
Description: This provides access to the source object that fired the event. For example, if you have a generic event handler for more than one component, this will give you the ability to differentiate between events. This is only important if you are programming with more than one instance of a component, otherwise you know the object that fired the event.
Parameters:

pVal – pointer to an interface pointer that will point to the device that fired the event
newVal – Sets the source of the event object (Used internally only. Should not be called by SDK developer)

Example: See Code Example

Description

Function HRESULT Description([out, retval] BSTR *pVal);
HRESULT Description([in] BSTR newVal);
Description: Returns a description of the event provided by the component that fired it.

Note: Not completely implemented. Reserved for future use.

Parameters:

pVal – pointer to string that will receive the description of the event
newVal – Sets the decription of the event. Should only be read by SDK programmer.

Example: See Code Example

Val

Function: HRESULT Val([out, retval] VARIANT *pVal);
HRESULT Val([in] VARIANT newVal);
Description: Returns or sets a variant value to the event.

Note: Not completely implemented. Reserved for future use.

Parameters:

pVal – pointer to variant that will receive the value of the event
newVal – Sets the value of the event. Should only be read by SDK programmer.

AttachData

Function: HRESULT AttachData([in]IJYDataObject *dataObject);
Description: For internal use only.

AttachResults

Function: HRESULT AttachResults([in]IJYResultsObject *resultsObject);
Description: For internal use only.

GetResult

Function: HRESULT GetResult([out,retval]IJYResultsObject **resultsObjectPtr);
Description: Returns an interface ptr to the result encapsulated in the current event. This only pertains to Update events. No other event has a result associated with it. In these cases, this call will return a NULL value.
Parameters:

resultsObjectPtr – pointer to the interface pointer that will receive the result interface

Example:

```
Private Sub myObj_Update(ByVal updateType As Long, ByVal eventInfo As
JYSYSTEMLIBLib.IJYEventInfo)
  Dim myResult As JYSYSTEMLIBLib.IJYResultsObject
  InfoList.AddItem "Received Update Event"
  Set myResult = eventInfo.GetResult
  InfoList.AddItem "Extracted Result"
  ` See Results Document for more details on handling results and data objects
End Sub
```

See Also: Results, Data Object Handling

IJYMonoReqd

Overview

This interface deals with operations specific to Monochromators.

MovetoGrating

Function: HRESULT MovetoGrating([in]double GratingDensity)
Description: Moves turret to a position that holds a grating with specified density.

Note: Recommended to use MoveToTurret instead. Grating density not guaranteed to be non-ambiguous.

Parameters:
GratingDensity is in Grooves/mm.
See Also: MovetoTurret, GetCurrentGrating

GetCurrentGrating

Function: HRESULT GetCurrentGrating([out] double* pGratingDensity, [out] VARIANT* pGratings);
Description: Retrieves the current grating density and a list of all grating densities.
Parameters:
Value pGratingDensity is in Grooves/mm.
Variant pGratings is a safearray of type double.
See Also: MovetoGrating

MovetoWavelength

Function: HRESULT MovetoWavelength([in]double newWavelength);
Description: Moves wavelength to specified value. If a move requires backlash correction, the backlash move is completed by the IsBusy method. The backlash-needed condition sets the busy status of the mono to TRUE. It is completed when IsBusy is called and returns FALSE.
Parameters:
Wavelength units can be set with IJYSystemReqd::SetDefaultUnits. The units are initialized to Nanometers in the component.
Example:

```
'Scan the mono from 500 to 600 nm with steps of 10
myMono.SetDefaultUnits( jyutWavelength, jyuNanometers )
for ( monoPos = 500 to 600 step 10 )
  myMono.MoveToWavelength( monoPos )
  while ( myMono.IsBusy() )
  '   do nothing
  wend
  '  do something, like take data
Next monoPos
```

See Also: GetCurrentWavelength, Calibrate

GetCurrentWavelength

Function: HRESULT GetCurrentWavelength([out, retval] double* pWavelength);
Description: Retrieves the current wavelength value. If the mono is moving, it returns the wavelength value at the moment of executing this method. To read final wavelength, repeatedly call method IsBusy until it returns FALSE.
Parameters:

Wavelength units are set with IJYSystemReqd::SetDefaultUnits. The units are initialized to Nanometers in the component.

Example:

```
` Watch the mono position during a long move
myMono.MoveToWavelength( 0 )
` Wait until we get to start
while ( myMono.IsBuxy() )
wend
dim longMove as double
longMove = 1000
myMono.MoveToWavelength( longMove )
while( monoPosRead < longMove )
  txtDisplay.Text = myMono.GetCurrentWavelength
wend
```

See Also: MovetoWavelength, Calibrate

Calibrate

Function: HRESULT Calibrate([in]double actualWavelength,[in,optional] BOOL positionInBaseGrating)
Description: Sets the current wavelength to the specified value. The setting persists until the mono is reinitialized. This command is used to correct the actual position of the monochromator. For non-autocalibrating monochromators, a first pass calibration should be done when the instrument is powered on.
Parameters:

actualWavelength: The actual wavelength the mono is at (based on the observed peak). If this value is being entered based on a physical counter on the monochromator, then this value must be in BASE UNITS for the BASE GRATING. Otherwise, it should be in the current working units for the monochramator.

positionInBaseGrating: If you are calibrating a device where the calibration position is being entered based on a counter, then you can set this value to TRUE and the calibration will take place in base grating units automatically. Otherwise, the units of the calibration function will be those that are currently set for the mono.

Note: Changing this parameter without a complete understanding may result in hardware damage.

See Also: MovetoWavelength, GetCurrentWavelength

MovetoSlitWidth

Function: HRESULT MovetoSlitWidth([in]enum SlitLocation sl, [in]double newWidth);
Description: Moves a slit to a specified width.
Parameters:

Slit location can be the enumerated values
Front_Entrance = 0,
Side_Entrance =1,
Front_Exit =2,
Side_Exit =3,
First_Intermediate =4,
Second_Intermediate =5
Slit width is in units set by method IJYSystemReqd::SetDefaultUnits. The units are initialized to millimeters in the component.

See Also: GetCurrentSlitWidth, CalibrateSlitWidth, SetDefaultUnits, GetDefaultUnits

GetCurrentSlitWidth

Function: HRESULT GetCurrentSlitWidth([in]enum SlitLocation sl, [out, retval] double* p Retrieves Val);
Description: If the slit is in motion, the value retrieved is the width at the moment of executing this method. If the move requires slit backlash, the busy status will be set to TRUE and the backlash move is completed when IsBusy returns FALSE.
Parameters:

Slit location can be the enumerated values
Front_Entrance = 0,
Side_Entrance =1,
Front_Exit =2,
Side_Exit =3,
First_Intermediate =4,
Second_Intermediate =5
Slit width is in units set by method IJYSystemReqd::SetDefaultUnits. The units are initialized to millimeters in the component.

See Also: MovetoSlitWidth, CalibrateSlitWidth

CalibrateSlitWidth

Function: HRESULT CalibrateSlitWidth([in]enum SlitLocation slitNumber);
Description: Sets the specified slit width to zero.
Parameters:

Slit location can be the enumerated values
Front_Entrance = 0,
Side_Entrance =1,
Front_Exit =2,
Side_Exit =3,
First_Intermediate =4,
Second_Intermediate =5

See Also: MovetoSlitWidth, GetCurrentSlitWidth

IsBusy

Function: HRESULT IsBusy([out, retval]VARIANT_BOOL* BusyStatus);
Description: Checks the busy status of the monochromator.
Parameters:

BusyStatus is TRUE if the grating or any accessories are still moving, or a backlash correction is needed for wavelength or slit. BusyStatus if FALSE when no motor is moving and no backlash flagged.

See Also: IsReady

MovetoMirrorPosition

Function: HRESULT MovetoMirrorPosition([in]enum MirrorLocation MirrorNumber, [in]enum MirrorLocation NewMirrorPosition);
Description: Moves the specified mirror to the specified position.
Parameters:

MirrorNumber can be one of the following values
EntranceMirror = 0
ExitMirror =1
NewMirrorPosition can be one of the following values
Front =2,
Side =3

See Also: GetCurrentMirrorPosition

GetCurrentMirrorPosition

Function: HRESULT GetCurrentMirrorPosition([in]enum MirrorLocation MirrorNumber, [out, retval]enum MirrorLocation* pMirrorPostion);
Description: Retrieves the position of the specified mirror.
Parameters:

MirrorNumber can be one of the following values
EntranceMirror = 0
ExitMirror =1
pMirrorPosition can be a reference to one of the following values
Front =2,
Side =3

See Also: MovetoMirrorPosition

OpenShutter

Function: HRESULT OpenShutter();
Description: Opens the "active" shutter. If the Entrance Mirror is in the Side (Lateral) position, the "active" shutter is the Side Entrance Shutter. On the other hand, if the Entrance Mirror is in the Frunt (Axial) position, or if there is no Entrance Mirror, the Front Entrance shutter in "active".
Parameters:
See Also: CloseShutter, GetCurrentShutterPosition

CloseShutter

Function: HRESULT CloseShutter();
Description: Closes the "active" shutter. Refer to the OpenShutter method for more information.
Parameters:
See Also: OpenShutter, GetCurrentShutterPosition

GetCurrentShutterPosition

Function: HRESULT GetCurrentShutterPosition([out, retval]enum OpenOrClose* pShutterPosition);
Description:
Parameters:
pShutterPosition refers to one of the following values:
ShutterClose = 0,
ShutterOpen = 1
See Also: OpenShutter, CloseShutter, GetCurrentShutterPosition

Stop

Function: HRESULT Stop();
Description: Stops the monochromator or slits.

IsReady

Function: HRESULT IsReady([out, retval] VARIANT_BOOL* ReadyStatus);
Description: Retrieves status of the monochromator's controller. Sending commands to a controller while the controller-issued moves are still in process may result in controller's command error. IsReady and IsBusy may return independ values in the case of multiplex controllers. For example, a DataScan controller may be used to controller two monochromators, Mono1 and Mono2. In the case that Mono1 is moving and Mono2 is not moving, the following values will be returned:
IsBusy IsReady
Mono1 TRUE FALSE
Mono2 FALSE FALSE
Parameters:
The returned ReadyStatus is TRUE if all the moves issued by the controller are completed. The returned ReadyStatus is FALSE if all the moves issues by the controller are not completed.
See Also: IsBusy

IsSubItemInstalled

Function: HRESULT IsSubItemInstalled([in] enum MonoSubItemType type, [out, retval]VARIANT_BOOL* installed);
Description: Retrieves the configuration of the monochromator regarding installation of difference items. Configuration information can be set using IJYSystemReqd::Configure.
Parameters:
The following values can be used for type:
Slit_Front_Entrance =0,
Slit_Side_Entrance =1,
Slit_Front_Exit =2,
Slit_Side_Exit =3,

Slit_First_Intermediate =4,
Slit_Second_Intermediate =5,
Mirror_Entrance =6,
Mirror_Exit =7,
MonoDrive =8,
Turret =9,
Front_Shutter =10,
Side_Shutter =11

See Also: **IJYSystemReqd:Configure**

GetCurrentGratingWithDetails

Function: HRESULT GetCurrentGratingWithDetails([out] double* pGratingDensity, [out] VARIANT* pGratings, [out]VARIANT* pBlaze, [out]VARIANT* pDescription);
Description: Retrieves current grating density, and lists of grating densities, Blaze information, and Descriptions.
Parameters:

pGratingDensity is density in Grooves/mm.
Variant pGratings is a safearray of type double.
Variant pBlaze is a safearray of type BSTR.
Variant pDescription is a safearray of type BSTR.

MovetoTurret

Function: HRESULT MovetoTurret([in] int turretNumberZeroBased);
Description: Moves to a turret position as specified.
Parameters:

turretNumberZeroBased is zero-based turret position.

See Also: MovetoGrating, GetCurrentTurret

GetCurrentTurret

Function: HRESULT GetCurrentTurret([out, retval] int* turretNumberZeroBased);
Description: Retrieves turret position.
Parameters:

The returned value turretNumberZeroBased points to zero-based turret position.

IsTargetWithinLimits

Function: HRESULT IsTargetWithinLimits([in]enum jyMonoMoveType what, [in] double where, [out] VARIANT* withinLimits, [out] double* minForCurrentGrating, [out] double* minForCurrentGrating, [in, optional] VARIANT whichSlit);
Description: Checks whether the target position is within the allowed physical range of the hardware. It also returns the high and low limits that can be used on user interface. The low and high limits are given in the current units and ,for the case of wavelength, for the current grating.
Parameters:

Value "what" can have one of the following values
jyMonoMoveTypeWavelength,
jyMonoMoveTypeSlit
Value "where" is the target position of type double in default units.
Value WithinLimits returns VARIANT_BOOL.

Value minForCurrentGrating is the low limit for the current grating in current units
Value maxForCurrentGrating is the high limit for the current grating in current units
Value whichSlit specifies the slit when checking slit ranges.

SetJYLoggerProperties

Function: HRESULT SetJYLoggerProperties([in]enum logLevel newLevel ,[in, optional] BSTR newFileName ,[in, optional] BSTR newFilePath ,[in, optional] int newMaxBackupFiles,[in, optional] int newMaxFileSize);
Description: For internal use only.

GetGratingMotorSpeeds/SetGratingMotorSpeeds

Function: HRESULT GetGratingMotorSpeeds([out]double* pFrequencyMin, [out]double* pFrequencyMax, [out]double* pRampTime);
HRESULT SetGratingMotorSpeeds([in]double FrequencyMin, [in]double FrequencyMax, [in]double RampTime);
Description: For internal use only.

SlitMotorSpeed

Function: HRESULT SlitMotorSpeed([in]enum SlitLocation sl, [out, retval]double* pFrequency);
HRESULT SlitMotorSpeed([in]enum SlitLocation sl, [in] double Frequency);
Description: For internal use only.

IJYSCDReqd

Overview

This interface deals with operations specific to Single Channel Detectors. An object that supports this interface must also support IJYDetectorReqd, IJYSystemReqd, IJYDeviceReqd.

OpenShutter

Function: HRESULT OpenShutter();
Description: Opens the shutter controlled by this device.
Parameters: None
Example:

```
if ( doBlank ) then
  scd.CloseShutter()
else
  scd.OpenShutter()
endif
```

See Also: CloseShutter

CloseShutter

Function: HRESULT CloseShutter();
Description: Closes the shutter controlled by this device.
Parameters: None
Example:

```
if ( doBlank ) then
  scd.CloseShutter()
else
  scd.OpenShutter()
endif
```

See Also: OpenShutter

Gain

Function: HRESULT Gain([out, retval] short *gainVall);
HRESULT Gain([in] short gainVal);
Description: Sets the gain for the detector. The value sent should be from the enumerated list of gains supported by this detector.
Parameters: gainVal – the value to be used for setting/getting the current gain.
Example:

```
gainToken = myListBox.GetItemData( myListBox.GetCurSel() )
myDetector.Gain = gainToken
```

See Also: Enumerations, Token String Pairs

Amplifier

Function: HRESULT Amplifier([out, retval] short *pVal);
HRESULT Amplifier([in] short newVal);
Description: To be defined.

DataUnitsOut

Function: HRESULT DataUnits([out, retval] short *pVal);
Description: Obsoleted.
e IJYSystemReqd::GetDefaultUnits(jyUnitTypeData, pVal)

DataUnitsIn

Function: HRESULT DataUnits([in] short newVal);
Description: Obsoleted. Use IJYSystemReqd::SetDefaultUnits(jyUnitTypeData, newVal)

TimeUnitsOut

Function: HRESULT TimeUnits([out, retval] short *pVal);
Description: Obsolete. Use IJYSystemReqd::GetDefaultUnits (jyUnitTypeTime, pVal)

TimeUnitsIn

Function: HRESULT TimeUnits([in] short newVal);
Description: Obsoleted. Use IJYSystemReqd::GetDefaultUnits (jyUnitTypeTime, newVal)

Bias

Function: HRESULT Bias([out, retval] long *biasVal);
HRESULT Bias([in] long biasVal);
Description: Set the bias or high voltage, if supported by this detector.
Parameters:
biasVal – the variable used to get/set the gain
Example:
myDetector.Bia = 950

ChannelNumber

Function: HRESULT ChannelNumber([out, retval] short *channelVal);
HRESULT ChannelNumber([in] short channelVal);
Description: Get or set the acquisition channel number for this detector. This is a configuration parameter and should not be changed without understanding the details of the hardware that is attached. If the detector is configured properly, this value should not be changed.

Note: Primarily for internal use.

Parameters:
channelVal – variable used to get/set the channel number

GetOffsetforGain

Function: HRESULT GetOffsetForGain([in] short gain, [out, retval] long *offset);
Description: Gets the detector offset for a specific gain setting. This is a configuration parameter and should not be changed without understanding the details of the hardware that is attached. If the detector is configured properly, this value should not be changed.

Note: Primarily for internal use.

Parameters:

gain – gain value of interest
offset – detector offset at this gain

GetVoltageforGain

Function: HRESULT GetVoltageForGain([in] short gain, [out, retval] double *voltage);
Description: Gets the voltage conversion factor for a specific gain setting. This is a configuration parameter and should not be changed without understanding the details of the hardware that is attached. If the detector is configured properly, this value should not be changed.

Note: Primarily for internal use.

Parameters:

gain – gain value of interest
voltage – voltage conversion factor at this gain

GetCurrentforGain

Function: HRESULT GetCurrentForGain([in] short gain, [out, retval] double *current);
Description: Gets the current conversion factor for a specific gain setting. This is a configuration parameter and should not be changed without understanding the details of the hardware that is attached. If the detector is configured properly, this value should not be changed.

Note: Primarily for internal use.

Parameters:

gain – gain value of interest
current – current conversion factor at this gain

SetOffsetforGain

Function: HRESULT SetOffsetForGain([in] short gain, [in] long offset);
Description: Sets the detector offset for a specific gain setting. This is a configuration parameter and should not be changed without understanding the details of the hardware that is attached. If the detector is configured properly, this value should not be changed.

Note: Primarily for internal use.

Parameters:

gain – gain value of interest
offset – detector offset at this gain

SetVoltageforGain

Function: HRESULT SetVoltageForGain([in] short gain, [in] double voltage);
Description: Sets the voltage conversion factor for a specific gain setting. This is a configuration parameter and should not be changed without understanding the details of the hardware that is attached. If the detector is configured properly, this value should not be changed.

Note: Primarily for internal use.

Parameters:

gain – gain value of interest
voltage – voltage conversion factor at this gain

SetCurrentforGain

Function: HRESULT SetCurrentForGain([in] short gain, [in] double current);
Description: Sets the voltage conversion factor for a specific gain setting. This is a configuration parameter and should not be changed without understanding the details of the hardware that is attached. If the detector is configured properly, this value should not be changed.

Note: Primarily for internal use.

Parameters:

gain – gain value of interest
current – current conversion factor at this gain

IJYSystemReqd

Configure

Function: HRESULT Configure()
Description: Launches a device-specific configuration dialog allowing customization of operating parameters.
Example:

```
' Launch the devices internal configuration dialog
m_sysReqd.Configure()
```

See Also: Setup

Setup

Function: HRESULT Setup()
Description: Launches a device-specific setup operation that allows basic interaction with a device that has been connected to and initialized.
Example:

```
` Launch the devices internal setup dialog
m_sysReqd.Setup()
```

See Also: Configure

UniqueId

Property: HRESULT Uniqueid([out, retval]BSTR *uniqueId)
HRESULT Uniqueid([in]BSTR uniqueId);
Description: This property sets/gets the unique id of the device. This ID is controlled by the original installation and configuration and should not be modified by the user. This id is used to uniquely identify a device when multiple devices are involved.
Parameters:

uniqueID – a string value that uniquely identifies the device in the context of the current PC

Example:

```
‘Compare interfaces for equality
if ( sysReqd1.Uniqueid = sysReq2.Uniqueid )
   MsgBox( “Interfaces reference the same object”)
Else
   MsgBox(“Interfaces DO NOT reference the same object”)
Endif
```

See Also: Load, OpenCommunications

DeviceClass

Function: HRESULT DeviceClass([out, retval] enum jyDeviceClass *devClass)
Description: Retrieves the class of the device. The “Class”is the top-level category of devices, including monos, detectors, accessories, and light sources.
Parameters:

devClass – the class to which this device belongs {see jyDeviceClass }

Example:

```
'Check if the given device is a detector
if ( m_systemReqd.DeviceClass( ) = jyDevClassDetector ) then
MsgBox( "This Device is a Detector" )
Else
MsgBox("This Device is NOT a detector");
Endif
```

See Also: DeviceType

DeviceType

Function: HRESULT DeviceType([out, retval] enum jyDeviceType *devType)
Description: Retrieves the device type of the device. The "Type" is the sub-category of device. Types include CCD, Single Channel Detector (types of detectors), Filter Wheel (type of accessory).
Parameters:

devType – the type of device (combined with DeviceClass, identifies a specific device type Detector : CCD) {see jyDeviceType }

Example:

```
'Check if the device is a CCD Detector
if( m_systemReqd.DeviceClass() = jyDevClassDetector and
m_systemReqd.DeviceType() = jyDevTypeMCD ) then
MsgBox( "This Device is a Multi channel Detector" )
Else
MsgBox("This Device is NOT a multi channel detector ");
Endif
```

See Also: DeviceClass

GetFirstSupportedUnits/GetNextSupportedUnits

Function: HRESULT GetFirstSupportedUnits([in]enum jyUnitsType unitsType, [out]enum jyUnits* pVal, [out, optional]VARIANT* pStrVal);
HRESULT GetNextSupportedUnits([in]enum jyUnitsType unitsType, [out]enum jyUnits* pVal, [out, optional]VARIANT* pStrVal);
Description: These functions enumerate all the types of units supported by the device for things like wavelength, data, time, temperature, etc… GetFirst gets the first unit of the specified unit type. GetNext gets the next unit of that specified type. You can not interweave calls for different jyUnitsTypes (i.e. – Do not call GetFirst for temperature, GetFirst for wavelength and then GetNext for temperature. Results of this are undefined).
Parameters:

unitsType – the type of units you are trying to enumerate {see jyUnitsType }
pVal – the token value for the units to be used in selecting the default units for any device {set jyUnits }
pStrVal – String representation of the units value

Example:

```
'Fill a list box with the available list of data unit types and select the current
units
   Dim unitName As String
   Dim unitToken As Long
   Dim currentDataUnits as long
   ccdObject.GetDefaultUnits( jyutDataUnits, currentDataUnits )
   ccdObject.GetFirstSupportedUnits( jyutDataUnits, unitToken, unitName)
```

```
If (token = -1) Then
   unitList.AddItem "NONE"
   unitList.Enabled = False
Else
   While (Not token = -1)
      unitList.AddItem (unitName)
      unitList.ItemData(unitList.NewIndex) = token
      If (currentDataUnits = token) Then
         unitList.ListIndex = unitList.NewIndex
      End If
      token = ccdObject.GetNextSupportedUnits(jyutDataUnits, unitToken,
unitName)
   Wend
End If
```

See Also: Enumerations, Token-String Pairs, GetDefaultUnits, SetDefaultUnits

GetDefaultUnits/SetDefaultUnits

Function: HRESULT GetDefaultUnits([in]enum jyUnitsType type, [out]enum jyUnits* pVal, [out, optional]VARIANT* pStrVal);
HRESULT SetDefaultUnits([in]enum jyUnitsType type, [in]enum jyUnits newVal);
Description: Gets/Sets the default units for a specific unit type. All operations using that category of units will expect the values to be presented in the default units.
Parameters:

unitsType – the type of units you are trying to enumerate {see jyUnitsType }
pVal – the token value for the units to be used in selecting the default units for any device {set jyUnits }
pStrVal – String representation of the units value

Example:

```
Dim currentDataUnits as long
Dim currentDataUnitsDisplay as String
ccdObject.GetDefaultUnits( jyutDataUnits, currentDataUnits,
currentDataUnitsDisplay)
MsgBox( "Current data units are" + currentDataUnitsDisplay )
```

See Also: Enumerations, Token-String Pairs, GetFirstSupportedUnits

Shutdown

Function: HRESULT Shutdown([in,optional]long msToWait);
Description: Performs an emergency shutdown of the device, terminating any waiting threads after msToWait milliseconds. This should only be used in extreme cases, where hardware is potentially hung. Once this is called, any application using the component should be shutdown and restarted. Power cycling of the hardware may also be required.
Parameters:

msToWait – time in milliseconds to wait for pending threads to terminate before forcibly terminating them.

Example:

```
` Started acquisition, but have not received data on threaded acquisition after
1 minute
m_SystemReqd.Shutdown( 100 )
```

See Also: Uninitialize, CloseCommunications

Description

Property: HRESULT Description([out, retval]BSTR *desc);
HRESULT Description([in]BSTR desc);
Description: Sets/Gets a detailed description of the device.
Parameters:
desc – text based description of the device.
Example:
m_ListBox.AddString(m_sysReqd.Decription)
See Also: Name, Uniqueid, DeviceClass, DeviceType

GetDeviceConfigProperty/SetDeviceConfigProperty

Function: HRESULT GetDeviceConfigProperty([in]enum jyDeviceConfigProperty property, [out, retval] VARIANT* pVal);
HRESULT SetDeviceConfigProperty([in]enum jyDeviceConfigProperty property, [in] VARIANT newVal, [in, optional] VARIANT newVal2, [in, optional] VARIANT newVal3, [in, optional] VARIANT newVal4);
Description: Gets/Sets the properties of the device based on the jyDeviceConfigProperty value. { see jyDeviceConfigProperty } Parameters required for Set are dependant on the property. This is an advanced operation and improper usage can cause problems with the hardware.
Parameters:
property – the property value to retrieve
pVal – the value of the property requested.
newVal – value to use in setting the property
newVal2 – optional value to use in setting the property
newVal3 – optional value to use in setting the property
newVal4 – optional value to use in setting the property
Example:
'Get the type of the SCD object
m_scd.GetDeviceConfigProperty(jyDetectorSCDType, &scdType);

GetConverterReference

Function: HRESULT GetConverterReference([out]IJYConverter** pVal)
Description: For internal use only.

ControllerChannelIndex

Property: HRESULT ControllerChannelIndex([out, retval] VARIANT* pVal);
HRESULT ControllerChannelIndex([in] VARIANT newVal);
Description: For internal use only.

Controller Specific Operations

Overview

These operations are very specific to a controller and are for engineering and test purposes. Not to be used unless completely understood and with direct support from engineering.

GetFirstControllerOperations/GetNextControllerOperation

Function: HRESULT GetFirstControllerOperation([out]enum jyKnownControllerSpecificOperations *token, [out]BSTR *description, [out]double *currentValue);
HRESULT GetNextControllerOperation([out]enum jyKnownControllerSpecificOperations *token, [out]BSTR *description, [out]double *currentValue);
Description: Enumerates all of the operations that are specific to the currently attached controller. Usage of these operations assumes specific knowledge of the device with which you are communicating.
For internal use only

SetControllerOperationValue/GetControllerOperationValue

Function: HRESULT SetControllerOperationValue([in]enum jyKnownControllerSpecificOperations token, [in]double newValue);
HRESULT GetControllerOperationValue([in]enum jyKnownControllerSpecificOperations token, [out]double *lastSet, [out,retval]double *value);
Description: Sets/Gets the values for a specific controller operation.
For internal use only

IsControllerOperationSupport

Function: HRESULT IsControllerOperationSupported([in]enum jyKnownControllerSpecificOperations whichOperation, [out,retval]VARIANT_BOOL *supported);
Description: Checks to see if a specific controller operation is supported on the currently connected device.
For internal use only

Operating Modes

Overview

Operating Modes are Device-Oriented capabilities that some devices in this category will support and others will not.

GetFirstOperatingMode/GetNextOperatingMode

Function: HRESULT GetFirstOperatingMode([out]BSTR *modeName, [out,retval]long *modeToken);
HRESULT GetNextOperatingMode([out]BSTR *modeName, [out,retval]long *modeToken);
Description: Enumerates the supported operating modes of the connected device.
For internal use only

GetOperatingModeValue/SetOperatingModeValue

Function: HRESULT GetOperatingModeValue([in]enum jyDeviceOperatingMode whichOpMode, [out,retval]VARIANT *pOpModeVal);
HRESULT SetOperatingModeValue([in]enum jyDeviceOperatingMode whichOpMode, [in]VARIANT newOpModeVal);
Description: Gets/Sets the operating modes.
For internal use only

IsOperatingModeSupported

Function: HRESULT IsOperatingModeSupported([in]enum jyDeviceOperatingMode opModeToCheck, [out,retval]VARIANT_BOOL *isSupported);
Description: Checks to see if a desired operating mode is supported by the current device.
For internal use only

Triggers

Overview

Triggers are handled by various devices based on the specifics of the controller to which they are attached. To facilitate generic handling of triggers, we allow for each device to enumerate the supported capabilities. The format of triggers is:

DEVICE: ADDRESS: EVENT: SIGNALTYPE. For example, Device = CCD3000, Address = OutputTrigger1, Event = OnStartEachAcq, SignalType = TTL High.

Think of triggers as a three-tier hierarchy with TriggerAddress as the root. For each TriggerAddress there can be one or more TriggerEvents. For each TriggerEvent, there can be one or more TriggerSignalTypes. The GetFirst/GetNext enumerations give you a way to navigate this tree of trigger capabilities.

{***See Advanced Topics: Triggering***}

Input Triggers

Function: HRESULT GetFirstSupportedInputTriggerAddress([out]long *trigAddress, [out]BSTR *trigAddressString);

HRESULT GetNextSupportedInputTriggerAddress([out]long *trigAddress, [out]BSTR *trigAddressString);

HRESULT GetFirstSupportedInputTriggerEvent([in]long trigAddress, [out]enum jyTriggerEvent *eventPtr, [out]BSTR *trigEventString);

HRESULT GetNextSupportedInputTriggerEvent([in]long trigAddress, [out]enum jyTriggerEvent *eventPtr, [out]BSTR *trigEventString);

HRESULT GetFirstSupportedInputTriggerSignalType([in]long trigAddress, [in]enum jyTriggerEvent event, [out]enum jyTriggerSignalType *trigSigType, [out]BSTR *trigSigTypeString);

HRESULT GetNextSupportedInputTriggerSignalType([in]long trigAddress, [in]enum jyTriggerEvent event, [out]enum jyTriggerSignalType *trigSigType, [out]BSTR *trigSigTypeString);

HRESULT EnableInputTrigger([in]long trigAddress, [in]enum jyTriggerEvent event, [in]enum jyTriggerSignalType sigType);

HRESULT DisableInputTrigger([in]long trigAddress, [in]enum jyTriggerEvent event, [in]enum jyTriggerSignalType sigType);

HRESULT DisableAllInputTriggers();

HRESULT IsInputTriggerSupported([in]long trigAddress, [in]enum jyTriggerEvent event, [in]enum jyTriggerSignalType sigType, [out, retval]VARIANT_BOOL *isSupported);

HRESULT IsInputTriggerEnabled([in]long trigAddress, [in]enum jyTriggerEvent event, [in]enum jyTriggerSignalType sigType, [out, retval]VARIANT_BOOL *isEnabled);

Output Triggers

Functions: HRESULT GetFirstSupportedOutputTriggerAddress([out]long *trigAddress, [out]BSTR *trigAddressString);

HRESULT GetNextSupportedOutputTriggerAddress([out] long *trigAddress, [out]BSTR *trigAddressString);

HRESULT GetFirstSupportedOutputTriggerEvent([in]long trigAddress, [out]enum jyTriggerEvent *eventPtr, [out]BSTR *trigEventString);

HRESULT GetNextSupportedOutputTriggerEvent([in]long trigAddress, [out]enum jyTriggerEvent *eventPtr, [out]BSTR *trigEventString);

HRESULT GetFirstSupportedOutputTriggerSignalType([in]long trigAddress, [in]enum jyTriggerEvent event, [out]enum jyTriggerSignalType *trigSigType, [out]BSTR *trigSigTypeString);

HRESULT GetNextSupportedOutputTriggerSignalType([in]long trigAddress, [in]enum jyTriggerEvent event, [out]enum jyTriggerSignalType *trigSigType, [out]BSTR *trigSigTypeString);

HRESULT EnableOutputTrigger([in]long trigAddress, [in]enum jyTriggerEvent event, [in]enum jyTriggerSignalType sigType);

HRESULT DisableOutputTrigger([in]long trigAddress, [in]enum jyTriggerEvent event, [in]enum jyTriggerSignalType sigType);

HRESULT DisableAllOutputTriggers();

HRESULT IsOutputTriggerSupported([in]long trigAddress, [in]enum jyTriggerEvent event, [in]enum jyTriggerSignalType sigType, [out, retval]VARIANT_BOOL *isSupported);

HRESULT IsOutputTriggerEnabled([in]long trigAddress, [in]enum jyTriggerEvent event, [in]enum jyTriggerSignalType sigType, [out, retval]VARIANT_BOOL *isEnabled);

Events

Initialized

Event: HRESULT Initialized([in]long status, [in]IJYEventInfo *eventInfo)
Description: This event is fired when a device has either completed initialization successfully or failed for some reason after IJYDeviceReqd::Initialize(..) has been called.
Subscribe to this event if you want know when the initialization is complete.
Parameters:

Status – flag indicating the status of initialization.
eventInfo – Interface to a IJYEventInfo object with more details about the event.

See Also: IJYEventInfo, IJYDeviceReqd:Initialize()

OperationStatus

Event: HRESULT OperationStatus([in]long status, [in]IJYEventInfo *eventInfo)
Description: These events are fired for various reasons when the status of a device has changed. These events are purely informational and can be ignored. The purpose of these status updates Is to allow the developer to provide useful feedback to the user interface while operations are taking place. Subscribe to this event if you want updates on status changes
Parameters:

status – flag regarding the status
eventInfo - Interface to a IJYEventInfo object with more details about the event.

See Also: IJYEventInfo

Update

Event: HRESULT Update([in]long updateType , [in]IJYEventInfo *eventInfo)
Description: Updates are events used to indicate to the user that an event has occurred that might/might not require additional action on the receiver's part. The primary event currently used by the components is a "Data Update". Subscribe to this event to receive data and other updates. The information relating to the update is encapsulated in the IJYEventInfo object.
Parameters:

status – flag regarding the status
eventInfo - Interface to a IJYEventInfo object with more details about the event.

See Also: IJYEventInfo, DoAcquistion()

CriticalError

Event: HRESULT CriticalError([in]long status , [in]IJYEventInfo *eventInfo)
Description: These events are fired when a critical error is detected that the component cannot recover from. (Currently not fired from anywhere in the framework)
Parameters:

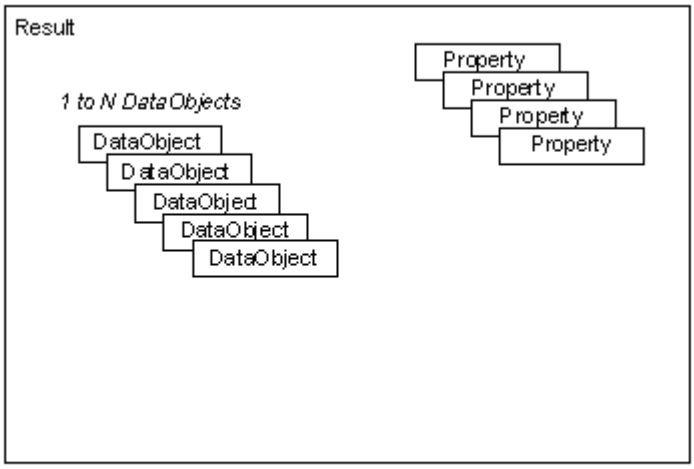
status – flag regarding the status
eventInfo - Interface to a IJYEventInfo object with more details about the event.

See Also: IJYEventInfo

Results and Subcomponents

Overview

Acquisitions and experiments will produce a Result Objects. These result objects consist of one or more Data Objects, depending on the acquisition parameters.



- Result and Result Provider Interfaces
- Data Object and Data Object Provider Interfaces

Result and Result Provider Interfaces

Overview

The IJYResultsObject is a generic interface for receiving results of device operations (primarily related to data acquisition and experimental operations). A Result consists of one or more DataObjects and a set of readable properties. An acquisition from any data-based device (CCD or Single Channel Detector) will construct and provide its data in the form of a result. The caller can then process the result as required, by reading the properties of the result and extracting the data from the result in the form of data objects.

The Result interface is designed for a consumer of the result (ie. – "Read Only" access). The ResultProvider interface is designed for those who plan to construct their own result or perform advanced data operations on a result.

IJYResultsObject

Derived From IDispatch

Save

Function: HRESULT Save([in]BSTR fileName);
Description: Saves the result into the currently specified file format.
Parameters:

fileName – name of the file that you want to save the result in. The filename/extension should match the FileType property value (i.e. – fileType = jySPC, filename = c:\myfile.spc)

Note: This function currently only works for results with one data object. Only the last data object is saved for results having more than one data object. Use the save operation of the individual data objects and determine your own naming convention.

Example:

```
` Save the result to a file
myResult = eventInfo.GetResult()
myResult.FileType = jySPC
myResult.Save( myFileName )
```

See Also: FileType, Load, IJYEventInfo

Load

Function: HRESULT Load([in]BSTR fileName);
Description: Loads data from a specified file into the result. !! NOT CURRENTLY IMPLEMENTED !!
Parameters:

filename – name of the file containing results info

GetFirstDataObject/GetNextDataObject

Function: HRESULT GetFirstDataObject([out,retval] IJYDataObject **dataObj);
HRESULT GetNextDataObject([out,retval] IJYDataObject **dataObj);
Description: Enumerates all the data objects within the result object. A NULL value indicates the end of the list.
Parameters:

dataObj – Interface to the data object

Example:

```
` Handling an update event with a multi-area result
myResult = eventInfo.GetResult
set dataObj = myResult.GetFirstDataObject
while ( not dataObj  is nothing )
   MyProcessingOfDataObject( dataObj )
   set dataObj = myResult.GetNextDataObject
wend
```

See Also: IJYDataObject, Enumerations

FileType

Function: HRESULT FileType([out, retval]enum jySupportedFileType *fileType);
HRESULT FileType([in]enum jySupportedFileType fileType);
Description: Sets/Gets the current file type to be used for Save operations.
Parameters:

fileType – indicates the type of file format to be used when saving

Example:

```
` Save the result to a file
myResult = eventInfo.GetResult()
myResult.FileType = jySPC
myResult.Save( myFileName )
```

See Also: Save

Description

Function: HRESULT Description([in]BSTR description);
HRESULT Description([out, retval]BSTR *description);
Description: Text description of the result, optionally filled in by the provider of the result.
Parameters:

description – optional description of the result

Example:

```
myResult.Description
```

DoOperation

Function: HRESULT DoOperation([in]enum jyOperation operation, [in]VARIANT operand1, [in,optional]VARIANT operand2);
Description: This function can be used to perform post processing on the Results Objects, either with constants or other results or data objects. This function currently supports basic math operations, but will be extended in the future.
Parameters:

operation – specifies the operation to perform on the current result
Supported operations = jyAdd, jySubtract, jyMultiply, jyDivide,

Operand1 – either a constant value or an interface to another result object to use in the operation
Operand2 – either a constant value or an interface to another result object to use in the operation

Example:

```
'Add one scan to another
set firstResult = MyDoScan() ' function to do the single scan and return the result
set secondResult = MyDoScan()
firstResult.DoOperation( jyAdd, secondResult )
' firstResult is now has secondResult added into it.
firstResult.DoOperation( jyDivide, 2 )
' firstResult is now the average of the original firstResult and secondResult
```

See Also: PerformAnalysis

DataObjectCount

Function: HRESULT DataObjectCount([out, retval] long *count);
Description: Returns the count of how many data objects are in this result object.
Parameters:

count – number of encapsulated data object in this result

Example:

```
howManyDataObjects = myResult.DataObjectCount
```

See Also: GetFirstDataObject, GetNextDataObject

InterpretAsImage

Function: HRESULT InterpretAsImage([in]long maxXCount, [in]long maxYCount, [out]VARIANT *image);
Description: This function takes a result for which the data objects consist of areas of a CCD detector and puts these data objects into an image that is maxXCount by maxYCount in size. Normally, when using multiple areas of a chip, you would use this function with the chip size for maxXCount and maxYCount, giving you an image that corresponds to the full chip, with non-data areas zero filled.
Parameters:

maxXCount – the maximum x dimension of the resulting image
maxYCount – the maximum y dimension of the resulting image
image – a Variant that will contain a SAFEARRAY of data representing the full image.

Example:

```
' Have a result with mulitple areas mapped within the chip and want to retrieve the data in
' the context of the chip
dim dataAsImage as variant
myCCD.GetChipSize( x, y )
myResult.InterpretAsImage( x, y, dataAsImage )
'Data as image contains an array of size 'x by y' with non-data areas zero filled
```

GetDataObjectbyIndex

Function: HRESULT GetDataObjectByIndex([in]long dataObjectIndex, [out] IJYDataObject **dataObj);
Description: Based on DataObjectCount, this function allows the caller acquire handles to specific data objects. See the IJYDataObject specification for more details. For Internal Use Only.

GetProperty

Function: HRESULT GetProperty([in]enum jyResultsProperty whichProperty, [out,retval]VARIANT *propValue);
Description: Allows the caller to extract property information that has been provided by the component that produced the result.
Parameters:

whichProperty – identifies the property that you want a value for
Values → jyRPDescription, jyRPHeader, jyRPTime,jyRPDate, jyRPSaveMode,
propValue – the value of the requested property

See Also: SetProperty

MakeCopy

Function: HRESULT MakeCopy([out,retval] IJYResultsObject **resultObject);
Description: Duplicates the current result object into a new result object. This is a deep copy, duplicating all of the encapsulated data objects.
Parameters:

resultObject – the interface pointer that will reference the copy on successful completion.

Example:

```
Set myResult = DoMyScan()
set copyOfResult = myResult.MakeCopy()
set myResult = DoMyScan()
` now have 2 results. If we did not copy it, the first result would be lost.
```

CurrentCycle

Function: HRESULT CurrentCycle([out, retval]long *cycle);
Description: For internal use only.

DumptoFile

Function: HRESULT DumpToFile(BSTR fileName);
Description: For internal use only.

IJYResultsProvider

Derived from IJYResultsObject

AppendDataObject

Function: HRESULT AppendDataObject([in] IJYDataObject *dataObj, [in, optional]VARIANT dataObjId);
Description: Appends the given data object to the result. This is done by reference (not copy).
Parameters:

dataObj – the data object to be appended
dataObjId – an identifier to be used for the data object if you wanted to retrieve it later by ID.

See Also: GetDataObjectById

GetDataObjectById

Function: HRESULT GetDataObjectById([in]VARIANT dataObjId, [out] IJYDataObject **dataObj,[in,optional] VARIANT cycle);
Description: Retrieves a data object by the identifier used when the object was appended to this result.
Parameters:

For internal use only.

WriteExperimentInfo

Function: HRESULT WriteExperimentInfo([in]BSTR fileName);
Description: For future use.

Initialize

Function: HRESULT Initialize([in, optional]VARIANT numCycles);
Description: Initializes a new result object.
Parameters:

numCycles – Optional Parameter that gives the result an idea of how many cycles will be used when filling this result. Defaults to 1.

Clear

Function: HRESULT Clear();
Description: Clears the result of all encapsulated data objects. Release is called on all internal data objects.

MakeFakeResult

Function: HRESULT MakeFakeResult([in]enum jyDataObjectDataFormat dataFormat, [in]long numDataObjects, [in]long dimsPerDataObject, [in, size_is(dimsPerDataObject)]long *dataObjectDims);
Description: Creates dummy data in the result. For internal use only.

SetProperty

Function: HRESULT SetProperty([in]enum jyResultsProperty whichProperty, [in]VARIANT value);
Description: Sets the specified property value into the result object. This value can be retrieved later via the GetProperty function.
Parameters:
whichProperty – specifies which property to set
{jyRPDescription, jyRPHeader, jyRPTime,jyRPDate, jyRPSaveMode,}
Value – the value to be associated with the given property.
See Also: GetProperty

AppendUpdate

Function: HRESULT AppendUpdate([in]IJYResultsObject *resultInterface);
Description: Dynamic building of results. For internal use only.

CurrentCycle

Function: HRESULT CurrentCycle([in]long cycle);
Description: Current cycle of dynamically built result. For internal use only.

CycleCount

Function: HRESULT CycleCount([out, retval]long *count);
Description: Total number of cycles in the result object. For internal use only.

GlueResult

Function: HRESULT GlueResult([in]IJYResultsObject *resultsObject, [in]enum jyGlueDataMode mode, [in,optional]VARIANT overlap);
Description: Glues one result to another with a specified amount of overlap. For internal use only.

Data Object and Data Object Provider Interfaces

Overview

The IJYDataObject interface provides basic "Read Only" access to the data object. This interface should be used by a data object "Consumer" who is simply querying the data object for information regarding its contents. The IJYDataProvider interface is primarily used internal to the instrument components to produce and fill the data objects for use. Documentation of this interface is provided for completeness, but most SDK developers should be primarily concerned with IJYDataObject interface.

IJYDataObject

Derived from IDispatch

Save

Function: HRESULT Save([in]BSTR fileName);
Description: Saves the given data object.
Parameters:
fileName – name of the file that you want to save the data in. The filename/extension should match the FileType property value (i.e. – fileType = jySPC, filename = c:\myfile.spc)
Example:
Set myResult = MyDoScan()
Set myDataObject = myResult.GetFirstDataObject
myDataObject.FileType = jySPC
myDataObject.Save("C:\MyFile.spc")
See Also: Load

Load

Function: HRESULT Load([in]BSTR fileName);
Description: Loads a data object from the specified file.
Parameters:
filename – name of the file containing results info
Example:
myDataObject.Load("C:\MyFile.spc")
` myDataObject contains the data from myFile.spc
See Also: Save

FileType

Function: HRESULT FileType([out, retval]enum jySupportedFileType *fileType);
HRESULT FileType([in]enum jySupportedFileType fileType);
Description: Indicates the type of file to be Saved when the Save() function is called.
Parameters:

fileType – the type of file to be saved

Example:

```
Set myResult = MyDoScan()
Set myDataObject = myResult.GetFirstDataObject
myDataObject.FileType = jySPC
myDataObject.Save( "C:\MyFile.spc")
```

See Also: Save, Load

Dimensions

Function: HRESULT Dimensions([out, retval]long *dimensions);

Description: Retrieves an array of long values that list the dimensions of the data object. You can determine the size of this array using the NumberOfDimensions property.

Note: It is recommended to use GetNumberOfDimensions + GetDimension to retrieve this information. Passing pointers of unknown size can be difficult depending on the development platform.

Parameters:

dimensions – will contain a pointer to an array of size NumberOfDimensions on return.

See Also: GetDimension, NumberOfDimensions

GetDimension

Function: HRESULT GetDimension([in]long whichDimension, [out, retval]long *dimensionValue);

Description: Retrieves a dimension value of the data object.

Parameters:

whichDimension – 1-based dimension value. Must be > 0 and <= NumberOfDimensions

dimensionValue - the size of the "whichDimension"

Example:

```
'Retrieve the dimensions of this data object
numDims = myDataObject.GetNumberOfDimensions()
for ( I = 1 to numDims )
  GetDimension( I,  dim )
Next I
```

See Also: NumberOfDimensions

GetOffset

Function: HRESULT GetOffset([in]long whichDimension, [out, retval]long *offsetValue);

Description: Offset allows for data objects to be configured relative to a base-coordinate system. For example, a subregion of a CCD produces a dataObject of the specified size, but it is also useful to know how this object relates to the overall CCD. The offset provides this information.

Parameters:

whichDimension – which dimension to retrieve the offset for

offsetValue – the value of the offset

Example:

```
' Want to find out where the first data point in this data object is relative to
the coordinate system it was
' acquired in
myDataObject.GetOffset( 1, offsetVal )
MsgBox ("First Data Point is at " + offsetVal + " in the original coordinate
system" )
```

GetAxis

Function: HRESULT GetAxis([in]long whichDimension, [out,retval]IJYAxis **axis);
Description: Retrieves the axis information for the given data objects requested dimension (See IJYAxis for more details).
Parameters:

whichDimension – which dimension to retrieve the axis for
Axis – interface ptr to contain the IJYAxis interface to the axis object for the requested dimension

Example:

```
' Get the x and y axis of a 2-dimensional data object
set xAxis = myDataObject.GetAxis( 1 )
set yAxis = myDataObject.GetAxis( 2 )
' Get the labels for the axis
xAxisLabel = xAxis.Label
yAxisLabel = yAxis.Label
```

See Also: IJYAxis

GetRawData

Function: HRESULT GetRawData([out, retval]VARIANT *rawData);
Description: Retrieves a SAFEARRAY containing the raw data (one dimensional) associated with the object. It is recommended that the GetDataAsArray functions be used to access the data, as the raw data can be much more difficult to work with and potentially error prone.
Parameters:

rawData – contains a SAFEARRAY that is one dimensional.

Example:

```
Dim rawData as Variant
rawData = myDataObject.GetRawData()
' raw data has not dimensionality.  Interpretation of data left to the caller
```

See Also: GetDataAsArray

DataLayout

Function: HRESULT DataLayout([out, retval] enum jyDataLayout *dataLayout);
Description: NOT IMPLEMENTED

DoOperation

Function: HRESULT DoOperation([in]enum jyOperation operation, [in]VARIANT operand1, [in, optional]VARIANT operand2);
Description: Performs the specified operation, using the provided operands. This function can be used to add, multiply, subtract or divide constants and other data

objects. This data object itself is the primary operand (i.e. – all operations are performed on it).

Parameters:

operation – specifies the operation to be performed
Supported operations = jyAdd, jySubtract, jyMultiply, jyDivide,
Operand1 – the first operand (constant or data object) to be used in the operation
Operand2 – (optional) 2nd operand (constant or data object to be used in the operation

Example:

```
'Add one object to another
set firstDataObject = MyGetDataObject() ' function to do the single scan and return the result
set secondDataObject = MyGetDataObject ()
firstDataObject.DoOperation( jyAdd, secondDataObject )
' firstDataObject is now has secondDataObject added into it.
firstDataObject.DoOperation( jyDivide, 2 )
' firstDataObject is now the average of the original firstDataObject and secondDataObject
```

MakeEvent

Function: HRESULT MakeEven([in, optional]VARIANT makeEvenBasedOn);
Description: NOT IMPLEMENTED

DataFormat

Function: HRESULT DataFormat([out, retval]enum jyDataObjectDataFormat *dataFormat);
Description: Retrieves the data format of the data object.

Parameters:

dataFormat – the format of the data in this object (i.e. – double, long, etc…) {jyDFInt8,jyDFUInt,jyDFInt16,jyDFUInt16,jyDFInt32, jyDFUInt32,jyDFFloat,jyDFDouble,jyDFLongDouble}

Example:

```
'Check the data object to make sure that the data is 32 bit integer type
if ( myDataObject.DataFormat = jyDFInt32 OR myDataObject.DataFormat = jyDFUInt32 )
   ' ok to process as 32 bit data
else
   ' not 32 bit data
endif
```

GetElement

Function: HRESULT GetElement([in]long numDims, [in, size_is(numDims)]long *elementDims, [out,retval]VARIANT *data);
Description: Retrieves a specific element value from within the data object.
Parameters:

numDims - specifies how many dimension are specified in the elementDims parameter

elementDims – array of long values of size [numDims] specifying the coordinates of the requested element

data – the value retrieved from the specified location

Note: Use of size is not well supported on all platforms. In C++ it works fine, but in VB, there may be problems using this construct. Developers on these platforms should extract the entire array of data and then manually retrieve individual elements.

Example:

```
` C++ Example
long elementDims[] = {10, 10}
VARIANT data;
myDataObject->GetElement( 2, elementDims, &data )
```

See Also: GetDataAsArray

DumpToFile

Function: HRESULT DumpToFile([in]BSTR fileName);
Description: For internal use only.

NumberOfDimensions

Function: HRESULT NumberOfDimensions([out,retval]long *numDims);
Description: Returns the number of dimensions in this data object.
Parameters:

numDims – the number of dimensions contained in this data object

Example:

```
`check if data is spectral or image
if ( myDataObject.NumberOfDimensions > 1 ) then
   `image data
else
   ` spectral data
```

Description

Function: HRESULT Description([in]BSTR description);
HRESULT Description([out, retval]BSTR *description);
Description: Description provided by the component that produced the object or optionally set by the developer.
Parameters:

description – text description of the data object

Example:

```
`low level processing of data object
myDataObject.Description = "Detailed Description"
` At a higher level, user can extract that description for display
```

dataDesc.Text = myDataObject.Description

PerformAnalysis

Function: HRESULT PerformAnalysis([in]enum jyAnalysisOperation operation, [out]VARIANT *result);
Description: This function performs some basic statistical analysis of the data.

Note: Analysis is performed only on the first request. Subsequent requests retrieve stored values.

Parameters:
operation – which operation do you want performed
{jyAnalysisStdDev, jyAnalysisAvg, jyAnalysisMin,
jyAnalysisMax, jyAnalysisSum, jyAnalysisSumSqrs}
Result – the value of the requested analysis
Example:
myDataObject.PerformAnalysis(jyAnalysisStdDev, stdValue)
myDataObject.PerformAnalysis(jyAnalysisAvg, avgValue)
myDataObject.PerformAnalysis(jyAnalysisMin, minValue)
myDataObject.PerformAnalysis(jyAnalysisMax, maxValue)
See Also: DoOperation

GetDataAsArray

Function: HRESULT GetDataAsArray([out]VARIANT *dataAsSafeArray,[in,optional]VARIANT copy, [in,optional]VARIANT accumulation) ;
Description: Retrieves the data of this data object in a SAFEARRAY format.
Parameters:
dataAsSafeArray – a variant that will contain a SAFEARRAY on successful return
Copy – this optional parameter defaults to VARIANT_TRUE. By default, we copy the data. Non-copy operations are provided, but have not been thoroughly tested. We recommend that the data be copied to the resulting SAFEARRAY.
Accumulation – specifies from which accumulation you want the data. This is only applicable when the AccumulationCount > 1. The default value of this parameter is 1.
Example:
Dim dataArray as VARIANT
myDataObject.GetDataAsArray dataArray
‘ You now have a SAFEARRAY that you can use as you wish

GetDataBinned

Function: HRESULT GetDataBinned([in]long xBin, [in]long yBin, [out]VARIANT *binnedDataAsSafeArray) ;
Description: Takes a multidimensional data object and “bins” the data in either the x or y direction (but not both at the same time) and produces a safe array of the binned data.
Parameters:
xBin – number of pixels in the x to bin together
yBin – number of pixels in the y to bin together

binnedDataAsSafeArray – a SAFEARRAY of data representing the binned result

Example:

NOT CURRENTLY TESTED. FOR INTERNAL USE ONLY

GetPropertyValue

Function: HRESULT GetPropertyValue([in]enum jyDataItemProperty property, [out, retval]VARIANT *value);

Description: Retrieves the value of the requested property.

Parameters:

property – Specifies the property value you want retrieved {jyDIPDescription, jyDIPSourceName, jyDIPXAxisLabel, jyDIPXAxisUnitsVal, jyDIPXAxisUnitsString, jyDIPYAxisLabel, jyDIPYAxisUnitsVal,jyDIPYAxisUnitsString, jyDIPZAxisLabel, jyDIPZAxisUnitsVal, jyDIPZAxisUnitsString, jyDIPTAxisLabel, jyDIPTAxisUnitsVal,jyDIPTAxisUnitsString,jyDIPDataLabel,jyDIPDataUnitsVal, jyDIPDataUnitsString, jyDIPDataOverrange, jyDIPHidden, jyDIPCosmicEnabled, jyDIPCosmicMethod, jyDIPCosmicParameter1, jyDIPCosmicParameter2, jyDIPFlippedX}

Value – the value of the requested property

Example:

` Get the x axis label for this data object
myDataObject.GetProperty(jyDIPXAxisLabel, xLabel)

See Also: SetPropertyValue

GetCycle

Function: HRESULT GetCycle([out,retval]long *pVal);

Description: For internal use only.

MakeCopy

Function: HRESULT MakeCopy([out, retval] IJYDataObject **dataObjCopy);

Description: Duplicates the current data object into the dataObjCopy parameter.

Parameters:

dataObjCopy – interface to object that will contain a duplicate of this data object

Example:

Set myDataObjectCopy = myDataObject.Copy()

SetElement

Function: HRESULT SetElement([in]long numDims, [in, size_is(numDims)]long *elementDims, [in]VARIANT data);

Description: Sets the value of an element in the data object based on the coordinates provided in the elementDims parameter.

Parameters:

numDims – the number of dimensions provided in the elementDims parameter.

elementDims – array of coordinates "elementDims" in size

Note: Use of size is not well supported on all platforms. In C++ it works fine, but in VB, there may be problems using this construct. Developers on these platforms should extract the entire array of data and then manually retrieve individual elements.

Example:

```
` C++ Example
long elementDims[] = {10, 10}
CComVariant data = (CComVariant)200;
myDataObject->SetElement( 2, elementDims, data )
```

See Also: GetElement

IJYDataProvider

Derived from IJYDataObject

Note: The DataProvider interface allows much more access to modifying (and possibly corrupting) the data. Extreme care should be used when using this interface.

Initialize

Function: HRESULT Initialize([in]enum jyDataObjectDataFormat format, [in]long numDims, [in, size_is(dims)]long *dims, [in, optional]VARIANT accumMode, [in, optional]VARIANT numAccumulations);
Description: Initializes the data object to the appropriate type and dimensions. Should only be called on new data objects, not on existing data objects.
Parameters:

format – the format to be used within this data object (long, int, double, etc…)
numDims – the number of dimensions this data will have
dims – array of long values of size "numDims"
accumMode – For internal use only – default value is 1.

Example:

```
` C++ Example
dims[0] = xSize / xBin;
dims[1] = ySize / yBin;
if ( dims[1] == 1 ) // Scan data
   numDims = 1;
else
   numDims = 2;
// Set the offsets...
offsets[0] = xOrigin;
offsets[1] = yOrigin;
// Construct an area of 32-bit data
hr = tmpDataObject->Initialize( jyDFInt32, numDims, dims );
for( int i = 0;i<numDims;i++ )
{
   hr = tmpDataObject->SetOffset( i+1, offsets[i] );
   ASSERT( SUCCEEDED( hr ));
```

```
}
```

Clear

Function: HRESULT Clear();
Description: Clears a data object of any of its contents.

SetOffset

Function: HRESULT SetOffset([in]long whichDimension, [in]long offsetValue);
Description: Sets the relative position of this data object with a larger virtual data environment.
Parameters:

whichDimension – 1-based dimension to set the offset value for
offsetValue – the value of the offset in the number of datapoints
Example:

```
` C++ Example
dims[0] = xSize / xBin;
dims[1] = ySize / yBin;
if ( dims[1] == 1 ) // Scan data
   numDims = 1;
else
   numDims = 2;
// Set the offsets...
offsets[0] = xOrigin;
offsets[1] = yOrigin;
// Construct an area of 32-bit data
hr = tmpDataObject->Initialize( jyDFInt32, numDims, dims );
for( int i = 0;i<numDims;i++ )
{
   hr = tmpDataObject->SetOffset( i+1, offsets[i] );
   ASSERT( SUCCEEDED( hr ));
}
```

Example:

```
` C++ Example
dims[0] = xSize / xBin;
dims[1] = ySize / yBin;
if ( dims[1] == 1 ) // Scan data
   numDims = 1;
else
   numDims = 2;
// Set the offsets...
offsets[0] = xOrigin;
offsets[1] = yOrigin;
// Construct an area of 32-bit data
hr = tmpDataObject->Initialize( jyDFInt32, numDims, dims );
for( int i = 0;i<numDims;i++ )
{
   hr = tmpDataObject->SetOffset( i+1, offsets[i] );
   ASSERT( SUCCEEDED( hr ));
}
```

SetAxis
Function: HRESULT SetAxis([in]long whichDimension, [in]IJYAxis *axis, [in]BOOL IsDefaultAxis);
Description: For internal use only.

SetRawData
Function: HRESULT SetRawData([in]VARIANT rawData);
Description: Takes a safe array of raw data and copies it into the data object. For internal use only.

MakeFakeData
Function: HRESULT MakeFakeData([in, optional]VARIANT functionType);
Description: Fills the data object with fake data. For internal use only.

AppendDataPoint
Function: HRESULT AppendDataPoint([in]VARIANT dataPoint);
Description: Used for dynamic data object creation. For internal use only.

CurrentDataPointIndex
Function: HRESULT CurrentDataPointIndex([in]long dataPointIndex);
HRESULT CurrentDataPointIndex([out,retval]long *dataPointIndex);
Description: Set/Get the current data point index. Used in dynamic data object creation. For internal use only.

GetDataPtr
Function: HRESULT GetDataPtr([out]VARIANT *dataPtr);
Description: Retrieves the raw data pointer for the given data object. The pointer is contained in a Variant with type VT_BYREF | <data type> . The caller must understand how to access the raw data based on the type of data contained. For internal use only.

SignalIndex
Function: HRESULT SignalIndex([in]long signalIndex);
HRESULT SignalIndex([out,retval]long *signalIndex);
Description: For internal use only.

AppendDataObject
Function: HRESULT AppendDataObject([in]IJYDataObject *dataObject);
Description: Appends a data object to this data object. Used for dynamic data construction. For internal use only.

AppendMode
Function: HRESULT AppendMode([in]enum jyAppendDataMode mode);
HRESULT AppendMode([out, retval]enum jyAppendDataMode *mode);

Description: Controls how dynamic data is appended to this data object. For internal use only.

GetCurrentIndexByDimension

Function: HRESULT GetCurrentIndexByDimension([out]long *index, [in]long dim);
Description: Accesses the dynamic index per dimension for dynamic data construction. For internal use only.

SetPropertyValue

Function: HRESULT SetPropertyValue([in]enum jyDataItemProperty property, [in]VARIANT value);
Description: Sets the specified property to the provided value.
Parameters:

property – specifies which property is to be set
Value – the value that the property is set to

See Also: GetPropertyValue

CommitOperation

Function: HRESULT CommitOperation();
Description: For dynamic building of data object, tells the data object it can commit all operations. For Internal Use Only.

AccumulationIndex

Function: HRESULT AccumulationIndex([out, retval]long *pVal);
Description: The current accumulation index. Used for dynamic data object construction. For Internal Use Only.

Cycle

Function: HRESULT Cycle([in]long pVal);
Description: Sets the current cycle to the data object. For Internal Use Only.

AccumulationCount

Function: HRESULT AccumulationCount([out, retval]long *count);
Description: Queries the total accumulation count of the data object.
Parameters:

count – how many accumulations are represented in this data object

GetFirstProperty/GetNextProperty

Function: HRESULT GetFirstProperty([out]enum jyDataItemProperty *property, [out, retval]VARIANT *value);
HRESULT GetNextProperty([out]enum jyDataItemProperty *property, [out, retval]VARIANT *value);
Description: Enumerates the properties and property values associated with the data object.

Parameters:

Property- the property token

Value – the value associated with the property token

See Also: Enumerations

MakeMirror

Function: HRESULT MakeMirror([out, retval] IJYDataObject **dataObjCopy);
Description: Constructs a data object with the same properties and dimensions as this data object. For Internal Use Only.
Parameters:

dataObjCopy – interface to object that will contain a duplicate of the data object minus the data

GlueDataObject

Function: HRESULT GlueDataObject([in]IJYDataObject *dataObject, [in]enum jyGlueDataMode mode, [in,optional]VARIANT overlap);
Description: Glues a data object to this data object - dynamic building of data object. For internal use only.

FlipX

Function: HRESULT FlipX();
Description: Reverses the X axis and data of this data object (ie- data object from 1 to n changed to from n to 1).
Parameters: none

GetCurrentAccumulationIndex

Function: HRESULT GetCurrentAccumulationIndex([out,retval]long *currentIndex);
Description: For dynamic data construction. For internal use only.

IJYAxis

(Derived From IDispatch)

Overview

The data object contains axis information per dimension. By default, the axis simply represents the dimensions of the data object. This axis information can be more robust to handle experimental axis or linearized axis. The axis object gives access to the properties and information specifically in the context of a data axis.

SetLimits

Function: HRESULT SetLimits([in]long min, [in]long max);
HRESULT GetLimits([out]long *min, [out]long *max);
Description: Used for construction of an axis object. For internal use only.

SetValue/GetValue

Function: HRESULT SetValue([in]long index, [in]VARIANT value);
HRESULT GetValue([in]long index, [out, retval]VARIANT *value);
Description: Puts a value into a specific position in an axis object.

SetValuesBy Array/GetValuesByArray

Function: HRESULT SetValuesByArray([in]VARIANT value);
HRESULT GetValuesByArray([out]VARIANT *value);
Description: Takes an array of values and sets them into the axis object.
Example:

```
'C++
CComVariant values;
axis->GetValuesByArray( &values );  // Retrieves copy of the current values
array
for ( long i=0;i < values.parray->rgsabound[0].cElements; i++ )
{
  'Process/Modify the array that was passed back in some way….
}
// Reset the values of the axis to the new values
axis->SetValuesByArray( values );
```

GetIndexNearestValue

Function: HRESULT GetIndexNearestValue([in]VARIANT value, [out, retval]long *index);
Description: NOT IMPLEMENTED

Label

Function: RESULT Label([in]BSTR label);
HRESULT Label([out, retval]BSTR *label);
Description: Retrieves the label for this axis object.

Units

Function: HRESULT Units([in]enum jyUnits type);
HRESULT Units([out, retval]enum jyUnits *type);
Description: Returns the unit of this axis as an enumeration value. The label should correspond to the string representation of the units.

DumpToFile

Function: HRESULT DumpToFile([in]BSTR fileName);
Description: For internal use only.

Advanced Topics

Advanced Topics: Triggering

Overview

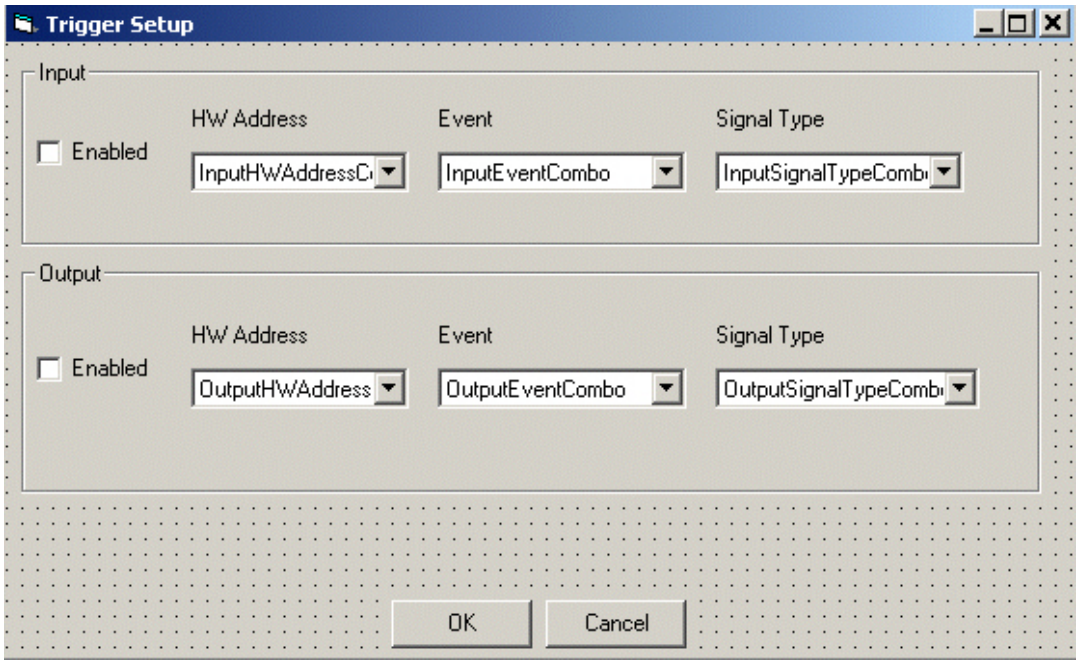
Triggers are handled by various devices based on the specifics of the controller to which they are attached. To facilitate generic handling of triggers, each device is allowed to enumerate the supported capabilities. The format of triggers is:

> DEVICE: ADDRESS: EVENT: SIGNALTYPE. For example, Device = CCD3000, Address = OutputTrigger1, Event = OnStartEachAcq, SignalType = TTL High.

Think of triggers as a three-tier hierarchy with the TriggerAddress as the root. For each TriggerAddress there can be one or more TriggerEvents. For each TriggerEvent, there can be one or more TriggerSignalTypes. The GetFirst/GetNext enumerations give you a way to navigate this tree of trigger capabilities.

Example:

Below is a form that allows a user to configure the input and output triggers of a specific device. The flow of this dialog is that as the user selects an address, the events for that address are enumerated. As an event is selected, the signal types available for that event are enumerated. The code associated with this is included below. To enable a trigger, simply select the token associated with each selected combo box and call the corresponding EnableInput/OutputTrigger functions.




```
Private Sub Form_Load()
```

```
    ReadInputTriggers
    ReadOutputTriggers
End Sub

Private Sub ReadOutputTriggers()
Dim trigString As String
Dim trigToken As Long
Dim trigEventToken As jyTriggerEvent
Dim trigEventString As String
Dim trigSignalToken As jyTriggerSignalType
Dim trigSignalString As String

        DataAcq.ccd.GetFirstSupportedOutputTriggerAddress trigToken,
        trigString
        While (trigToken > jyTriggerTypeUndefined)
                OutputHWAddressCombo.AddItem trigString
                OutputHWAddressCombo.ItemData(OutputHWAddressCombo.NewI
                ndex) = trigToken
                DataAcq.ccd.GetNextSupportedOutputTriggerAddress trigToken,
                trigString
        Wend

End Sub

Private Sub ReadInputTriggers()

Dim trigString As String
Dim trigToken As Long
Dim trigEventToken As jyTriggerEvent
Dim trigEventString As String
Dim trigSignalToken As jyTriggerSignalType
Dim trigSignalString As String

    DataAcq.ccd.GetFirstSupportedInputTriggerAddress trigToken, trigString
    While (trigToken > jyTriggerTypeUndefined)
        InputHWAddressCombo.AddItem trigString
        InputHWAddressCombo.ItemData(InputHWAddressCombo.NewIndex) =
trigToken
        DataAcq.ccd.GetNextSupportedInputTriggerAddress trigToken, trigString
    Wend

End Sub

Private Sub InputEventCombo_Click()
    Dim mySignal As jyTriggerSignalType
    Dim mySignalString As String
    ' Get the item data and update the Events
    InputSignalTypeCombo.Enabled = True
    InputSignalTypeCombo.Clear
    Call
DataAcq.ccd.GetFirstSupportedInputTriggerSignalType(InputHWAddressCombo
.ItemData(InputHWAddressCombo.ListIndex),
InputEventCombo.ItemData(InputEventCombo.ListIndex), mySignal,
mySignalString)
    While (mySignal > jyTrigSigTypeUndefined)
```

```
      InputSignalTypeCombo.AddItem mySignalString
      InputSignalTypeCombo.ItemData(InputSignalTypeCombo.NewIndex) =
mySignal
      Call
DataAcq.ccd.GetNextSupportedInputTriggerSignalType(InputHWAddressCombo
.ItemData(InputHWAddressCombo.ListIndex),
InputEventCombo.ItemData(InputEventCombo.ListIndex), mySignal,
mySignalString)
   Wend
End Sub

Private Sub OutputEventCombo_Click()
   Dim mySignal As jyTriggerSignalType
   Dim mySignalString As String
   ' Get the item data and update the Events
   OutputSignalTypeCombo.Enabled = True
   OutputSignalTypeCombo.Clear
   Call
DataAcq.ccd.GetFirstSupportedOutputTriggerSignalType(OutputHWAddressCo
mbo.ItemData(OutputHWAddressCombo.ListIndex),
OutputEventCombo.ItemData(OutputEventCombo.ListIndex), mySignal,
mySignalString)
   While (mySignal > jyTrigSigTypeUndefined)
      OutputSignalTypeCombo.AddItem mySignalString
      OutputSignalTypeCombo.ItemData(OutputSignalTypeCombo.NewIndex)
= mySignal
      Call
DataAcq.ccd.GetNextSupportedOutputTriggerSignalType(OutputHWAddressCo
mbo.ItemData(OutputHWAddressCombo.ListIndex),
OutputEventCombo.ItemData(OutputEventCombo.ListIndex), mySignal,
mySignalString)
   Wend
End Sub

Private Sub InputHWAddressCombo_Click()
   Dim myEvent As jyTriggerEvent
   Dim myEventString As String
   ' Get the item data and update the Events
   InputEventCombo.Enabled = True
   InputEventCombo.Clear
   Call
DataAcq.ccd.GetFirstSupportedInputTriggerEvent(InputHWAddressCombo.Item
Data(InputHWAddressCombo.ListIndex), myEvent, myEventString)
   While (myEvent > jyTrigEventUndefined)
      InputEventCombo.AddItem myEventString
      InputEventCombo.ItemData(InputEventCombo.NewIndex) = myEvent
      Call
DataAcq.ccd.GetNextSupportedInputTriggerEvent(InputHWAddressCombo.Item
Data(InputHWAddressCombo.ListIndex), myEvent, myEventString)
   Wend
End Sub

Private Sub OutputHWAddressCombo_Click()
   Dim myEvent As jyTriggerEvent
```

```
    Dim myEventString As String
    ' Get the item data and update the Events
    OutputEventCombo.Enabled = True
    OutputEventCombo.Clear
    Call
DataAcq.ccd.GetFirstSupportedOutputTriggerEvent(OutputHWAddressCombo.It
emData(OutputHWAddressCombo.ListIndex), myEvent, myEventString)
    While (myEvent > jyTrigEventUndefined)
        OutputEventCombo.AddItem myEventString
        OutputEventCombo.ItemData(OutputEventCombo.NewIndex) = myEvent
        Call
DataAcq.ccd.GetNextSupportedOutputTriggerEvent(OutputHWAddressCombo.It
emData(OutputHWAddressCombo.ListIndex), myEvent, myEventString)
    Wend
End Sub

Private Sub InputSignalTypeCombo_Click()
    InputTrigEnabled.Enabled = True
End Sub

Private Sub OkBtn_Click()
    Visible = False
End Sub

Public Sub GetEnabledInputTrigger(trigType As jyTriggerType, trigEvent As
jyTriggerEvent, sigType As jyTriggerSignalType)
    trigType =
InputHWAddressCombo.ItemData(InputHWAddressCombo.ListIndex)
    trigEvent = InputEventCombo.ItemData(InputEventCombo.ListIndex)
    sigType =
InputSignalTypeCombo.ItemData(InputSignalTypeCombo.ListIndex)
End Sub

Public Sub GetEnabledOutputTrigger(trigType As jyTriggerType, trigEvent As
jyTriggerEvent, sigType As jyTriggerSignalType)
    trigType =
OutputHWAddressCombo.ItemData(OutputHWAddressCombo.ListIndex)
    trigEvent = OutputEventCombo.ItemData(OutputEventCombo.ListIndex)
    sigType =
OutputSignalTypeCombo.ItemData(OutputSignalTypeCombo.ListIndex)
End Sub
```

Advanced Topics: Multiple Accumulations/Hardware Time-based Acquistion

Overview

Multiple Accumulations (Blast Mode) is a mode of operation available in some controllers that allows the controller to stream multiple acquisitions with hardware based timing (delays, etc.) removing the inaccurate and unreliable timing of the Windows-Based host computer. Not all controllers support this mode of operation. To determine if the controller can do multiple accumulations, you can use the IsOperatingModeSupported command to check for support of the jyDevOpModeAcqHWTimeBased mode of operation. If this mode of operation is supported, you can then use the Get/SetOperatingModeValue functions to determine if the mode is enabled and/or enable/disable it.

Although the data will be acquired by the hardware on the first acquisition, the caller still needs to request the data as if acquiring multiple independent acquisitions. This allows the calling application to behave the same way whether the controller is performing the acquisitions or the host is handling multiple acquisitions. This means that the caller must always loop through 'n' acquisition starts and reads (either synchronous or asynchronously) regardless of the mode of operation.

Example

This example demonstrates how to acquire 5 accumulations in 'Blast' mode (the timing is handled entirely by the controller).

```
If (ccd.IsOperatingModeSupported(jyDevOpModeAcqHWTimeBased)) Then
        ccd.SetOperatingModeValue(jyDevOpModeAcqHWTimeBased, True)
        ccd.SetMultiAcqCleanCount jyMultiAcqStateBeforeFirstOnly, CleanCountBefore
        ccd.SetMultiAcqDelay jyMultiAcqStateBeforeFirstOnly, delayBefore
        ccd.MultiAcqShutterMode = jyMultiAcqStateBeforeFirstOnly
        ccd.AcquisitionCount = 5
Endif

` Following code will work whether or not Multiple HW Based accumulations are
supported
numAcqs = 0
While ( numAcqs < ccd.AcquisitionCount )
        ccd.StartAcquisition
        While ( ccd.AcquisitionBusy )
        Wend
        myResult = ccd.GetResult
        MyProcessResultFunction( myResult )
Wend
```

Filename: SynerJY_SDK
Directory: W:\Andrea Witter\My Documents\RoboHelp Office\RoboHTML\SDK\!SSL!\Printed_Documentation
Template: C:\Documents and Settings\AWitter.WINNT\Application Data\Microsoft\Templates\Normal.dot
Title: SynerJY SDK
Subject:
Author: andrea witter
Keywords:
Comments:
Creation Date: 12/21/2004 2:11:00 PM
Change Number: 48
Last Saved On: 12/22/2004 6:07:00 PM
Last Saved By: andrea witter
Total Editing Time: 680 Minutes
Last Printed On: 12/22/2004 6:10:00 PM
As of Last Complete Printing
Number of Pages: 144
Number of Words: 25,424 (approx.)
Number of Characters: 169,837 (approx.)