HSFW dll Documentation

Table of contents

Introduction

Introduction

This document describes the public members of Optec's High Speed Filter Wheel API. The name of the dll is "OptecHID_FilterWheelAPI.dll". The HID portion stands for Human Interface device. When an Optec High Speed Filter Wheel is attached to the computer using a standard USB cable the Windows operating system will recognize the device as a Compatible Human Interface Device. All versions of Windows include built-in drivers for communicating with such a device. However, the operating system does not know what a filter wheel is and how to tell it to perform basic options. This is the job of the API.

The dll is COM capible but needs to be registered if you want to use COM. To regester the dll use the Windows Commandline tool regasm and pass the dll name as the argument. Once this is finished the dll is registered for COM in the system registry. This allows and COM based language to make calls to the API accessing its public methods and properties. These methods and properties are described in this document.

Before we get into the specifics of how to use the dll you need to understand its structure. There are two main classes in the dll, *FilterWheel* and *FilterWheels*. The *FilterWheels* class is necessary for cases when multiple High Speed Filter Wheels are attached to the PC at one time. This class allows you to obtain a list of the attached devices and then create a reference to a specific device. This reference it so a *FilterWheel* object which is the second class in the dll. An instance of the *FilterWheel* class represents a physical High Speed Filter Wheel which has been connected to the PC. Once you have obtaibed a reference to a *FilterWheel* object you can easially control the device and obtain information regarding the current status of the device.

FilterWheels Class

Properties

FilterWheelList

FilterWheelList

The *FilterWheelList* property allows you to access the list of all the *FilterWheel* Objects that have been attached since the instance of the class was created.
This property returns the *FilterWheel* objects in an ArrayList. You must cast the items in the ArrayList as *FilterWheel* objects before you can use them.

Note: Every object in the *FilterWheelList* is not necessarily connected. Check the *IsAttached* property of the *FilterWheel* Object to see if it is currently connected.

Example:

```
FilterWheels myFilterWheelManager = new FilterWheels();
foreach (FilterWheel fw in myFilterWheelManager.FilterWheelList)
{
	Console.WriteLine("Found a Filter Wheel! Serial Number = " + fw.SerialNumber);
}
```

AttachedDeviceCount

AttachedDeviceCount

The *AttachedDeviceCount* property provides quick and easy access to the number of filter wheel devices that are currently attached to the system.
The *AttachedDeviceCount* is not simply the number of elements in the *FilterWheelList* because the *FilterWheelList* contains an object representing every filter wheel device that has been connected to the PC since the initial creation of the *FilterWheels* class instance. To obtain a count of the attached filter wheels manually you would have to iterate through the *FilterWheelList* and count each device that is attached by checking the *IsAttached* property.

Example:

```
FilterWheels myFilterWheelManager = new FilterWheels();
if(myFilterWheelManager.AttachedDeviceCount == 0)
{
	MessageBox.Show("There are no filter wheels currently attached.");
}
```

API_Diagnostics

API_Diagnostics

The API_Diagnostics property provides a means of determining the version numbers associated with the various assemblies that are used by the API as well as the current LoggingLevel of the EventLogger.

Example:

C#
```
FilterWheels myFilterWheelManager = new FilterWheels();
MessageBox.Show(myFilterWheelManager.API_Diagnostics);
```

JScript:
```
var FWManager = new ActiveXObject("OptecHID_FilterWheelAPI.FilterWheels");
var s = FWManager.API_Diagnostics;
WScript.Echo(s);
```

Events

FilterWheelRemoved

public event FilterWheelRemoved

This event is triggered anytime an HSFW is removed from the USB bus.

FilterWheelAttached

public event FilterWheelAttached

This event is triggered anytime an HSFW is attached to the USB bus.

FilterWheel

Properties

Centering Offset

CenteringOffset (read/write)

The centering offset is used to adjust the final resting position of the wheel after each move or home command.

Adjust the centering positive or negative to cause the wheel to travel further clockwise or counterclockwise after each move.

Default Value	0
Min Value	-128
Max Value	+127

Example:
FilterWheel1.CenteringOffset = -5;

CurrentPosition

CurrentPosition (read/write)
Property Type:

The CurrentPosition property is used to get or set the current filter position for the device.

Example:
// Move to filter number 3
FilterWheel1.CurrentPosition = 3;
// Read the current position
MessageBox.Show("The current position is " + FilterWheel1.CurrentPosition.ToString());

ErrorState

ErrorState (read-only)
Property Type: Integer

The *ErrorState* property is used to get the current error state of the device.

If during operation, the firmware detects that an error condition has occurred the error state is set and no other commands can be processed until the error state has been cleared using the *ClearErrorState* method.

Use the *GetErrorMessage* method to get a string description of the error.

ErrorState	Description
0	No error has occurred. (cleared state)
1	The 12VDC power has been disconnected from the device.
2	The device stalled during a Home of Move procedure.
3	An invalid parameter was received during communication with the host(PC).
4	An attempt to Home the device was made while the device was already in motion.

5	An attempt to change filter positions was made while the device was already in motion.
6	An attempt to change filter positions was made before the device had been homed.
7	Optec use only
8	Unknown critical error

FirmwareVersion

FirmwareVersion (read-only)
Property Type: String

The FirmwareVersion property returns the version of the firmware programmed into the device.

Example:
MessageBox.Show(FilterWheel1.FirmwareVersion)

Output:
V1.0.0

IsAttached

IsAttached (read-only)
Property Type: Boolean

The IsAttached property can be used to check whether a FilterWheel device is physically connected to the PC.

Example:
```
if(myFilterWheel.IsAttached) myFilterWheel.HomeDevice;
else MessageBox.Show("The Filter Wheel Was Removed!");
```

IsHomed

IsHomed (read-only)
Property Type: Boolean

The IsHomed property can be used to check if the Filter Wheel has been homed. The Filter Wheel Device automatically home on power up so the device should always be "homed" however, if the device stalls for some reason, the IsHomed property would be set to False and an Error State would be set in the device firmware.

Example (C#):
```
if(!myFilterWheel.IsHomed) MessageBox.Show("The filter wheel device has stalled!");
```

IsHoming

IsHoming (read-only)
Property Type: Boolean

The IsHoming property should be used to check if the filter wheel device is homing at the moment. You can check this property continuously while the device is homing and use it as an indicator of when the homing process is complete

Example:
```
myFilterWheel.HomeDevice();
while (myFilterWheel.IsHoming)
{
        System.Threading.Thread.Sleep(100);
        textbox1.Text = "Device is Homing";
}
// The homing procedure has finished
```

textbox1.Text = "Device is Homed!"

IsMoving

IsMoving (read-only)
Property Type: Boolean

The IsMoving property can be used to check if the device is in motion at any instance. You can use this property to tell when a move is complete.

Example:
```
myFilterWheel.CurrentPosition = 3;
Textbox1.Text = "Moving to Position 3";
while (myFilterWheel.IsMoving)
{
        // wait for move to complete...
        System.Threading.Thread.Sleep(50);
}
Textbox1.Text = "Move Complete. " + myFilterWheel.CurrentPosition.ToString();
```

Manufacturer

Manufacturer (read-only)
Property Type: String

The Manufacturer property returns a string containing the manufacturer of the device as specified in the device firmware.

Example:

```
Console.WriteLine(myFilterWheel.Manufacturer);
```

****
Output:

"Optec Inc."
****

Name

Name (read-only)
Property Type: String

Example:

NumberOfFilters

NumberOfFilters (read-only)
Property Type: Short

The NumberOfFilters property returns the number of filter positions there are in the wheel that is currently installed in the filter wheel device.

Example:

```
Console.WriteLine(myFilterWheel.NumberOfFilters.ToString());

**********
```

Output:

```
5
**********
```

SerialNumber

SerialNumber (read-only)

Property Type: String

The SerialNumber property returns the Serial Number of the filter wheel device. Each filter wheel device is programmed with a unique serial number at the time of production.

Example:

```
Console.WriteLine(myFilterWheel.SerialNumber);

**********
```

Output:

```
"12345"
**********
```

Methods

ClearErrorState

ClearErrorState ()

Return Type: String

If the device firmware detects that an error has occurred it sets an Error State. When this occurs the device ignores all commands until the Error State is cleared. The intent of this is to make sure that the user/ developer realizes that the error has occurred. If you detect that an Error State has been set you can use the ClearErrorState to clear it. Be sure to home the device any time you clear the error state.

Example:

```
if(myFilterWheel.ErrorState != 0)
{
        Console.WriteLine("An error occurred in the Filter Wheel Device.");
        Console.WriteLine("The error description is: " + myFilterWheel.GetErrorMessage());
        Console.WriteLine("Clearing Error State...");
        myFilterWheel.ClearErrorState();
        Console.WriteLine("Error State Cleared!");
        Console.WriteLine("Homing Device...");
        myFilerWheel.HomeDevice();
        Console.WriteLine("Device is homed and ready for use!");
}
```

GetErrorMessage

GetErrorMessage()
Return Type: String

If an error state has been set in the device you can call the GetErrorMessage method to get a description of what caused the error message.

Example:

```
MessageBox.Show(myFilterWheel.GetErrorMessage());
```

GetFilterNames

GetFilterNames()
Return Type: String[]

Call the *GetFilterNames()* method to get a string array containing the names of all of the filters in the current filter wheel that are stored in the devices non-volatile memory.

Example:

```
Console.WriteLine("The current filter wheel contains the following filters...");
foreach (string name in myFilterWheel.GetFilterNames())
{
        Console.WriteLine(name);
}
```

// Output... Assuming wheel B is inserted in the device and the default names have not been changed.

```
**********************************************************************************************************************
***
 The current filter wheel contains the following filters...
 Filt - B1
 Filt - B2
 Filt - B3
 Filt - B4
 Filt - B5
**********************************************************************************************************************
***
```

GetWheelNames

GetWheelNames()
Return Type: String[]

Call the *GetWheelNames()* method to retrieve a string array containing all of the filter wheel names that have been stored in the devices non-volatile memory.

Example:

```
Console.WriteLine("Filter Wheel Names: ");
foreach (string name in myFilterWheel.GetFilterNames())
{
        Console.WriteLine(name);
}
```

// Output... Assuming wheel B is inserted in the device and the default names have not been changed.

****
***
Filter Wheel Names:
Wheel-A
Wheel-B
Wheel-C
Wheel-D
Wheel-E
****

HomeDevice

HomeDevice ()
Return Type: Void

The HomeDevice() method causes the device to perform the homing procedure. The wheel will spin until it reaches the Filter #1 position at which point it will stop and center.
The HomeDevice() is a blocking method. It will not return until the home procedure completes.

Example:

```
myFilterWheel.HomeDevice();
MessageBox.Show("Homing Complete");
```

HomeDevice_ASync

HomeDevice_ASync ()
Return Type: Void

The HomeDevice() method causes the device to perform the homing procedure. The wheel will spin until it reaches the Filter #1 position at which point it will stop and center.
The HomeDevice_ASync() is a non-blocking method. It returns immediately after the homing process begins.

This method will automatically return if the device is already homing or moving at the time the method is called.

Use the *FilterWheel.IsHoming* property to check when the homing procedure is finished.

RestoreDefaultNames

RestoreDefaultNames ()
Return Type: Void

The RestoreDefaultNames() method will set all of the filter names and wheel names back to their default values.

Example:

```
myFilterWheel.RestoreDefaultNames();
```

UpdateFilterName

UpdateFilterName (`char WheelID, short FilterNumber, string NewName`)
Return Type: Void

The UpdateFilterName method is used to update the name of a single filter in the devices non-volatile

memory.
This method takes three parameters:
WheelID must be of type char and must represent a valid Wheel ID (A through E).
FilterNumber must be of type short and must represent a valid Filter number. (Either 1 through 5 or 1 through 8 depending on the wheel).
NewName must be a string with a max length of 8 characters. The New Name parameter represents the new name for the filter that will be stored.

Example:
To store the name "Green" for filter # 2 with Wheel 'C'...

```
myFilterWheel.UpdateFilterName('C', (short)2, "Green");
```

UpdateWheelName

UpdateWheelName (`char WheelID, string NewName`)
Return Type: Void

The UpdateFilterName method is used to update the name of a single wheel in the devices non-volatile memory.
This method takes two parameters:
WheelID must be of type char and must represent a valid Wheel ID (A through H).
NewName must be a string with a max length of 8 characters. The New Name parameter represents the new name for the filter that will be stored.

Example:
To store the name "RGB" for wheel 'D'...

```
myFilterWheel.UpdateFilterName('D', "RGB");
```

Events

DeviceRemoved

Event: DeviceRemoved

This event is triggered when a FilterWheel is physically unplugged from the system.

HomingStarted

Event: HomingStarted

This event is triggered anytime a homing procedure is initiated.

HomingComplete

Event: HomingComplete

This event is triggered when a homing process completes.

MoveStarted

Event: MoveStarted

This event is triggered right right when the device begins changing filter positions.

MoveComplete

Event: MoveComplete

This event is triggered when the device has finished changing filter positions.

Application Examples

Application Examples

The documents in this section demonstrate how to operate the Optec High Speed Filter wheel from various scripting languages by making calls to the OptecHID_FilterWheelAPI via COM.

These example files are also installed into the download directory/Example Files folder during the install process. The folder also includes an example for C# and C++.

Excel

Microsoft Excel

To access the VBA editor in Microsoft Excel open a new spreadsheet and press Alt+F11. Create a new module and add the following code. You will also have to add some buttons to make calls the appropriate subroutines. See the example Excel file in the download directory.
For more information please contact Optec Technical Support directly.

```
Sub ExcelExample()
Set FWs = CreateObject("OptecHID_FilterWheelAPI.FilterWheels")
Set FWArrayList = FWs.FilterWheelList
Set myFilterWheel = FWArrayList(0)

'Clear the test output data
Range("B5:D20").Value = ""
DoEvents

'Print the serial numbers of detected devices
Range("DetectedDevices").Select
For Each FilterWheel In FWArrayList
ActiveCell.Value = FilterWheel.SerialNumber
ActiveCell.Offset(1, 0).Select
Next
DoEvents

'Test the first Filter Wheel in the list
Range("OutputData").Select
ActiveCell.Value = "Homing Device at index 0..."
ActiveCell.Offset(1, 0).Select
DoEvents
myFilterWheel.HomeDevice

ActiveCell.Value = "Moving to Position 3"
ActiveCell.Offset(1, 0).Select
DoEvents
myFilterWheel.CurrentPosition = 3

ActiveCell.Value = "Moving to Position 5"
ActiveCell.Offset(1, 0).Select
DoEvents
myFilterWheel.CurrentPosition = 5

ActiveCell.Value = "Reading Number of Filters..."
ActiveCell.Offset(1, 0).Select
ActiveCell.Value = "Number of Filters = " & myFilterWheel.NumberOfFilters
ActiveCell.Offset(1, 0).Select
DoEvents
```

```
ActiveCell.Value = "Reading WheelID..."
ActiveCell.Offset(1, 0).Select
ActiveCell.Value = "Current WheelID = " & Chr(myFilterWheel.WheelID)
ActiveCell.Offset(1, 0).Select

End Sub
```

LabView

National Instruments - LabView Example

Before I get into explaining this example I should warn you that I am by no means a LabView expert. In fact, making this example has been my first experience ever using LabView. That being said please bare with me if I am using the wrong terminology and techniques.

Optec's USB_FilterWheel_DLL is written in C# and is a .NET assembly. Fortunately, LabView has built in tools for accessing .NET objects quite easily. The Block Diagram and Front Panel can be used to control an Optec High Speed Filter Wheel device. The dial indicator labeled Attached Filter Wheels shows the number of filter wheel devices that are currently connected to the PC. It does so by displaying the count of the items in the FilterWheelList ArrayList in the FilterWheels class.

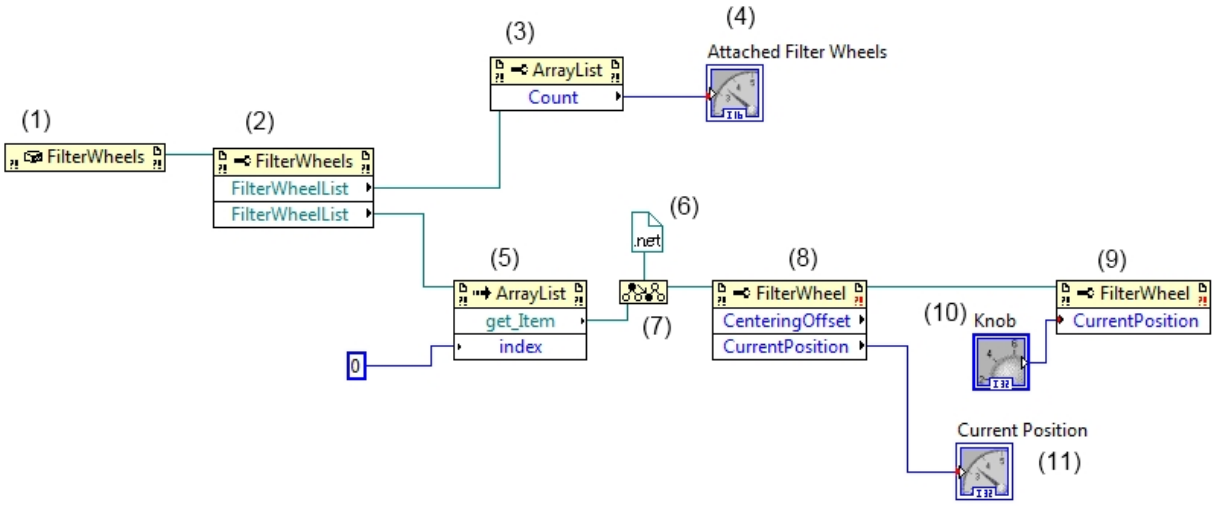
The Block Diagram begins with a *Constructor Node* - (1) - from the .NET Palette . This *Constructor Node* creates an instance of the FilterWheels class which is used to manage all the filter wheels attached to the computer at any given time. Node (2) is a *Property Node* and it is used to access properties of a .NET class. In our case the *Property Node* is accessing the FilterWheelList property of the FilterWheels object. Accessing the FilterWheelList property returns an ArrayList containing all of the attached Filter Wheels. Node (3) is another *Property Node* which accesses the Count property of the ArrayList (FilterWheelList). The output from this node is displayed on the Attached Filter Wheels dial indicator so that the end result is an indicator which displays the total number of filter wheels attached to the computer.

The bottom branch of the Block Diagram starts with an *Invoke Node* (5), also from the .NET Palette, which is used to make calls to methods of .NET objects. The next step is where we actually create an individual *FilterWheel* object so that we can control a device. If we make a call to the *get_item()* method of the ArrayList we can obtain an object that represents a single filter wheel. The *index* input parameter is used to specify which index of the ArrayList we want to access. In my case, there is only one filter wheel hooked up to the computer so I just want to specify index 0. Now, at this point we have an object that represents a filter wheel but it is still just a generic object and LabView doesn't know exactly what type it is because ArrayLists can contain any type of object you can imagine. Node (7) is a tool called *To A More Specific Class,* which is found in the .NET palette. It is used to cast an object from one type to another. The top parameter of this tool is the *Target* class and it is used to specify which type of class will come out the output. Right-click at the the top of the tool click *Create -> Constant* and a little .NET constant object(6) will appear. Right-click on that and choose *Select .NET Class* and on the resulting list you should see the *FilterWheel* class; choose it. Now we finally have a FilterWheel object and we can access all of the public methods and properties of it very simply by using the Invoke and Property nodes as shown in nodes 8 through 10.

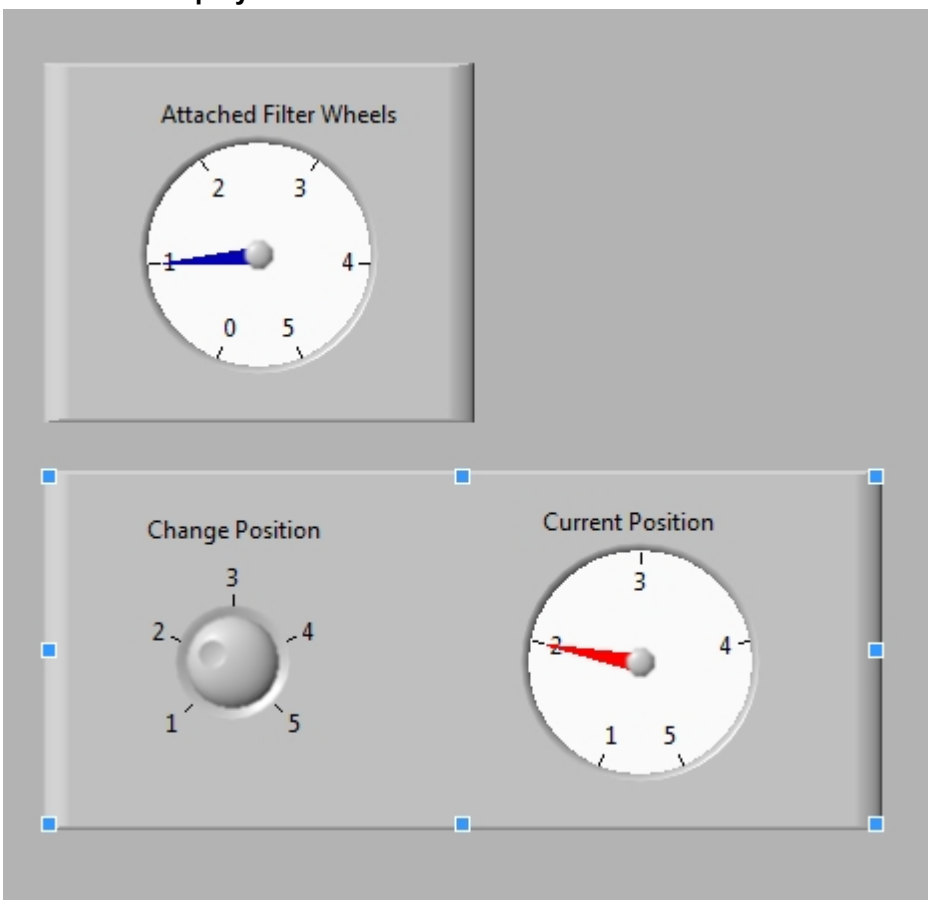
The Front Panel Display below is an example of a quick user interface that I whipped up for this example. It's quite simple, when you turn the knob the FilterWheel changes to the new position and the dial indicator displays the current position from the device.

If you have any questions you can try shooting us an email and I will do what I can to help out but your best bet is probably to head straight to the web and find a LabView forum.

Block Diagram:



Front Panel Display:



JavaScript

JavaScript

The javascript engine is built into windows and is very simple to use. Create a new text file somewhere on your harddrive and change the extension to (.js). Then, paste in the following code. To run the script just double click the .js file. For more information please contact Optec Technical Support direct.

```
try{
//Create a FilterWheels Object to provide a means of accessing attached devices
var FWManager = new ActiveXObject("OptecHID_FilterWheelAPI.FilterWheels");
```

```
// Get the list of attached Devices (an ArrayList of FilterWheel objects)
var ListOfDevices = FWManager.FilterWheelList;
// Output the number of attached devices
WScript.Echo("Found " + ListOfDevices.Count + " High Speed Filter Wheel(s)");
// Create a reference to the first device in the list
var FW = ListOfDevices(0);
// Home the device
FW.HomeDevice;
// Move to various positions
FW.CurrentPosition = 4;
FW.CurrentPosition = 1;
FW.CurrentPosition = 2;
FW.CurrentPosition = 3;
FW.CurrentPosition = 5;
// Create a string do hold device info.
var msg = "";
msg += "Filter Wheel with Serial Number " + FW.SerialNumber + " has " +
FW.NumberOfFilters + " filters in it.\n";
msg += "The inserted Wheels ID is " + String.fromCharCode(FW.WheelID) + "\n";
msg += "The devices firmware version is " + FW.FirmwareVersion + "\n";
msg += "Test Complete!";
WScript.Echo(msg);

}
catch (e) {
    WScript.Echo(e.description);
}
```

VBScript

VBScript

VBScript or Visual Basic Script is also very simple to use from the Windows operating system. Simple create a new file with a .vbs extension and paste in the following code. Double click the file to run the script.

```
'Create a variable to hold the test output data
Dim msg
msg = "Starting Test..." & vbNewLine
'Create an instance of the FilterWheels Class
'this provides you with a means of accessing all of the FilterWheels attached to the PC
Set FiltWheelsObj= WScript.CreateObject ("OptecHID_FilterWheelAPI.FilterWheels" )
'Get an ArrayList of FilterWheel Objects representing the attached devices
Set ListOfDevices=FiltWheelsObj.FilterWheelList
'Create a Reference to the first item in the list.
Set FW = ListOfDevices(0)
'Home the device
FW.HomeDevice
'Change to various positions
FW.CurrentPosition = 4
FW.CurrentPosition = 1
FW.CurrentPosition = 2
FW.CurrentPosition = 3
FW.CurrentPosition = 5
'Get the Number of Filters from the device
msg = msg & "This filter wheel has " & FW.NumberOfFilters & " filters" & vbNewLine
'Get the WheelID from the device (Make sure to cast it using chr())
msg = msg & "This filter wheel ID is " & chr(FW.WheelID) & vbNewLine
'Get the Firmware version from the device
msg = msg & "Firmware Version = " & FW.FirmwareVersion & vbNewLine
msg = msg & "Test Complete"
```

```
'Print the output data
WScript.Echo(msg)
```