autoMicromanager

Table of Contents:

12. Adding autoMicromanager manually to Labview Menus.

autoMicroscopy

Experimental microscopy often requires an experimentalist to build a new setup from scratch, outside the normal parameters of the standardized commercial microscopy solutions. For example, when using an optical trap, one might want to be able to use a joystick to move an optical stage around but also to switch from manual operation of the stage to a computer controlled operation after a certain trigger is detected by a Labview program that is used to analyze the microscopy images.

We present a scripting toolkit for the acquisition and analysis of a wide variety of imaging data by integrating the ease of use of various programming environments such as Labview, Igor Pro, Mat lab, Skylab and so forth [REFS]. This toolkit is designed to allow the user to quickly program a wide variety of standard microscopy components for custom microscopy applications. Included are both programming tools as well as graphical user interface (GUI) classes allowing a standard, consistent and easy to maintain scripting environment.

Setting up a custom microscopy solution requires a period of time in which Labview or some other programming environment is used to program the required behavior for the components making up the specific experiment. This process often requires assembling a lot of code to set up the microscope, camera, stage, shutters and other equipment on the table. This can require extensive troubleshooting or result in code that can only be used for that one particular application. There are already many microscopy software solutions on the market, allowing acquisition and analysis of images as well as control over a commercial microscope. These packages are limited in two important ways. First, they are often proprietary, making them both expensive and difficult to change. Second, they are often single purposed for one particular brand of equipment. This package is intended to be a powerful and free replacement for such programs as Scientific Imaging Toolkit, Chicago, SIDX, or Digital Optics.

In order to solve this problem, the Micromanager[1] project has been attempting to build a constant application programming interface (API) that can take the complexities of hardware programming and provide a constant and easy to use interface for any application. The Micromanager project uses Java to program its GUI. While Java is a powerful programming language, it lacks the ability to interface with many other programming languages that are used experimentally, notably Labview and Igor Pro. In addition, the Java Native Interface can make importing machine native libraries difficult. We present a solution that leverages the power of the Micromanager platform to most of the common scripting and image analysis scripting languages. The API has been tested with Labview, R[2], Scilab[3], Matlab and python (with SciPy[4]). The API has plugins that can feed the acquired images back into ImageJ[5] returning the circle to Java compatibility.

This toolkit was created with C#, a .Net language from Microsoft. The .Net framework allows a great number of programming languages and interfaces to work together including C#, visual basic, C++, python and ruby, as well as many more.   C# was chosen as the .Net framework language for its power, speed, and also for its similarity with Java.

This tutorial will consist of a number of simple examples that show how to access the micromanager core, a number of simple examples of how to control the camera and a variety of devices, and finally a fully automated version of the device.

Overview of Program structure.

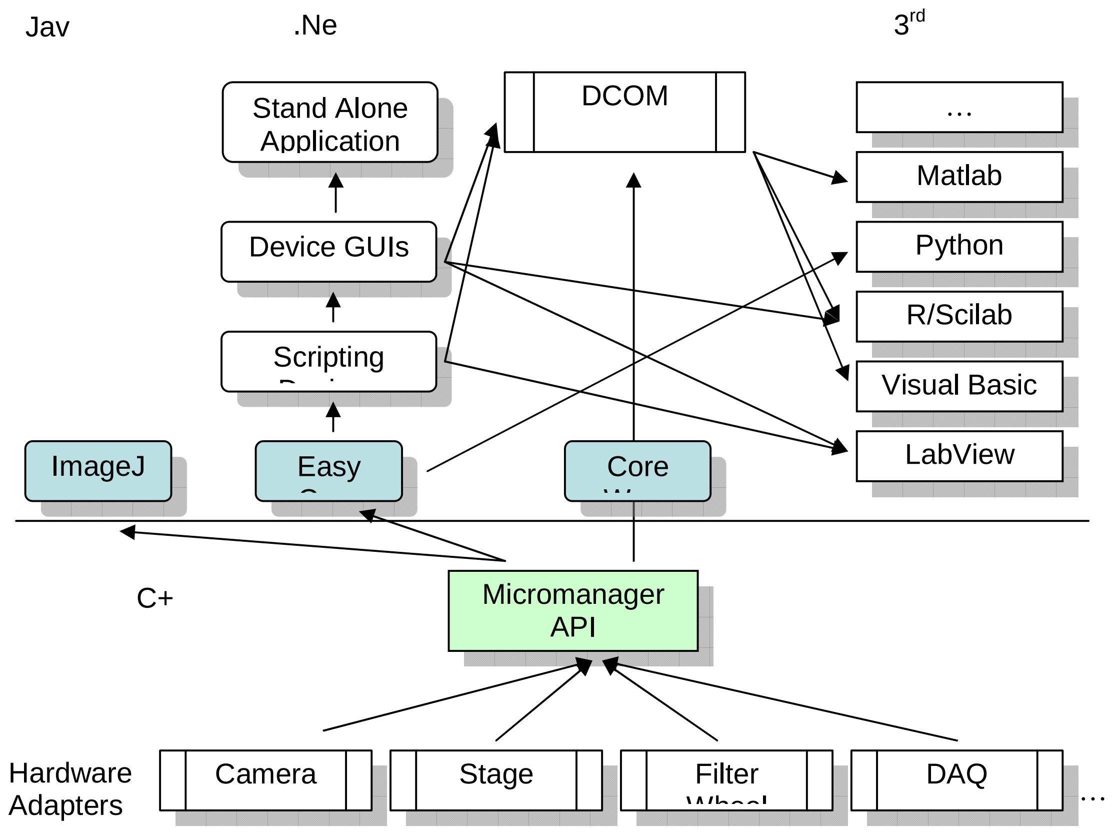


Figure 1: A general overview of all the ways that you can use the package to link c++, microscopy devices and any number of higher level programming languages.

Microscopy Toolkit is designed to leverage the considerable power of the Micromanager program for use in scripting applications. In order to help the end user in using the full power of this program, a quick overview of the Micromanager library is included.

The Micromanager library provides a consistent set of interfaces for a number of common microscopy tools such as cameras, filter wheels, shutters and stages. These interfaces provide the means for Micromanager to talk to either a Thorlabs filter wheel or a Leica filter wheel seamlessly. The addition of new hardware adapters to the system can be accomplished by creating a new hardware adapter using the interfaces defined by Micromanager. Creation of these adapters is shown on the Micromanager website (http://www.micro-manager.org/documentation.php).

Once the Micromanager engine is set up, it is possible to use the overlaying API, which we call EasyCore, to control the microscopy system Graphical Interface. For maximum power, the AutoMicromanager engine is set up with two components for each possible microscopy device. The first component has does all the hard work for scripting the device and handling device properties. The second device is a graphical component that has all the buttons for the user to interact. For an example, the package includes a device component for running a signal IO DAQ. There are two specific GUI components that work with the signal IO component. One is a signal generator and one is a galvo driver. All components work as plugins so they can be inserted into the code with ease to make the program even more powerful.

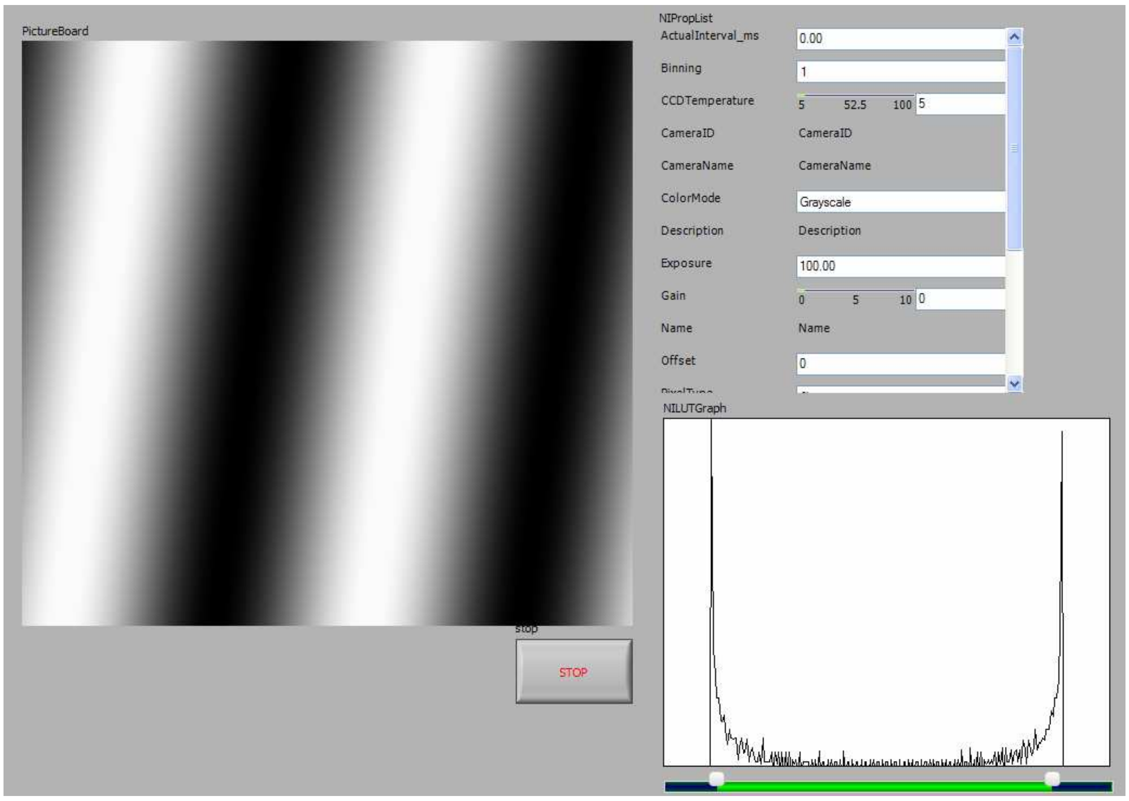
This allows the maximum flexibility and power of the Micromanager framework, while shielding the user from the complexity of C++. Finally, the framework allows microscopy devices to be added into Micromanager which are difficult to control from C++, such as .Net controls, COM widgets and activeX controls. Everything can be linked into one easy to use package. A document on the engine will be created at a further time. This will be specific for the .Net framework.

Instructions on use with LabView

This help file is structured as a list of common microscopy tasks in LabView. Along the way, we will try to explain how the different components of the code work together. The examples discussed here are available for download in LabView version 8.2.

First you'll get an idea of how to work with the microscopy toolkit in LabView and then we will work our way to a virtual microscope with a camera and a stage with a joystick control.

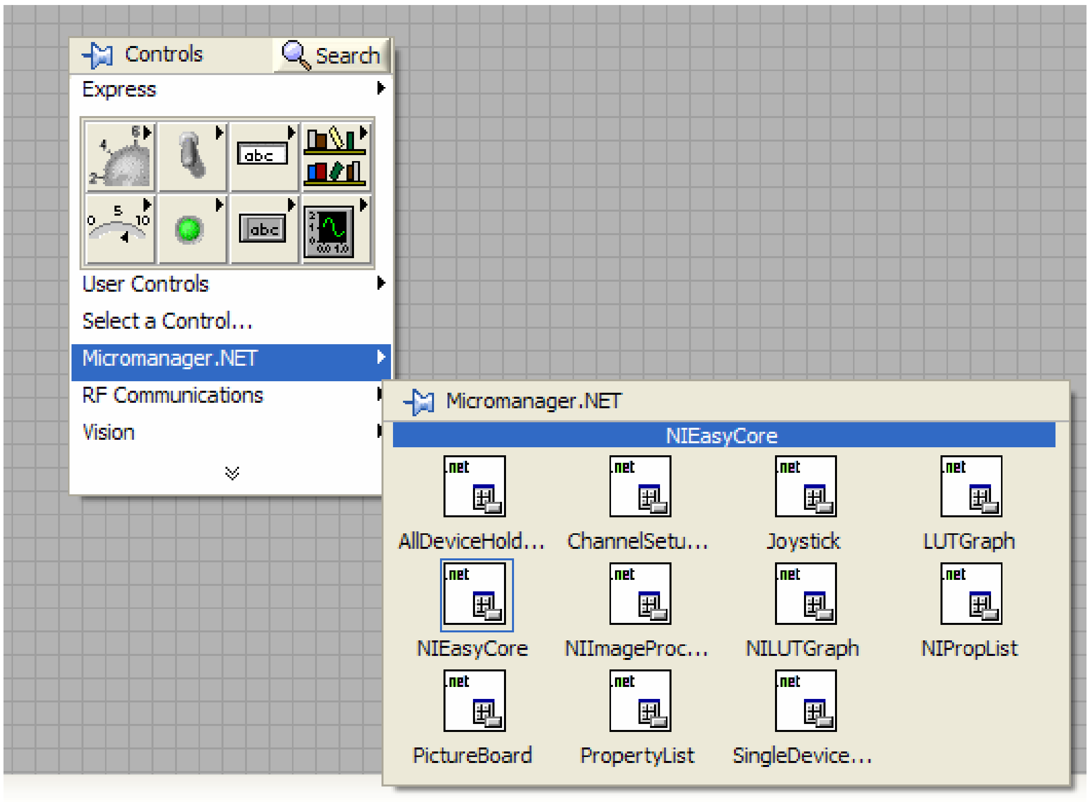
It takes several steps to make a simple application that opens a camera and then displays the image on the screen after the brightness and contrast have been adjusted. We have to import .Net into Labview, then we have to start the EasyCore API and then we will create a camera. Along the way we will explain two things that most LabView users will appreciate a tutorial on: how to import a library and how to invoke nodes. An example of the resulting Front Panel is shown below.



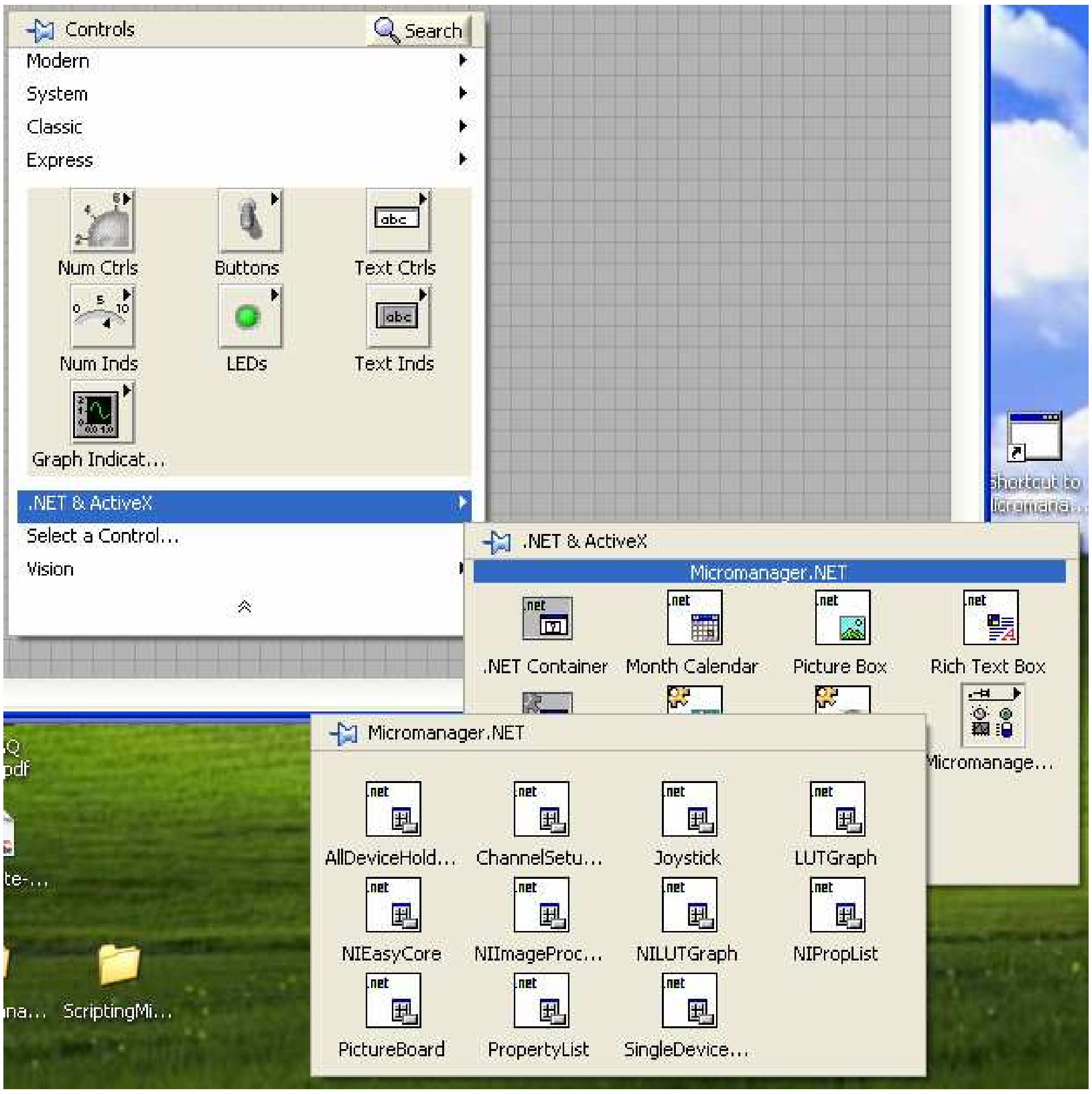
ActualInterval_ms	0.00
Binning	1
CCDTemperature	5 52.5 100 5
CameraID	CameraID
CameraName	CameraName
ColorMode	Grayscale
Description	Description
Exposure	100.00
Gain	0 5 10 0
Name	Name
Offset	0

Importing .Net

The first thing that must always be done is to start the Micromanager core. This is performed by starting the .Net Invoke Node. In LabView this is done by opening a blank VI, and while in the front panel, right clicking on the front page and finding the



For Labview 8.0 the controls will appear in a submenu of .NET & ActiveX Controls



AutoMicromanager tab and then clicking on the NIEasyCore box. Drag the NIEasyCore control to the front page to begin. This will allow Labview to use the functions of the Micromanager library.

Invoke Nodes

Now switch to the block diagram to wire this simple example. Right click on the NIEasyCore terminal, use the .Net palette to select the invoke node as shown in Figure 5. This lets the Labview program call functions inside of .Net. .

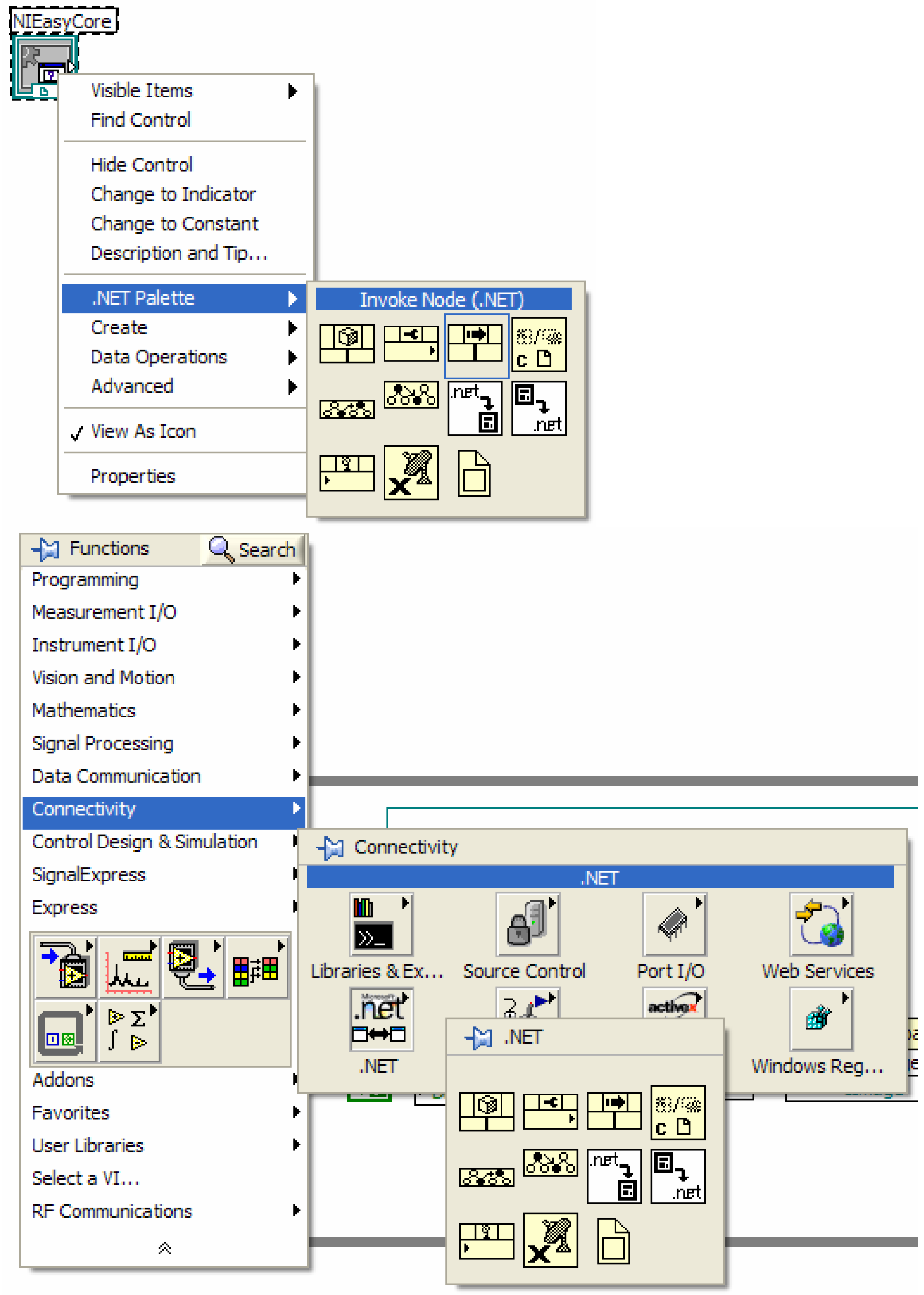


Figure 5. Two ways to get to the .Net Invoke nodes.

Now wire the NIEasyCore terminal to the invoke node's reference node and then right click on the **Method** line. As you can see on the Block diagram as well as in Figure 6, the NIEasyCore terminal has many callable functions related to its function as a front panel control. There are only two functions that should interest most users. Use **Select Method** and then find **StartEcore** from the list.

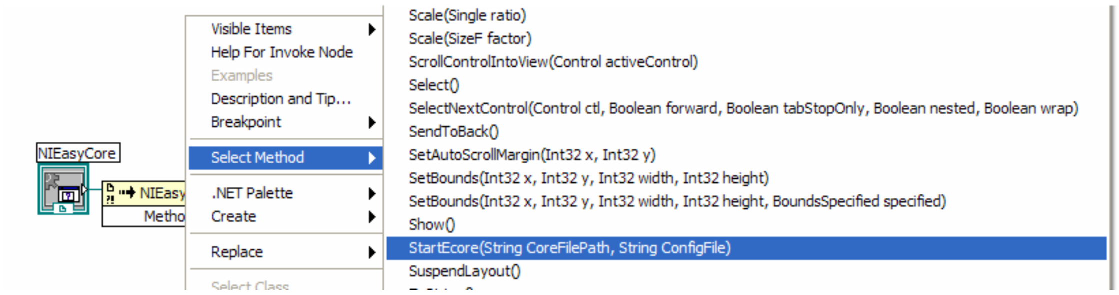

NIEasyCore
NIEasy
Metho
Visible Items
Help For Invoke Node
Examples
Description and Tip...
Breakpoint
Select Method
.NET Palette
Create
Replace
Select Class
Scale(Single ratio)
Scale(SizeF factor)
ScrollControlIntoView(Control activeControl)
Select()
SelectNextControl(Control ctl, Boolean forward, Boolean tabStopOnly, Boolean nested, Boolean wrap)
SendToBack()
SetAutoScrollMargin(Int32 x, Int32 y)
SetBounds(Int32 x, Int32 y, Int32 width, Int32 height)
SetBounds(Int32 x, Int32 y, Int32 width, Int32 height, BoundsSpecified specified)
Show()
StartEcore(String CoreFilePath, String ConfigFile)
SuspendLayout()

Figure 6.

1. Starting EasyCore

(This lesson corresponds to Tutorial1_Start Camera.vi in the examples folder)

There are two ways to run AutoMicromanager: 1. Manually set up the cameras and peripherals and then run all the operations in the Labview script. 2. Load a hardware config file and then listen in to the self standing operation of the suite, while directing operations from the script.
Option #1 is good for learning how to use the suite, so it is taught first in the manuals, but option #2 is better for operations, so you should familiarize yourself with the manual options and then start work on the hardware config files.

StartEcore requires one parameter to start. A blank string is used to provide this parameter indicating to StartECore that there is no config file, config files are a powerful method to define the microscope and then interact with the system. The result should resemble Figure 7

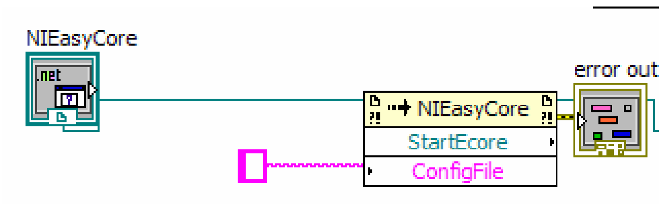


Figure 7.

StartEcore now provides most of the important functions for controlling the whole microscopy setup, but NIEasyCore has one more function that makes using AutoMicromanager easier. This is the GetHelpers function that is dedicated to making a few steps easier in the Labview environment. This function must be called only after the StartCore Node has been called. In order to perform this step, a .Net invoke node is again placed on the block diagram and the reference is wired to the previous node.

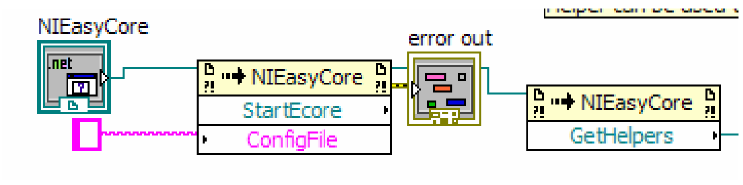


Figure 8.

1a. Creating a Device

GetHelpers contains functions for creating a number of devices in the Labview environment. We place another .Net Invoke node on the block diagram to create the camera device. We must now wire the GetHelpers block to the new invoke block and then provide 3 different parameters to determine what kind of camera is desired.

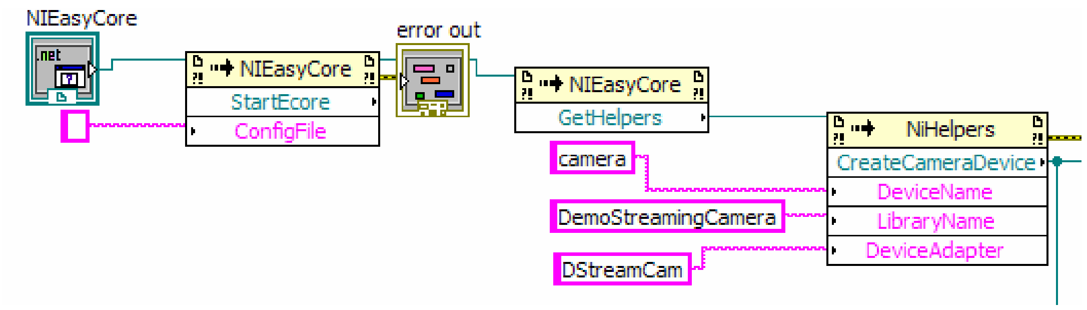


Figure 9

The first parameter, DeviceName, is arbitrary is used to index the camera. For simplicity it is called “camera”. The next parameter refers the Micromanager library name, last is the DeviceAdapter.   The names of these parameters refer to the device adapters that tell Micromanager how to function. If you need to find these names, there is a VI in the example folder called **FindLibraryAndDevice.vi** that shows the names of the libraries and their contents for all adapters on the system. You can also ask about this on the forum.

In this case we are just building a demo camera that does not attach to any hardware. This is done by telling the system to index this device as “camera”, use the library “DemoStreamingCamera” and finally used the “DStreamCam” for the adapter. This gives a very simple and boring demonstration camera for the system.

Next an invoke node is used to MakeOfficial shown in figure 10, meaning this camera should be used as the primary camera for the whole system. Last, the property list has to be attached to camera to give the user full control of the camera on the front panel of the VI. This is done very simply by telling the camera where its property list is with the SetPropUI Invoke node as seen in figure 10.

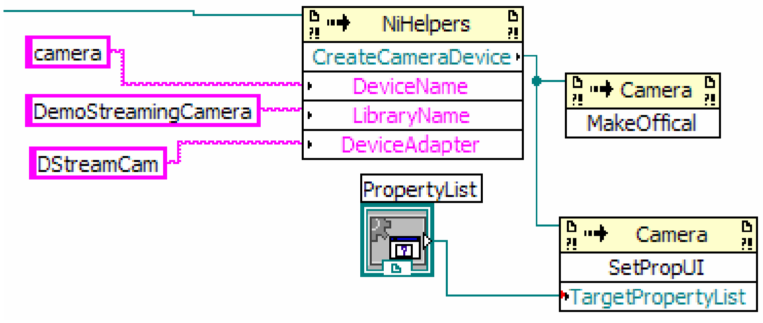


Figure 10.

1b. Displaying an Image

Now the camera is turned on and ready to go. It is now time to acquire the first image. This is done by using another .Net Invoke node which is connected to the camera line and then the method that is selected is: SnapOneFrame as shown in figure 11. This method takes one parameter, DisplayImage.   This parameter is for more advanced usage and is set to false for now.

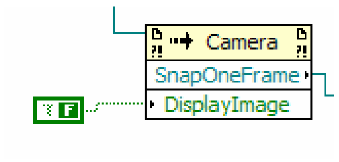


Figure 11.

The result of this node is a CoreImage. This is a powerful tool for the manipulation of intensity images. It can be used for image addition, subtraction, extraction of statistics and to change the image into a Labview array. For now, it will just be used to set the contrast of the image and display it on the screen.

A Look Up Table (LUT) is a common method to adjust the contrast of microscopy programs. The NILutGraph is designed to help the end user to set the contrast easily. So the Coreimage is sent to the NILutGraph for contrast adjustments with a .Net invoke node as shown in figure 12.

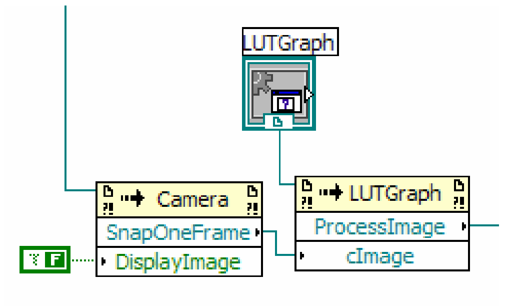


Figure 12.

This results in another CoreImage which has its contrast settings set as defined by the NILutGraph control. This image is now sent to the PictureBoard for display as shown in figure 13.

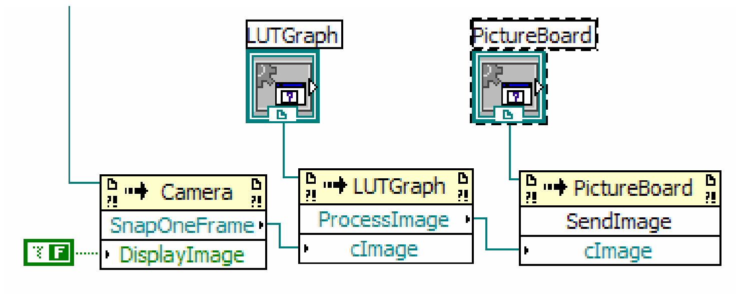


Figure 13.

We have now managed to set up a camera, make it so the end user can adjust the camera as desired and then taken a picture. This took 8 simple invoke nodes to accomplish and can be greatly improved upon with even a slight level of programming ability. The next sections will show how to make an even more effective microscopy setup. It may be useful to now try this program with your own camera to check the performance.

2. Getting the Data: Converting Images to Arrays

(This tutorial corresponds to the Tutorial2_Image Manipulation.vi File in example Folder)

It may be useful for your particular application to have the data in an array format or in a format that can be used with the NI vision package. This is done very easily after looking at the Tutorial1.VI file you can make a few small modifications. The CoreImage object has a variety of tools to convert the intensity image into a variety of array types for use with normal Labview or NI vision components. The CoreImage class has a large number of functions for these types of manipulation. Using the proceeding example, we will insert a command to convert the image to an array. Figure 2.1 shows this insertion.

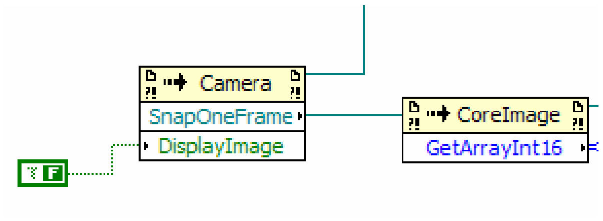


Figure 2.1

By right clicking on the GetArrayInt16 invoke node and then using **Select Method** you can see that there are a number of GetArrayXXX methods. Select the appropriate data type and then the output will be an array containing the image in an array type. This array can be manipulated in any number of ways, so the inverse will be performed and then the new image sent back to the PictureBoard as shown in figure 2.2.

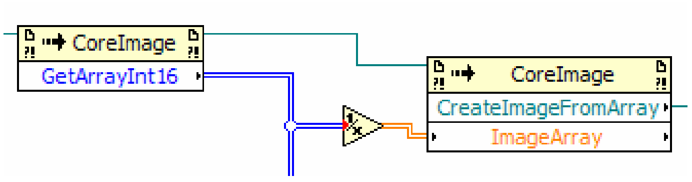
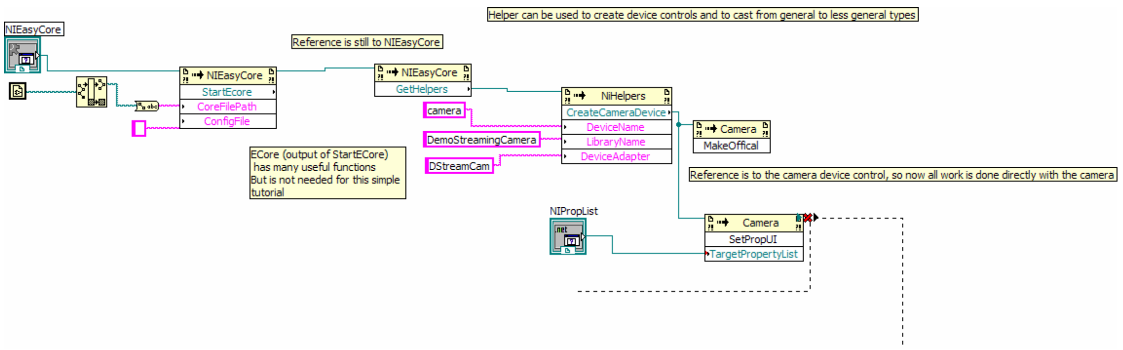


Figure 2.2

You must select the correct input type into the CreateImageFromArray method. The result is a new CoreImage that works the same as before for display and manipulation.

3. Streaming Camera/ Saving Images

For some situations, it is necessary to have the camera operate in a streaming mode. It is also important to save your images. This means that the camera sends a series of images to the computer with no break. This can result in a flood of information entering the camera. For this mode the camera is started in the same manner as in example 1.



Helper can be used to create device controls and to cast from general to less general types

Reference is still to NIEasyCore

ECore (output of StartECore) has many useful functions But is not needed for this simple tutorial

Reference is to the camera device control, so now all work is done directly with the camera

Figure 3.1

Next working with streams requires 4 commands, which figure 3.2 shows.

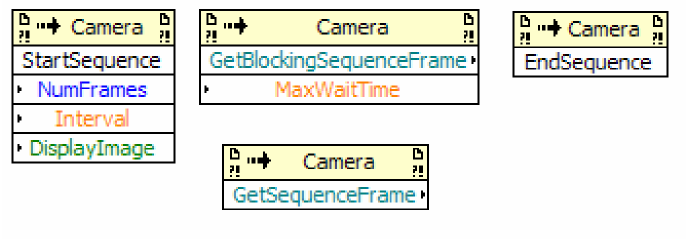


Figure 3.2

StartSequence must be called first which tells AutoMicromanager to set up all the variables for a sequence acquire. You must specify how many frames the camera should send, as well as the interval between the frames in milliseconds. Last, you can specify if the internal system should display the image. This option should be false at this time.

Next, the frames need to be read from the buffer as they are grabbed from the camera. There are two options provided here. GetBlockingSequenceFrame waits until a new frame is acquired and then the image is returned, GetSequenceFrame will read from the buffer and return an image, a null, or an error depending on the state of the buffer. It is you requirement to deal with these eventualities. It is recommended that the GetBlockingSequenceFrame is used. Last the Sequence should be formally ended with EndSequence. Tutorial 3 gives a simple example of the sequence tool in action.

Last we save the last frame from the sequence in two different formats. This is done by calling the Save .Net Invoke Node on the CoreImage after it comes out of the camera.

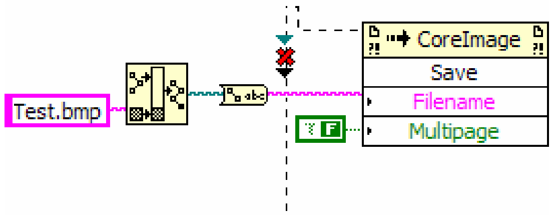


Figure 3.3

You must specify the filename. The extension of the filename will determine what format the file will be saved. If you are saving a stack, then the multipage option can be used.

4. Adding Stage Control /Adding More Devices

(This tutorial corresponds to the file Tutorial5_Camera and Stage.vi)

Adding a stage to the camera is even easier than working with the camera. First you should tell EasyCore that you want to start and stage device. This is done with the line from the GetHelpers node. An example is showing in figure 4.1. Information about the meaning of all of these .Net invoke nodes is shown in section 1.

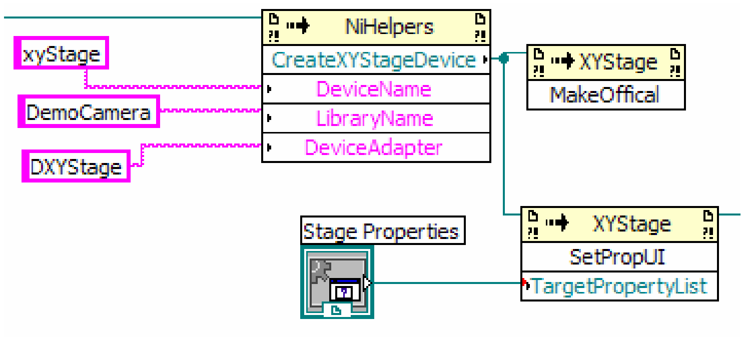


Figure 4.1

After these nodes have been passed, there are a few commands that can be sent to the stage to control its position. These are shown in figure 4.2

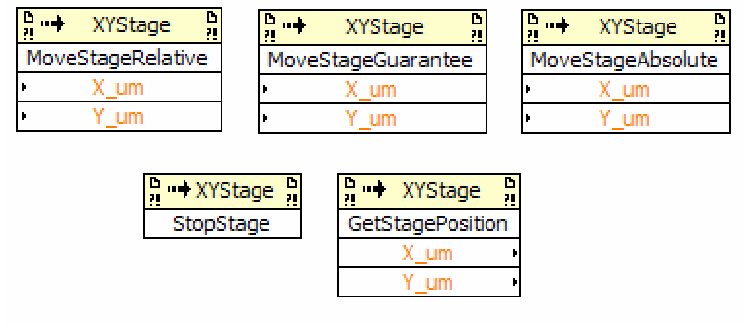
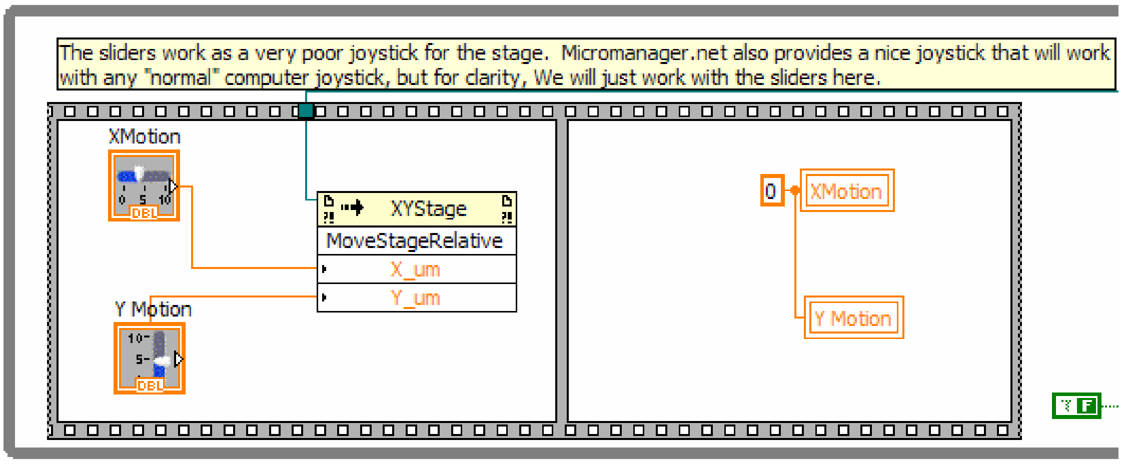


Figure 4.2

MoveStageRelative will move a distance relative to the current stage position, it will return before the motion is complete. MoveStageAbsolute will move the stage to an absolute position, it will return before the move is complete. The meaning of this absolute position depends on the make and type of the stage and should be checked with your equipment manual. Last is MoveStageGuarantee. This Invoke node will send the stage to some absolute position and then return after that position has been reached.

If you wish to stop the current motion, StopStage is used and last if the current position of the stage is needed, GetStagePosition will return the position as the stage equipment reports it.

Now we will connect the output of CreateXYStageDevice to the MoveStageRelative node in order to move the stage around. In order to control the stage we use two sliders, one for the X axis and one for the Y axis. This is for simplicity and the AutoMicromanager library provides a nice joystick control for this purpose. This joystick control is shown in later examples. The MoveStageRelative node is placed inside a while loop as shown in figure 4.3 so it will be called over and again. By default the previous command is cancelled and then the new command is used to control the stage. The sliders are set back to zero in order to have the stage stop once the mouse releases the axis slider.


The sliders work as a very poor joystick for the stage. Micromanager.net also provides a nice joystick that will work with any "normal" computer joystick, but for clarity, We will just work with the sliders here.

This same loop should contain the code to acquire images. In this way an image will be acquired while the stage is adjusted allowing precise adjustment of position.

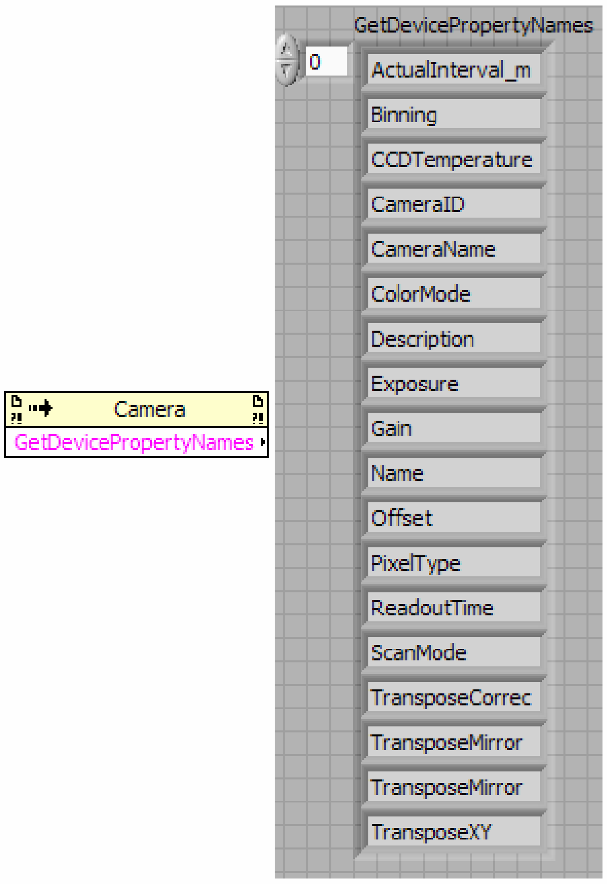
The tutorial camera does not change with the stage position as the demo camera does not communicate with the demo stage.

Any further device can be added in a similar fashion allowing easy control of the microscopy setup with any number of difference devices.

5. Programmatically Adjust Device Values

(This chapter corresponds to the file Tutorial6_Device Properties.vi)

It may be required to change the various parameters of each device within the program instead of just allowing the user to adjust these values from the **PropertyList.** There are a number of tools to control all the devices in a consistent and powerful way. First, you can get information about the device by querying the device for its property information. After setting up a camera as was done in example 1, a new invoke node can be called, GetDevicePropertyNames. This node will return all the properties associated by the device adapter as shown in figure 5.1. This list changes from camera to camera, you must make allowances for your own equipment.



ActualInterval_m
Binning
CCDTemperature
CameraID
CameraName
ColorMode
Description
Exposure
Gain
Name
Offset
PixelType
ReadoutTime
ScanMode
TransposeCorrec
TransposeMirror
TransposeMirror
TransposeXY

Figure 5.1

These properties names are all case sensitive so you must get both the spelling and the case right or an error will be generated.

Once the property names are know, it is quite easy to manipulate the values as shown in figure 5.2. SetDeviceProperties allows a new value to be placed in the property. When this is done, the PropertyList will also be updated for the end user to see what is happening. GetDevicePropertyValue will query the program for the current value. With these two commands you can control almost any part of the microscopy setup. One thing must be considered, the information must be passed as a string for all variable types. This is a feature of the Micromanager core to allow the maximum control and flexibility for the properties. It is quite straightforward to convert the values to strings and back to values in Labview and you can see this in the tutorial VI.

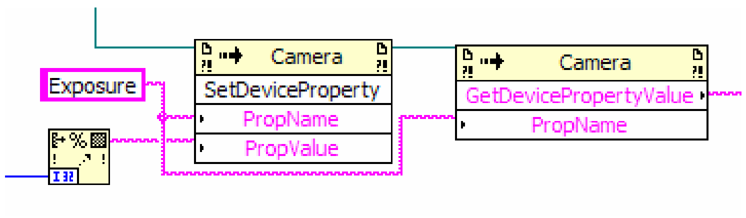


Figure 5.2

Last, it may be desirable to write a VI that adapts to what camera is being used. This requires just one more .Net Invoke node that returns all the information about a property to allow full control and advanced FrontPage information. As shown in figure 5.3, GetDevicePropertyInfoDetails returns all the information about what a property can do. You must specify which property you need the info for and then you will get a number of variables in return. First is the current value of the property, next is HasLimits. This is true if the value must fall between a certain max and min given by MaxValue and MinValue. These values are meaningless if HasLimits is false. Next is the property type. Types can be Undef, String, Float, or Integer. Some times the properties are only there for reporting, so ReadOnly is set to true.   Last, if the property has only a set of allowed values, these will be specified with HasAllowedValues and then the values will be in AllowedValues in a string format. With this information it is possible to construct your own control that functions like the PropertyList or for the program to already know what the limits can be for gain.

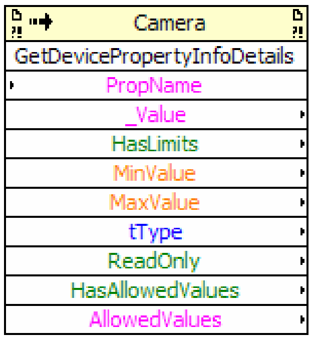
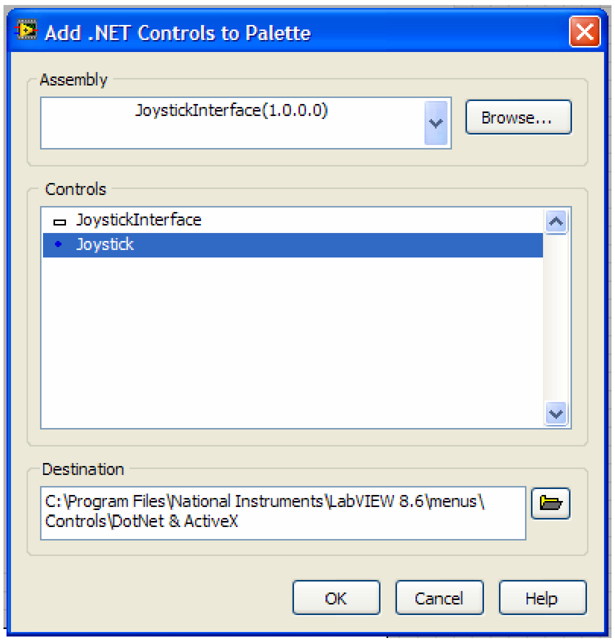


Figure 5.3

6. Advanced Joystick Control

(This tutorial corresponds to the file Tutorial7_Joystick Control.vi)

As Labview does not have a nice joystick for use with the stage, the Joystick Control is provided. Click on the Menubar at the top of the front panel, the **Tools/import/.Net controls to palette.** Now click the browse button to navigate to the directory with AutoMicromanager. There you will find a file called JoystickInterface.dll. There you can click on the only available control, the Joystick as shown in figure 6.1.


Add .NET Controls to Palette

Assembly: JoystickInterface(1.0.0.0)

Controls:
- JoystickInterface
  - Joystick

Destination: C:\Program Files\National Instruments\LabVIEW 8.6\menus\Controls\DotNet & ActiveX

Figure 6.1

Now you can place the joystick on the Front panel in order to make it useful. Do this by right clicking on the Front panel, then finding the **.Net & ActiveX** menu and finding the Joystick Icon.

The result is something that looks like figure 6.2. The orange ball can be pulled around by the mouse to control the stage, or will react to any plug

& play type of joystick that has been added to the system. Further support for other mouse buttons will be added later.

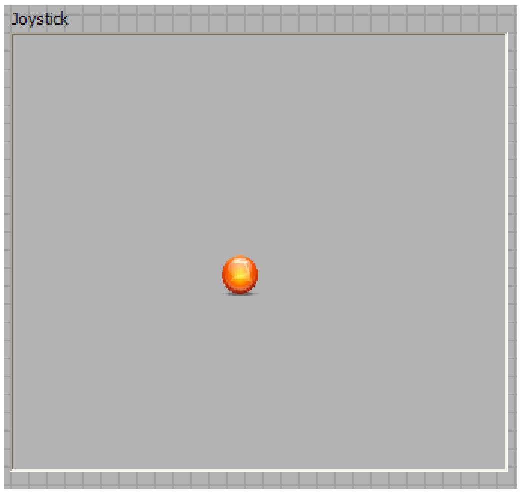


Figure 6.2

The joystick has a few simple methods that can be used on the block diagram to control its behavior. First you must connect the Joystick control to the interface and to the game ports. This is done with the BeginJoystickAction node as shown in figure 6.3. If you have a plug & play joystick attached to the system, you can specify true for the parameter, and the joystick will search for the hardware. The hardware must not be in use with any other application for this to work and

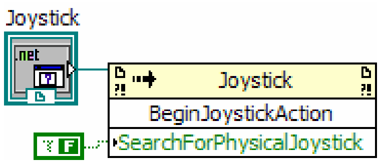


Figure 6.3

Once the joystick is started we can begin. It is possible to receive event notifications when the joystick is moved, or to just check the state of the joystick on each iteration of a while loop. This is done with the GetJoyStickCoords node as shown in figure 6.4. This node returns the current joystick state. The default returned value is from -1 to 1 for joystick control. You must multiply this value by a scale factor that is appropriate for your microscope.

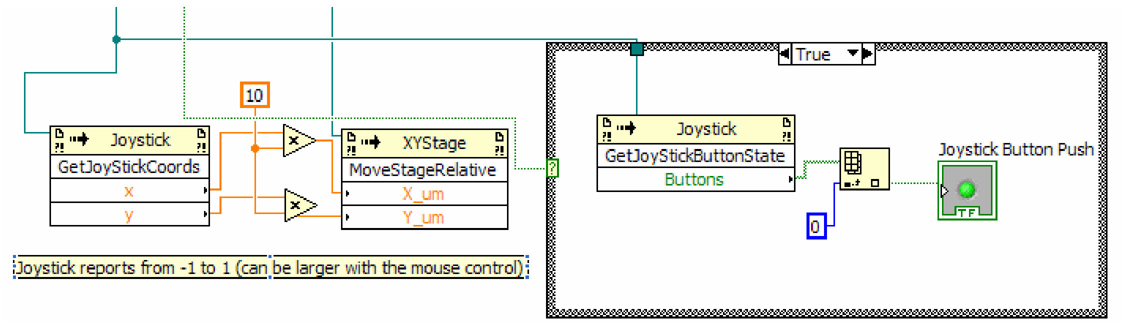


Figure 6.4

Last, if you are using a physical joystick, you may wish to know the state of the buttons. GetJoyStickButtonState returns all the button's states on each query. You must map the index of the array to your particular joystick.   (This is done most easily by pushing the buttons and seeing what part of the array changes)

7. Putting it all together/Labview in Full Control

(This chapter corresponds to Tutorial8_Camera_Stage_Filter.vi)

There are no new concepts introduced in tutorial7, it is instead an example of how multiple devices can be integrated with a nice graphical user interface to produce a very simple microscope. You can add scripting at will with this setup.

8. Working with Channels

(This chapter corresponds to the file Tutorial9_Channels.vi)

Channels allow the microscope to change its configuration in any definable way and images to be made. These changes can be movement of filter wheels, application of lamps or changes of exposure. They always include the changing of the properties of the microscope. We will start the microscope in the same manner as we have for all the other tutorials, with StartEcore, GetHelpers and CreateCamera. But we must add a new .Net control to the vi, so select the front panel and then **Tools/Import/.Net** controls browse to the Micromanager_net.dll and then when the names are listed expand the channels tab and select ChannelSetupControl. Place two of these controls on the front page. Next the channels must be created. This is done in the context of a ChannelGroup which is found in The helpers object as shown in figure 8.1

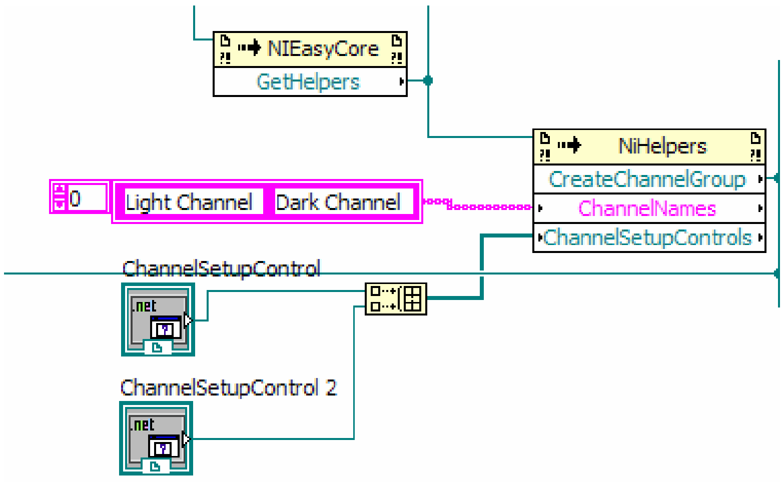


Figure 8.1

You must provide names of these controls and also an array of the ChannelSetupControls. The numbers must be equal or an error will be generated.

The Helpers object will generate a ChannelGroup object with two channels that are already defined. So now we can put commands into the channels.

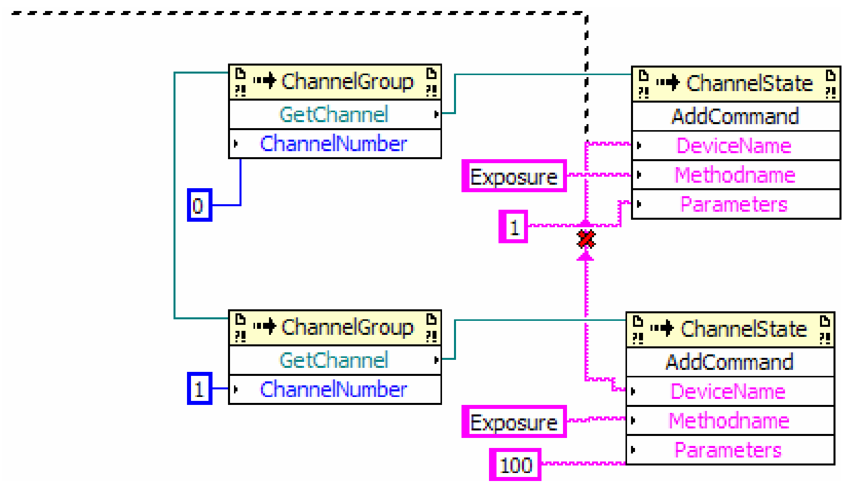


Figure 8.2

Channelgroup has a GetChannel command that will extract each of the channels. Then you can use AddCommand to setup the channel. You can have any number of commands, but in this case we will only have one. The devicename must match the device name from createCameraDevice earlier. In the first channel we with to have a dim exposure of 1 ms as shown in figure 8.2. In the second channel, we have a longer exposure of 100ms.

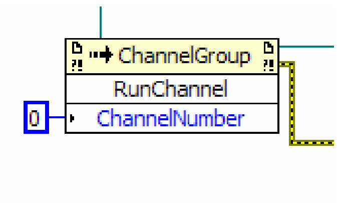


Figure 8.3

Last we enter into a loop to collect images. Now we use channelgroup to switch from one channel to another to collect images. The images are sent first to one PictureBoard and then the other.

This technique is very valuable if a number of changes must be made at the same time. Then the program can take care of monotonous changes and RunChannel is all the programmer has to use. This is also very useful if the end user wants to be able to change the channel definitions as the program is running from the ChannelSetupControls.

9. Image Processors / Running AutoMicromanager in Parallel with Labview

(This tutorial works with the file Tutorial9.0_Image Catching.vi & Tutorial9.2_Register Paint Board.vi)

There a number of situations where you would want Labview to run some piece of equipment, while AutoMicromanager mostly runs independently. This will greatly improve the speed, as well as allowing you to do less work in maintenance of your VI. This will also be an improvement for flexibility once you learn how to implement a hardware configuration file.

To introduce the concept of imageprocessors, we will wait to show the information about config files until later. For this VI we start in the normal method of StartEcore, GetHelpers, and CreateCameraDevice as we have for most of VIs up to this point. After this, a change is made. We must register the PaintBoard surface directly with Ecore, as well as assigning a LUTgraph to the PaintBoard surface. As shown in figure 9.1, this is performed with the AttachLUTGraph and the PaintSurface .Net Invoke Nodes.

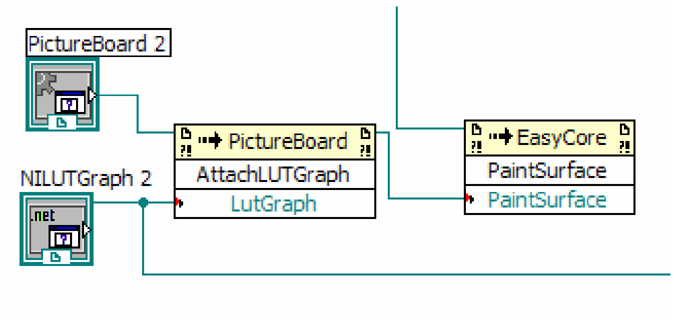
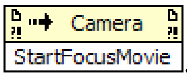
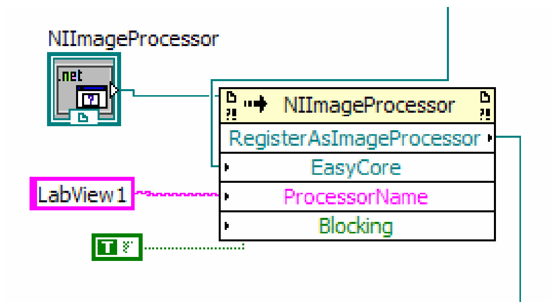


Figure 9.1

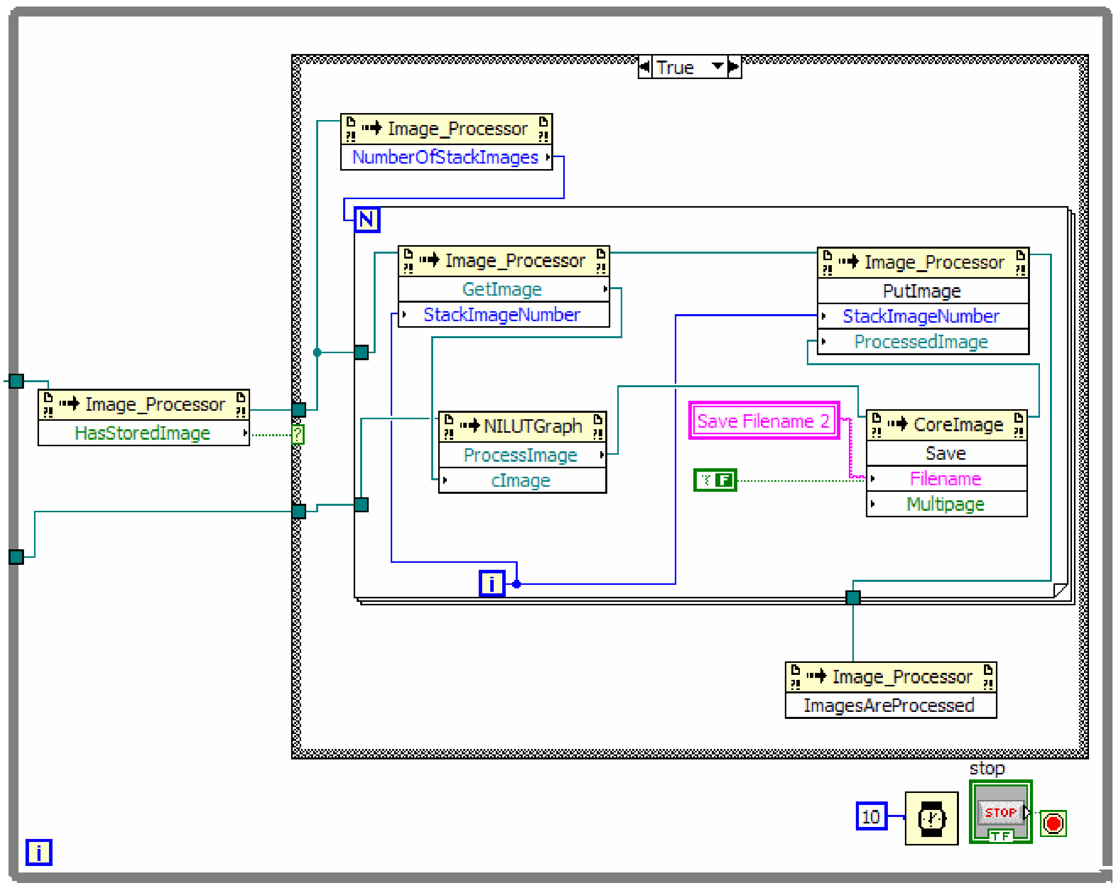
Now AutoMicromanager knows how to handle any image that is produced. You tell it to take images by telling the camera to use the StartFocusMode invoke node . You can run the VI (Tutorial9a.vi) at this point and you will see that the images are collected very quickly.

Now you have the images appearing on the screen quickly, but they are not being saved. AutoMicromanager provides a method to listen in on the stream of images and then manipulate them if needed. This is by inserting an imageprocessor. You must place the NIImageProcessor on the front page.   You can have multiple imageProcessors if needed, each will react to the emission of a new Image individually. In order for an ImageProcessor to work, it must be registered with EasyCore. This is done with RegisterAsImageProcessor



You should give the processor a descriptive name, if you planning on adding multiple ImageProcessors to the system. You must decide if you want the image processor to be blocking or not. If it is blocking, AutoMicromanager waits for your code to finish before acquiring the next image. Otherwise it just sends all the images over as fast as possible. This can be good, unless the Labview system is too slow and the memory fills up. All three examples use Blocking because it is safer.

Now AutoMicromanager knows that you wish to process Images, it knows where you want to display the images, and you have told it to start getting images.   Now a listening loop is needed to catch the images. This is accomplished with a normal while loop.



The while loop checks if there has been an image acquired, and when one is in the ImageProcessor buffer, The program moves to the next loop. Instead of a single Image like there has been before, an array of images is returned. This is due to the channel stacks. They are returned as arrays to help any processing.   So we can check how many images are in the ImageStack and then use GetImage with the proper index. The image is then manipulated, and finally PutImage sends the image back to AutoMicromanager. The very last step of the process is to call ImagesAreProcessed.   This must be called as it notifies AutoMicromanager to get new images. This must be outside the loop and must be the last thing that is executed. Strange errors will appear if this rule is not followed.

Now this example does not make sense as it saves every image to the same name, but it should be easy for you to adapt it to your needs.

10. Automatic Channel/Stack Handling

If you want to acquire a number of images   from multiple channels in a way that does not require a lot of programming effort. We are almost there with Tutorial10b.vi so we will start there. First the Microscope is setup in the same way as normal. First StartEcore, GetHelpers, CreateCameraDevice (see chapter 1). Then we have our first new concept. I wish to have each channel in its own window, so I must register the same number of windows as channels.

This is quite easy in practice as can be seen in Figure 10.1

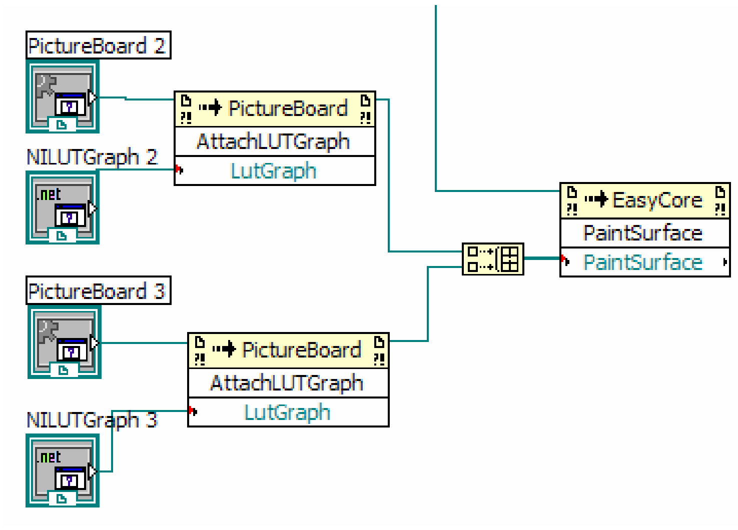
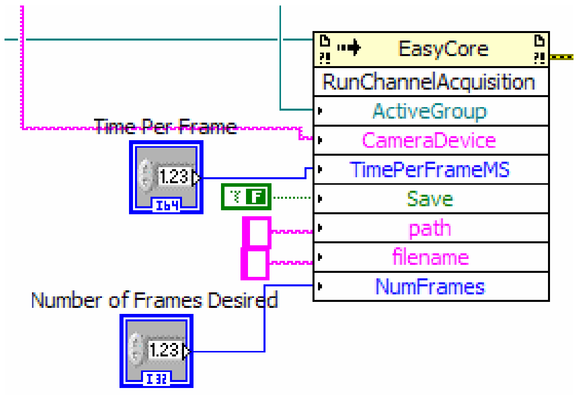


Figure 10.1

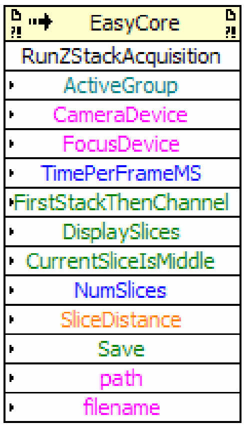
First a LUTgraph is attached to each Pictureboard, then the PictureBoards are wired into an array and then EasyCore is notified of the addition.

The next section of the graph shows the standard process for setting up a channel. See chapter 8 to get a better idea about how this works. We are now ready to acquire images with the PaintBoards registered, the channelgroup registered and the channels registered.

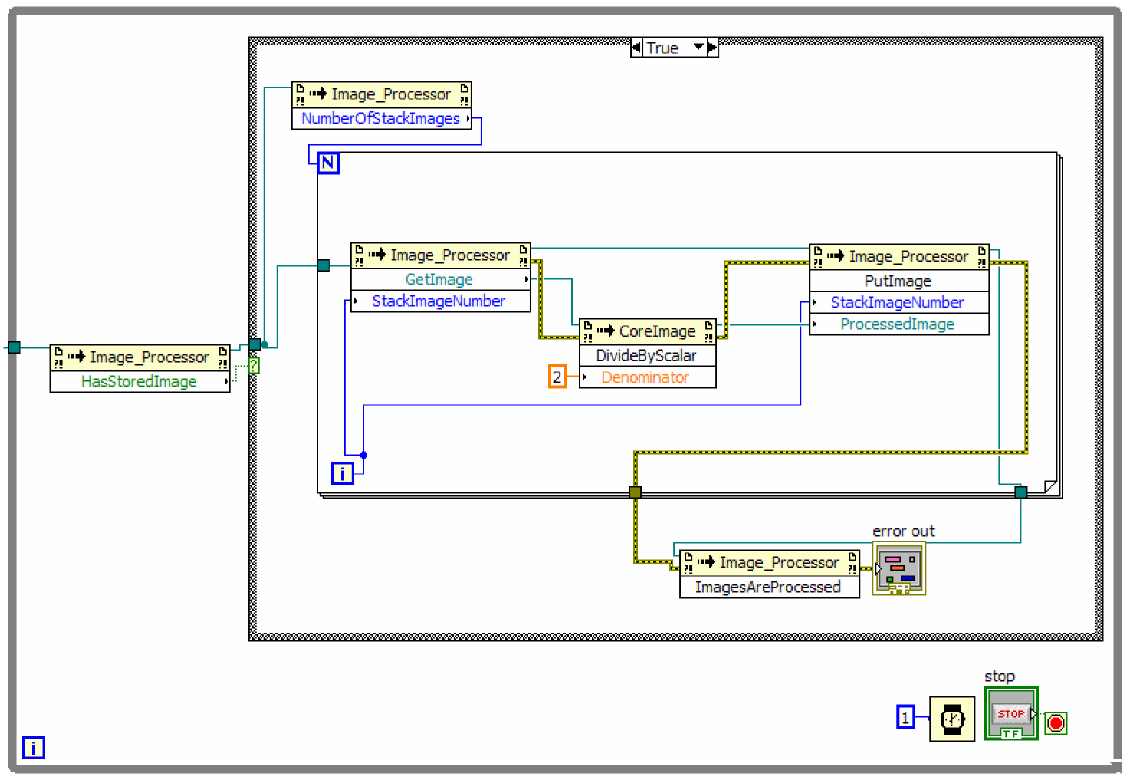
The acquiring is performed by the RunChannelAcquisition Invoke Node. There are a large number of parameters that must be addressed. ActiveGroup indicates the created Channelgroup. You can decide to use a different camera to get these images than the default camera, so you must provide the device name. Next you choose the time from one frame to the next, last you can decide here if you wish to save this sequence of images. You must specify a directory and a root filename. Numbers will be added to the filename to make the frames distinguishable. Last is the number of frames that you wish to acquire. -1 results in never ending images.



A second powerful help in getting stacks is the RunZStackAcquision. This Invoke node has all the tools needed to make a stack, a stack of all the channels, and many other options.



In Tutorial10.vi we return to ImageProcessors. If you wish to manipulate the channel data in anyway, the imageprocessors provide the best method. It is clear that Tutorial10.vi is the same through the whole top of the program as Tutorial10b.vi was. It is also clear that the Imageprocessor step was just covered in chapter 9. When the image is captured by the imageprocessor, it is sent through a trivial step and then returned to the main program. This is done for both channels as shown by the for loop.

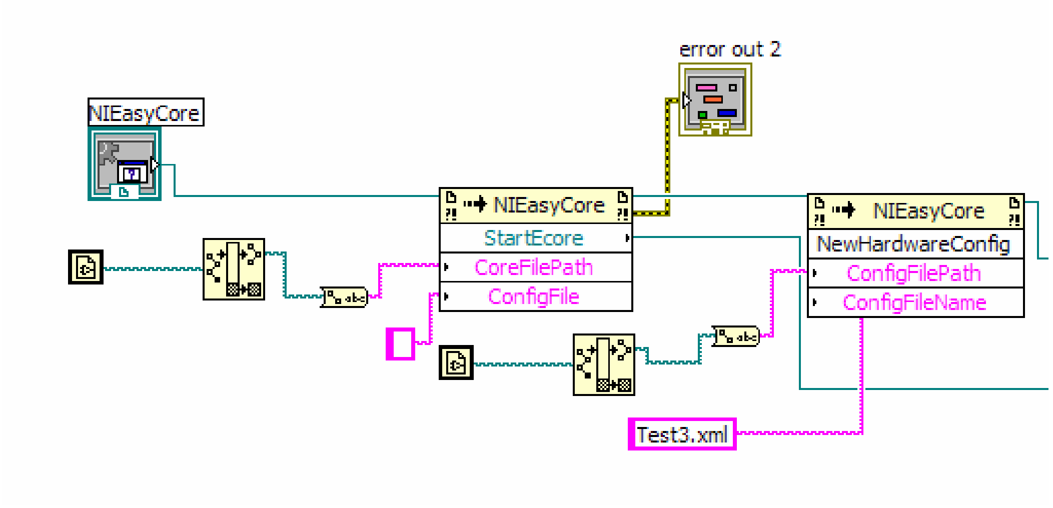


This could be done with separate processing for each channel. This is only important for the user's application. Last, you must notify the AutoMicromanager program that you are done with that image by using the ImagesAreProcessed invoke Node.

11. Loading from HardwareConfig File (The easy way)

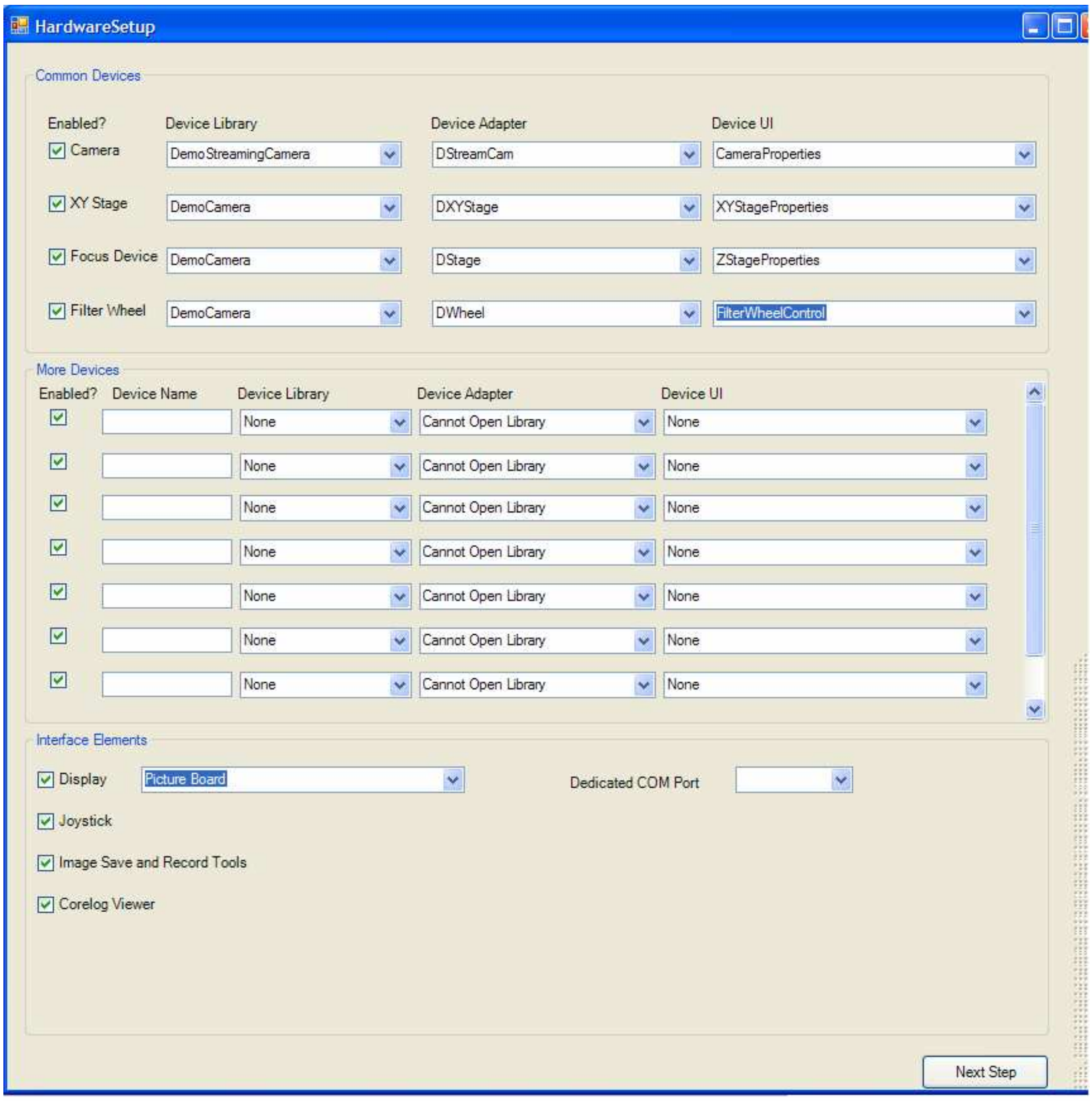
(This Tutorial links to Tutorial_ConfigFile_Device Control.vi, Tutorial_ConfigFile_Image Capture.vi, Tutorial_ConfigFile_Make Hardware Config File.vi, and Tutorial_ConfigFile_Load Config File.vi)

After all these other channels, this chapter is going to feel like a letdown. It makes it so easy to control the microscope with so little programming. First we must create a Hardware Config File. This is done with one of the commands in NiEasyCore called NewHardwareConfig.



When this command is called, you must specify where the config file should end up (directory name) and specify the config file name. It must be a name that ends with xml, just like the example.

A dialog will pop up and you must answer the questions to setup the hardware. An example if shown in figure 11.1. The DeviceUI list should be easy to guess the correct GUI from. Some experimentation can help find the rest of the devices. The FindLibraryAndDevice.vi file will show all these devices spread out to help you to identify your hardware.



HardwareSetup

Common Devices

Enabled?	Device Library	Device Adapter	Device UI
✓ Camera	DemoStreamingCamera	DStreamCam	CameraProperties
✓ XY Stage	DemoCamera	DXYStage	XYStageProperties
✓ Focus Device	DemoCamera	DStage	ZStageProperties
✓ Filter Wheel	DemoCamera	DWheel	FilterWheelControl

More Devices

Enabled?	Device Name	Device Library	Device Adapter	Device UI
✓		None	Cannot Open Library	None
✓		None	Cannot Open Library	None
✓		None	Cannot Open Library	None
✓		None	Cannot Open Library	None
✓		None	Cannot Open Library	None
✓		None	Cannot Open Library	None
✓		None	Cannot Open Library	None

Interface Elements

✓ Display: Picture Board

Dedicated COM Port:

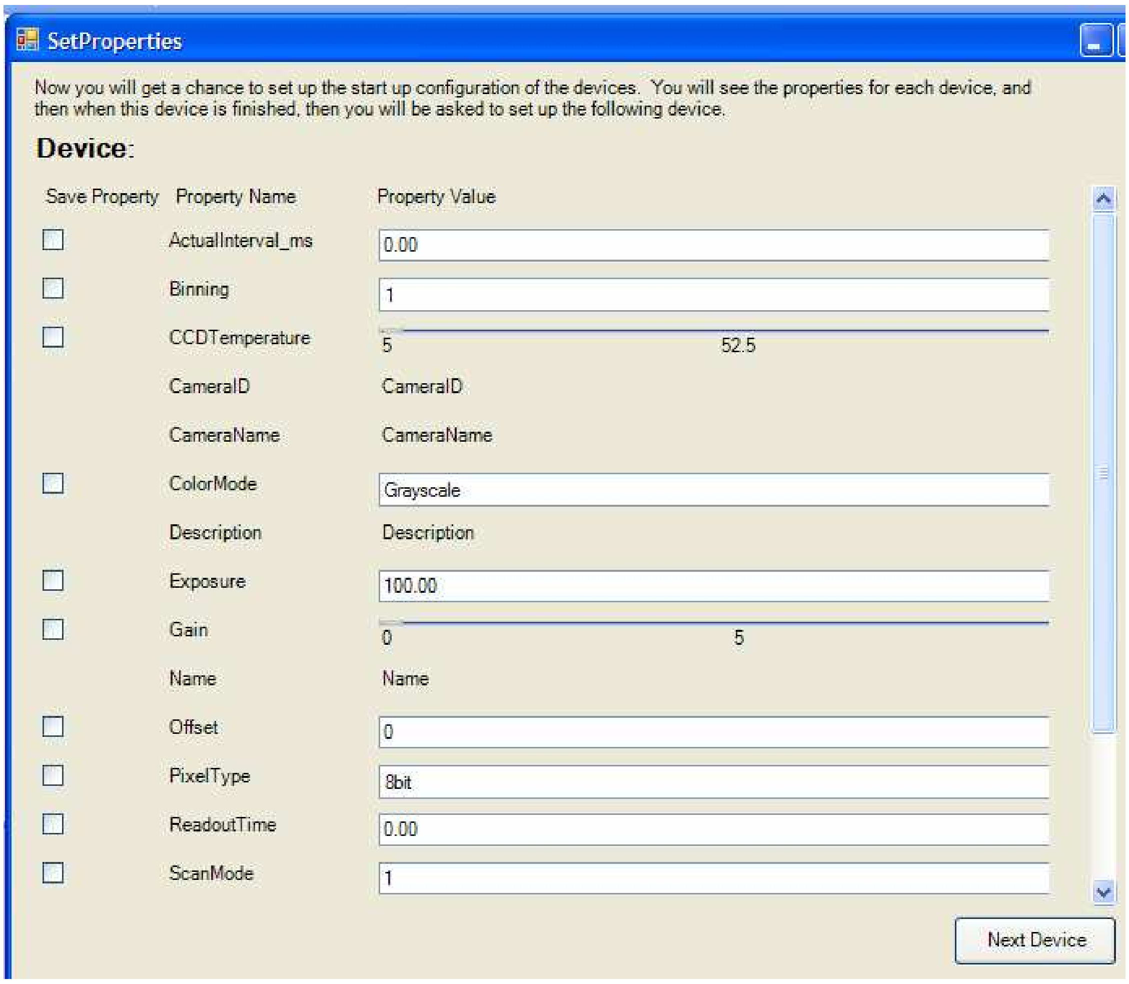
✓ Joystick

✓ Image Save and Record Tools

✓ Corelog Viewer

Figure 11.1

Once everything is selected you should click the next step button. This will bring up the properties page for your first device as shown in figure 11.2


SetProperties

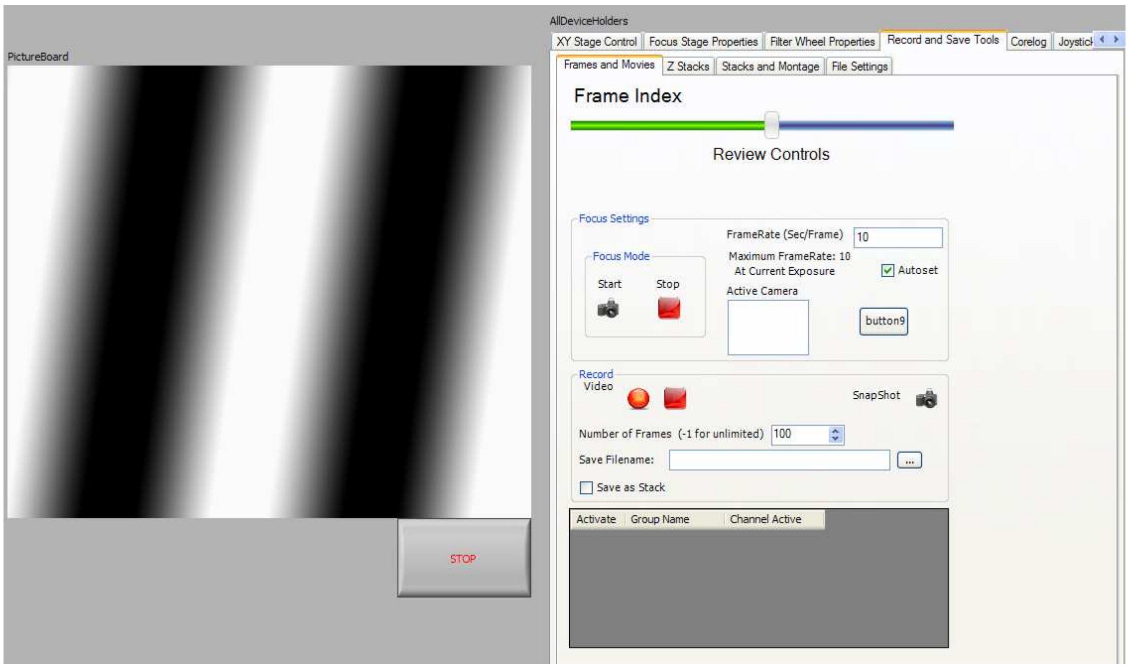
Now you will get a chance to set up the start up configuration of the devices. You will see the properties for each device, and then when this device is finished, then you will be asked to set up the following device.

Device:

Save Property	Property Name	Property Value
☐	ActualInterval_ms	0.00
☐	Binning	1
☐	CCDTemperature	5 52.5
	CameraID	CameraID
	CameraName	CameraName
☐	ColorMode	Grayscale
	Description	Description
☐	Exposure	100.00
☐	Gain	0 5
	Name	Name
☐	Offset	0
☐	PixelType	8bit
☐	ReadoutTime	0.00
☐	ScanMode	1

Figure 11.2

To the left is checkboxes called Save Property. This means that the property in line with this checkbox will be included in the hardware config file. Some properties such as location and exposure should not be saved normally, but others like axis assignments , starting filter wheel locations, and COM ports should be saved here in the config file. Set those special properties to the correct values, check the save box of those properties that you wish to save and then click Next Device to move through the list. When you are finished with the setup, the files will be generated for this setup and then you will be able to see your hardware setup.



FrameRate (Sec/Frame) 10

Maximum FrameRate: 10
At Current Exposure

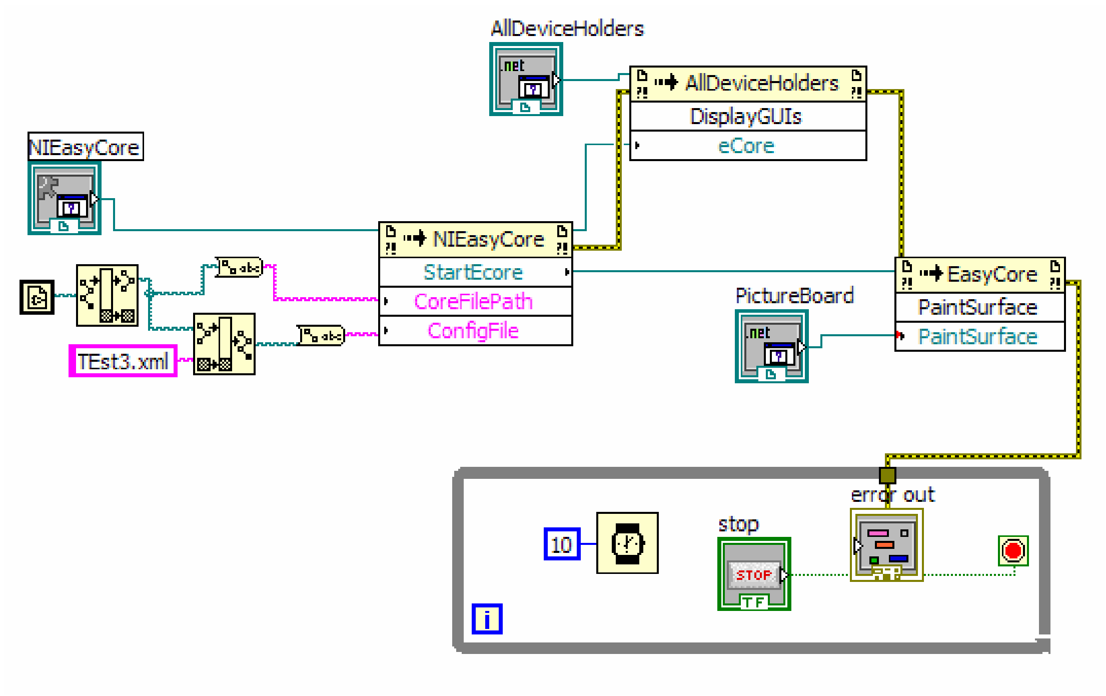
[✓] Autoset

Number of Frames (-1 for unlimited) 100

[ ] Save as Stack

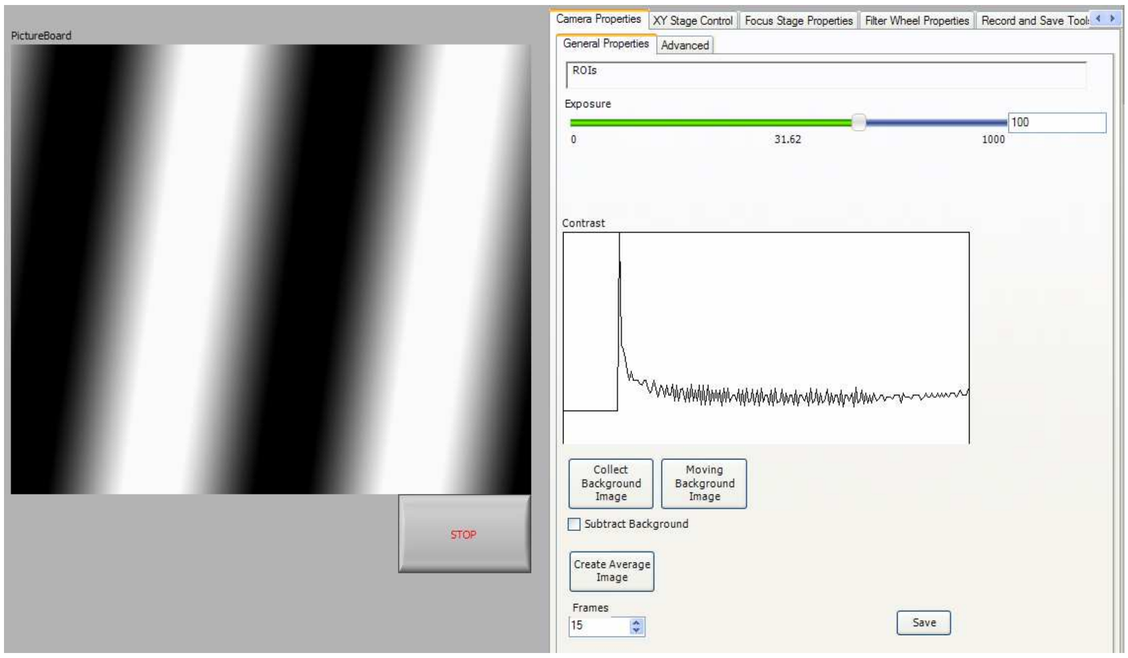
You already have a very capable microscope for use in the lab.

Now to reopen the config file, the procedure is extremely easy. It takes only three commands to be able to reach all the devices, channels, channel groups and recording possibilities of AutoMicromanager. Here is the figure.

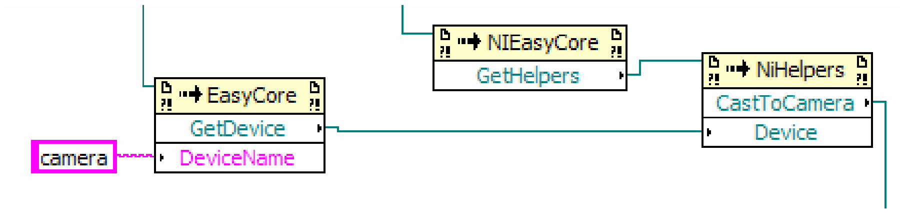


So StartEcore is started with a config file, the config file tells EasyCore how to put together the microscope. DisplayGUI puts together the FrontPage, and finally PaintSurface shows the images. The Record and Save tools manages the recording and saving as well as Channels and ChannelGroups. There are stages, joysticks and all the other tools needed.

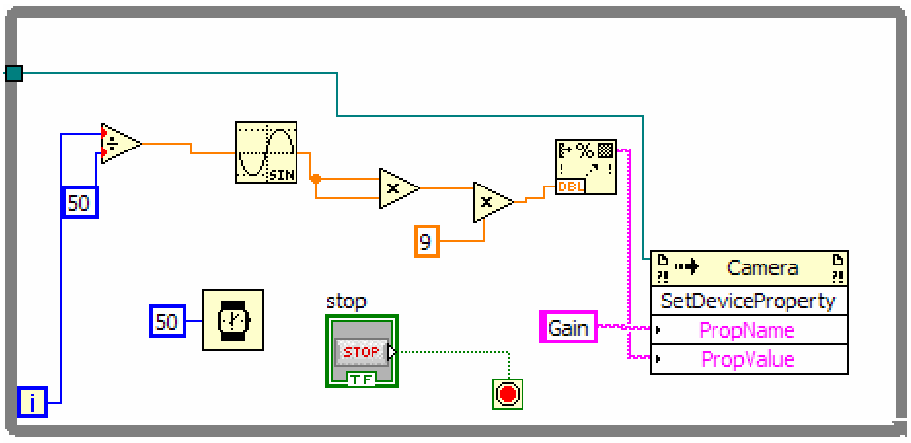
You may wish to have direct programmatic control of some part of the system. This is possible as EasyCore stays active on the MainVI. Here is a shot of the newly loaded interface.



The reason all the devices are named is so they can easily be accessed from EasyCore. If we wish to interact with the camera during the acquisition, this is possible by asking EasyCore for access to the camera.



This is achieved by GetDevice. You must enter the name exactly the way it has been loaded. If you are not sure of the names, you can find them in the hardware config file. Usually the camera is names "camera", so we collect the camera. Now, in order to use the power of the camera object, it must be converted from a generic device into a camera. GetHelpers provides a number of functions for all the common devices, so we used CastToCamera. Now a fully functional camera is available for use. Since the demo camera includes "Gain" as one of its properties, this is my target property and I actuate the gain with a squared sine wave.

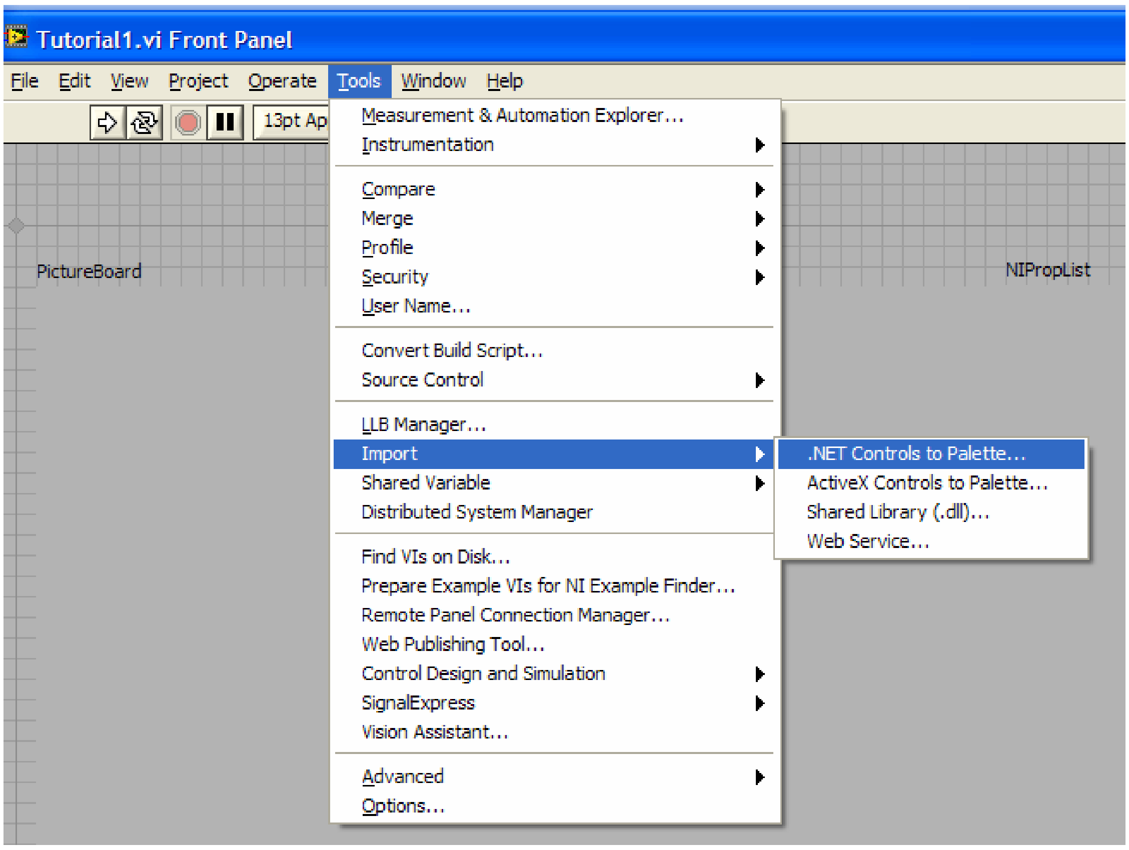


On the demo camera, this does nothing for the image, so you have to navigate to the property page in order to see the gain moving.

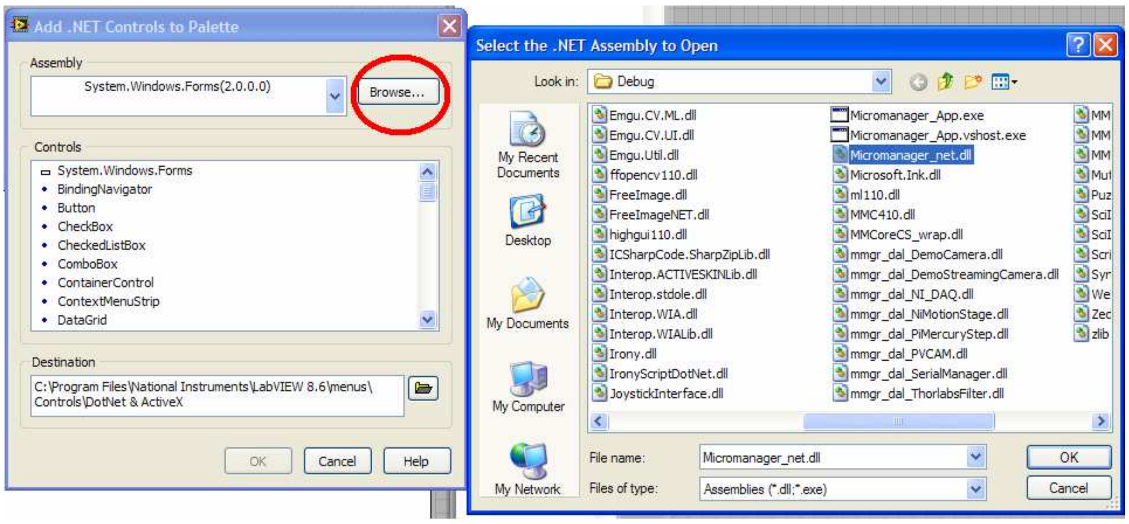
If your application requires image processing, the NIImageprocessor works the same as it did in chapter 8 allowing full control of the imaging processes.

Supplement

1. Adding AutoMicromanager tools to the Controls Menu manually.



Tools/Import/.Net Control to Palette on the upper menu (This menu sequence changes in some versions of Labview, but there is always an option to import a .Net control into Labview in the tools menu). This will bring up a dialog as show in figure 2


Add .NET Controls to Palette

Assembly: System.Windows.Forms(2.0.0.0)

Destination: C:\Program Files\National Instruments\LabVIEW 8.6\menus\Controls\DotNet & ActiveX

Select the .NET Assembly to Open

Look in: Debug

File name: Micromanager_net.dll

Files of type: Assemblies (*.dll;*.exe)

Figure 2.

Click the browse button to import a custom .Net library and then navigate to the folder where Micromanager_net.dll (“c:\program files\autoMicromanager\micromanager_net.dll”) is stored. This .dll is usually stored in the same directory as the directory where all the example Labview VIs are stored. Next, you will be able to select all the desired controls from the .Net library. You must double click on CoreDevices.NI_Controls to get to those controls. Select NIEasyCore all the controls under the NI_Controls tab as shown in Figure 3.

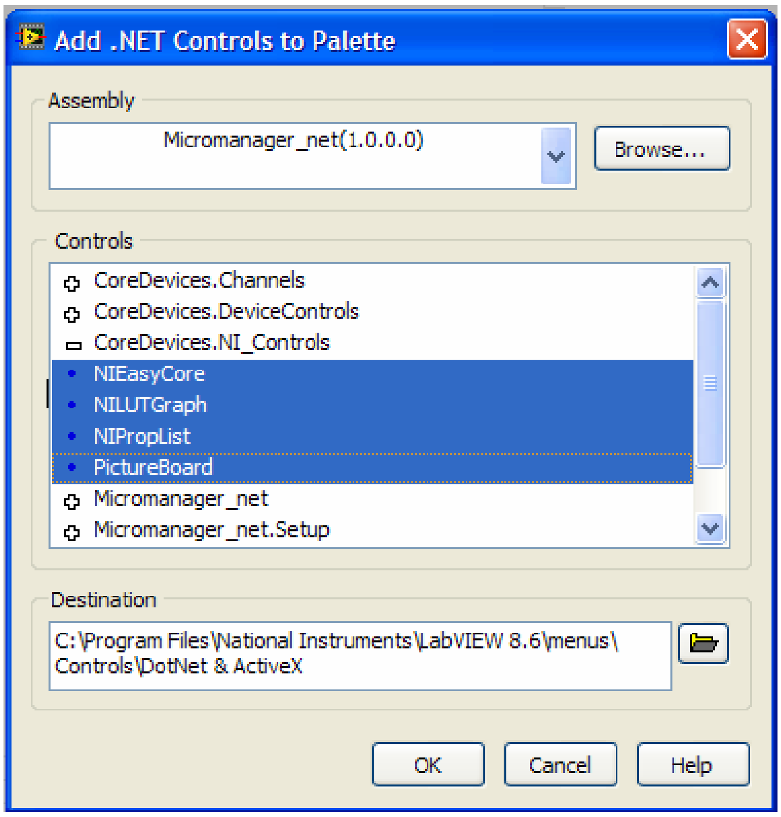


Figure 3.

The Labview program will load and convert these .Net controls to Labview controls and then you will end up back at the blank VI. Now you must right click on the front panel of the VI to get to the control menu.

Click the double down arrow on this menu to see all the options and you will see a submenu listed as **.Net & ActiveX.** Click this item and use the mouse cursor to discover the identity of each control. Place the NIEasyCore on the front panel as shown in figure 4. Do the same for the NILutGraph, NIPropList and PictureBoard controls which will be found in the same submenu. All .Net controls which are imported can be found in this submenu.

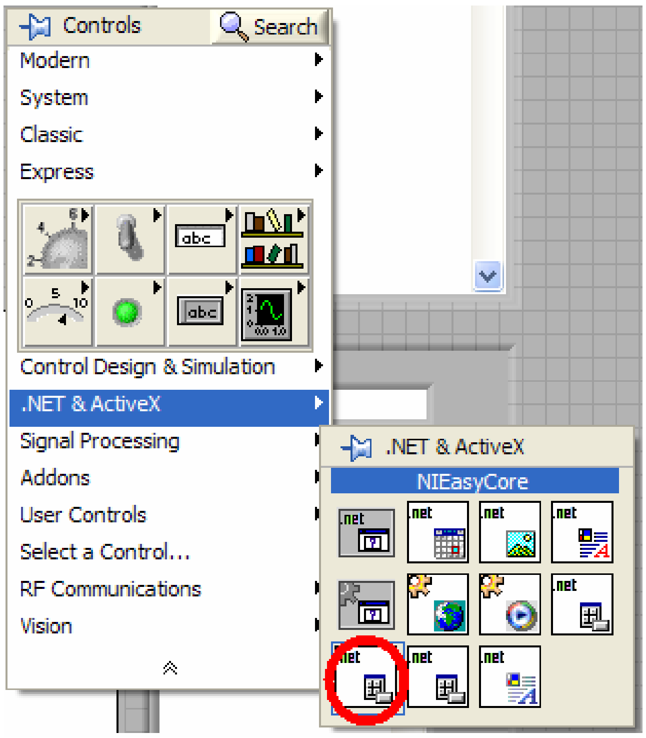


Figure 4