

Controlling EO-Drive controller via National Instruments LabView programming



Engineering Technical Support

If you require additional information, contact our technical experts for help.

24 Hour Phone Support

1-800-363-1992 or +1 856-547-3488

Sunday 8:00pm – Friday 6:00pm

Send e-mail to techsup@edmundoptics.com (during business hours)

Chapter 1: Introduction

This tutorial is designed to illustrate communication with an Edmund Optics EO-Drive. The EO-Drive is equipped with a USB digital interface, and communication is performed using LabVIEW on a computer with a Windows operating system. This tutorial will describe, in detail, how to create example LabVIEW programs (VIs) from scratch. The purpose is to help users understand how to communicate with the EO-Drive controller via USB, how to program certain useful motions, and how the example VI's are constructed. Completion of this tutorial will require the EO-Drive USB software has been installed. If this has been done, a folder exists on your hard drive containing EO-Drive software, manuals, and example VI's.

This tutorial is written for users with a very basic understanding of LabVIEW, but it is not intended to be a comprehensive tutorial for LabVIEW. National Instruments provides several excellent tutorials that should be used to learn about the LabVIEW programming environment. In addition, LabVIEW has excellent help documentation, that the user should become familiar. Each chapter is written as a stand-alone chapter so that readers with more than a very basic understanding of LabVIEW can skip ahead if desired. Each chapter incorporates and builds on ideas from the previous chapters. Note that the tutorial does not demonstrate how to produce every VI on the CD, but if you complete the tutorial, you should be able to build and incorporate any of the VI's provided.

LabVIEW can be thought of as a way to make a custom digital controller. This tutorial will illustrate how to set up LabVIEW VI's and how to use them to command your EO Drive and the objective lens nanopositioning system. It is our hope that after completing this tutorial, you will understand how our example LabVIEW VIs were constructed, how commands are passed from the computer's VI to the nanopositioner, and some of the common ways to move a nanopositioner. We also hope that after completing this tutorial you will have some ideas of how to modify these VIs to suit your own purposes.

Before discussing LabVIEW, there are several important concepts to discuss in order to make your LabVIEW programming tasks easier to understand.

Equipment List

To follow the steps in this tutorial, you will need the following:

1. Edmund Optics objective lens nanopositioning stage
2. Edmund Optics EO-Drive controller with USB interface
3. Windows computer with EO-Drive USB interface driver installed
4. National Instruments LabVIEW installed on your computer

Using a computer to communicate with the EO-Drive

The EO-Drive USB digital interface allows the user to communicate with the objective lens nanopositioner from a Windows based PC. To make this communication possible, we provide a dynamically linked library (DLL). A DLL is a compilation of functions that can be called from a Windows user mode program. A user mode program is an executable program that is written in a high level programming language such as C/C++, JAVA, Basic, LabVIEW, etc. LabVIEW

owes much of its versatility to the fact that it can be used to make function calls to any Windows DLL, and it has a specific procedure for doing this. These function calls are referred to in LabVIEW as a call library function, and the user should consider reading the excellent LabVIEW documentation on the subject. In this tutorial we will demonstrate explicitly how to use call library functions to interact with the EO-Drive.

Another important concept of the Windows operating system is the Handle. If you want to pick up your suitcase, you would grab it by the handle. In Windows, if you want to take hold of a device, you also want to grab it by the Handle. In Windows, the device Handle is really an address. In this tutorial, we will show you how to grab a device Handle for the EO-Drive. In addition, we will show you how to release the Handle when you are done communicating with the EO-Drive.

The final Windows operating system concept that is important for our customers is the issue of Windows latency. In user mode programs such as LabVIEW, the Windows operating system controls all interaction between your program and all other parts of the operating system, DLL's, hardware and other programs. User mode programs are not allowed to interact directly with the PC hardware. That includes all USB, RS232, GPIB or PCI bus devices. The operating system only allows your program to talk to hardware through a driver, and the driver can only access the hardware about every 1 millisecond. This means that you can only send or receive information from your hardware device, like the EO-Drive, every millisecond or so, and this is referred to as Windows latency.

How do you want to use your nanopositioner?

There is a diversity of applications for objective lens nanopositioning systems. We have produced general LabVIEW examples that meet the basic requirements of many customers, but none of the LabVIEW examples have been designed specifically to do your experiment. Examples are provided so that you can build more complex programs that are tailored to your exact requirements. We provide several examples for building a scanning program, but you will have to decide whether to use a saw tooth scan or a triangle scan for your application. In addition, you will have to make decisions about scan speed, scan accuracy and settling time.

What are the specifications of my nanopositioner?

The specifications of your system are written into the EEPROM of the USB interface, and can be accessed via a LabVIEW VI.

When you move the stage, you are really changing a command voltage with a DAC (digital to analog converter). This DAC has 16 bit resolution. If your system has a full range of motion of 100 microns and a 16 bit DAC, then the smallest step, S , you can perform is;

$$S = 100 \mu\text{m} / 65536 = 1.6 \text{ nm.}$$

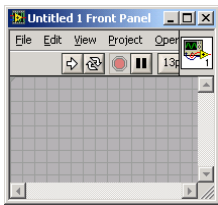
Where is the documentation for EO-Drive.dll?

The documentation for EO-Drive.dll is included on your distribution disk under the file name EO-Drive_1_0.doc, but it is also included in Appendix A of this document for quick reference.

Chapter 2: Introduction to the LabVIEW Programming Environment

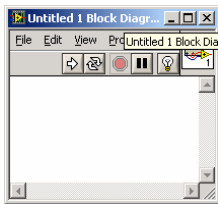
The goal of this Chapter is to introduce the structure of LabVIEW programs, and to review some basic LabVIEW concepts which must be understood before you can create programs of your own. Subsequent chapters demonstrate using LabVIEW to control an Edmund Optics EO-Drive. Users already familiar with LabVIEW may wish to skip ahead to Chapter 3.

LabVIEW programs are often created to imitate the function and appearance of a physical instrument. A LabVIEW program, called a VI (Virtual Instrument), consists of two separate windows, the Front Panel and Block Diagram.



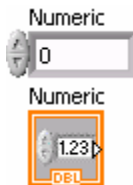
The Front Panel

Every new VI is automatically created with a Front Panel. The Front Panel contains the controls and indicators which allow the user to manipulate the operation of the VI. Any Control or Indicator created here is also automatically placed on your Block Diagram.



Block Diagram

Every new VI is automatically created with a Block Diagram. The Block Diagram contains the code of the VI. The values of Front Panel components can be read and assessed within the Block Diagram. The Block Diagram can also pass data to the Front Panel to be displayed on graphical indicators. Think of the Block Diagram as the guts of your virtual instrument.

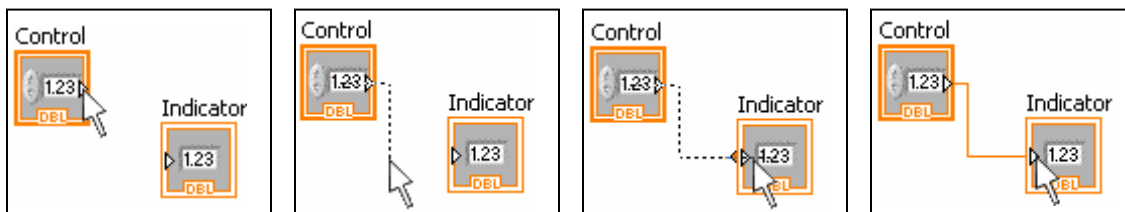


Controls and Indicators

Any parameters of the LabVIEW program that must change according to the user's specifications should be editable via the Front Panel. This is accomplished by first creating a Control on the Front Panel, then using the Control within operations on the Block Diagram. Indicators are used to transfer info from the Block Diagram to a display visible from the Front Panel.

Wire Connections

After a component of the Block Diagram performs an operation, any data it has generated can be passed to other Block Diagram components via wire connections. Therefore, the Block Diagram will contain a mixture of components interconnected with wires. The sequence of images below demonstrates the method of creating wire connections in LabVIEW:



Adding Components to the Front Panel/Block Diagram

Controls and Indicators are examples of components that can be added to either the Block Diagram or the Front Panel of the LabVIEW VI. However, many components exist which can be added to the Block Diagram but not the Front Panel, and vice versa. LabVIEW allows the programmer to add *Controls* to the Front Panel, and *Functions* to the Block Diagram. Therefore, things that can be added to the Front Panel are found within the Controls Palette (including indicators), and things that can be added to the Block Diagram can be found within the Functions Palette.

The Controls Palette

The Controls palette is available only on the front panel and contains the controls and indicators you use to customize the Front Panel. This makes sense because you are trying to create a virtual instrument, and the knobs, displays and switches always go on the Front Panel.

The Functions Palette

The Functions palette is available only from the block diagram. The Functions Palette contains the SubVIs and functions you use to build the block diagram. The basic palette (depending on the level of your LabVIEW license) contains a wealth of preprogrammed functions which can be used to construct your virtual instrument.

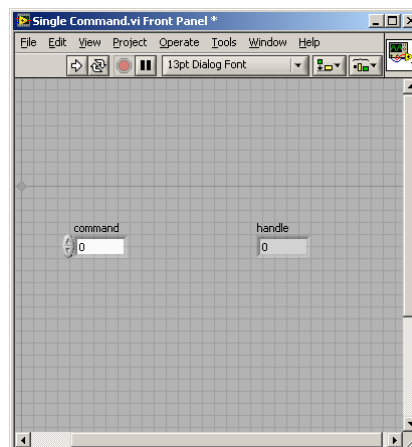
Chapter 3: Creating a LabVIEW VI to Control an EO-Drive Axis

You will be creating a LabVIEW VI that commands a user entered value to the EO-Drive. The EO-Drive's USB interface interprets a write command as a position in microns; this value is then converted to a voltage by the on-board DAC (Digital to Analog Converter). The resolution of the position command is therefore determined by the underlying resolution of the DAC. Likewise, the smallest step size of your stage will be determined by the DAC resolution and the range of motion of your stage. Before and after changing the position, the VI waits 100 milliseconds to mimic performing additional tasks.

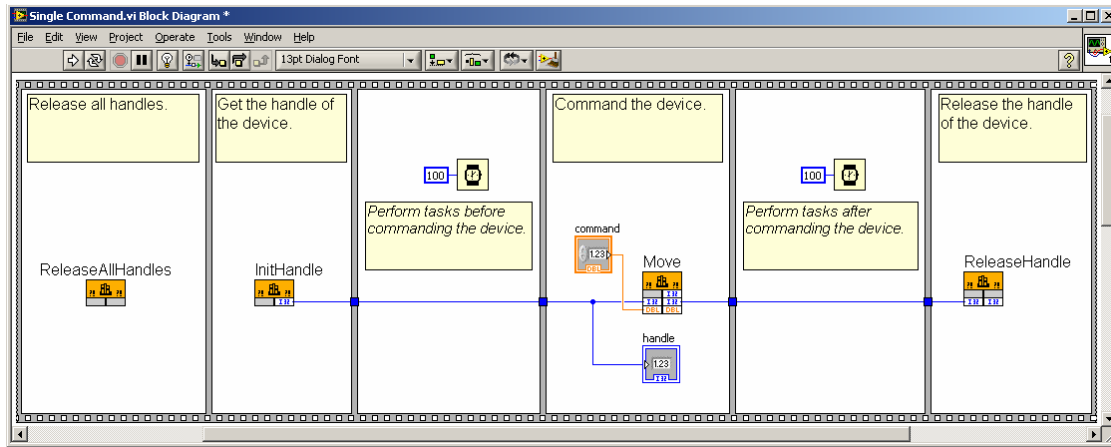
Topics Include:

1. Basic editing of the Front Panel of a LabVIEW VI.
2. Basic editing of the Block Diagram of a LabVIEW VI.
3. Interfacing with the EO-Drive using the EO-Drive.dll.
4. Referencing the EO-Drive_1_0.doc for function information (see Appendix A as well).
5. Using Structures, Timers, and other LabVIEW components to control dataflow.
6. Accessing functions within the EO-Drive.dll via Call Library Function Nodes.

The VI shown below will be created in Chapter 3:



(Image of the finished Front Panel by the end of Chapter 3)



(Image of the finished Block Diagram by the end of Chapter 3)

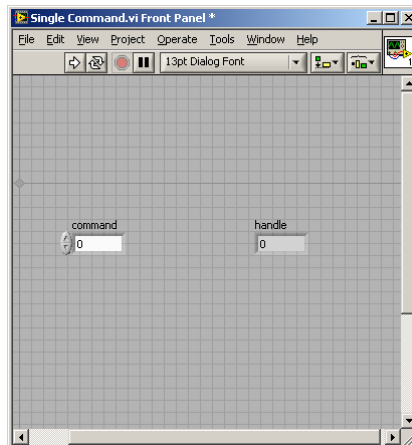
Section 3.1 - Preparing the Front Panel interface

Project Goal

In this section you will create a Front Panel containing all the necessary elements to make some simple EO-Drive commands. The Block Diagram will not be edited within this section. Complete all steps of this section to create the Front Panel depicted below.



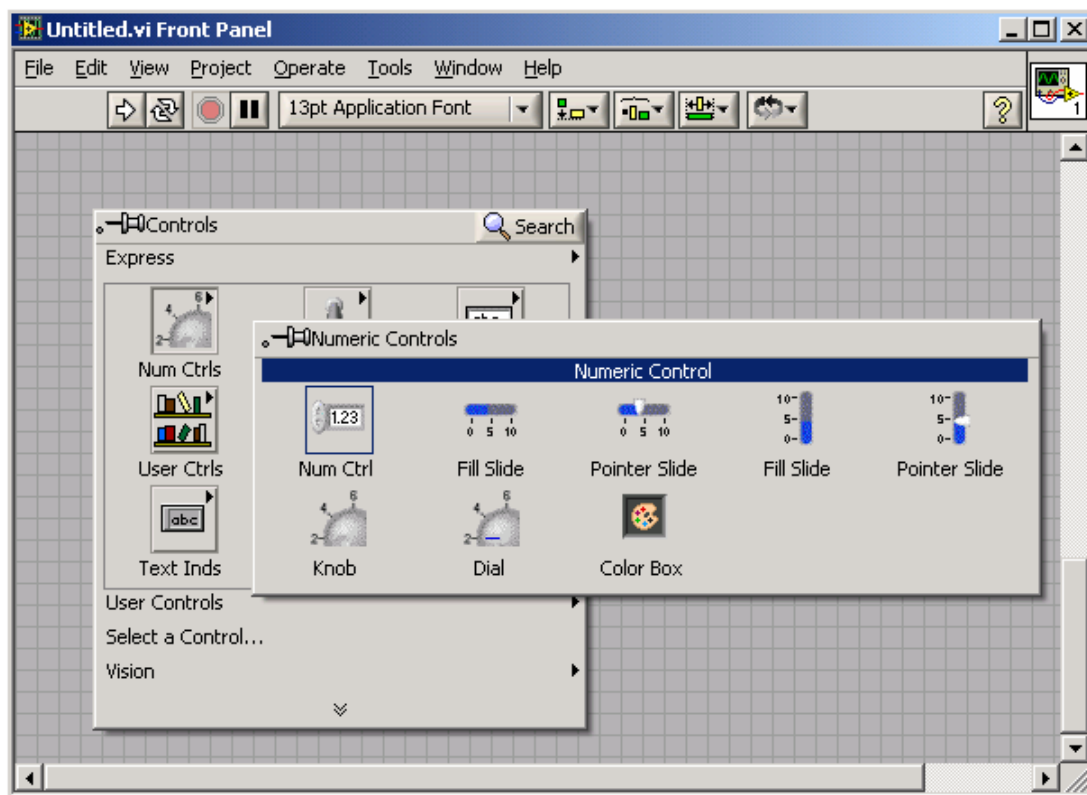
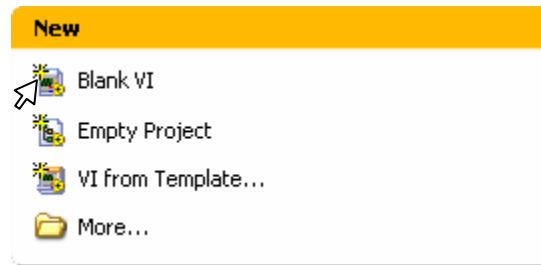
Note: Although the properties of the Numeric Controls/Indicators are modified within this section, the modifications have been limited to data type and label. Experienced LabVIEW users are encouraged to skip the intermediate steps and build the application according to the picture and description presented at the beginning of this section.



(Image of the finished Front Panel following completion of Section 3.1)

3.1.1 Start a new VI, and create the Numeric Control

- Begin LabVIEW and start a new VI by selecting *Blank VI* from the *Getting Started* Window. The Front panel and Block Diagram of a new VI appear.
- Right Click anywhere within the Front Panel area (seen below) to access the *Controls* menu.



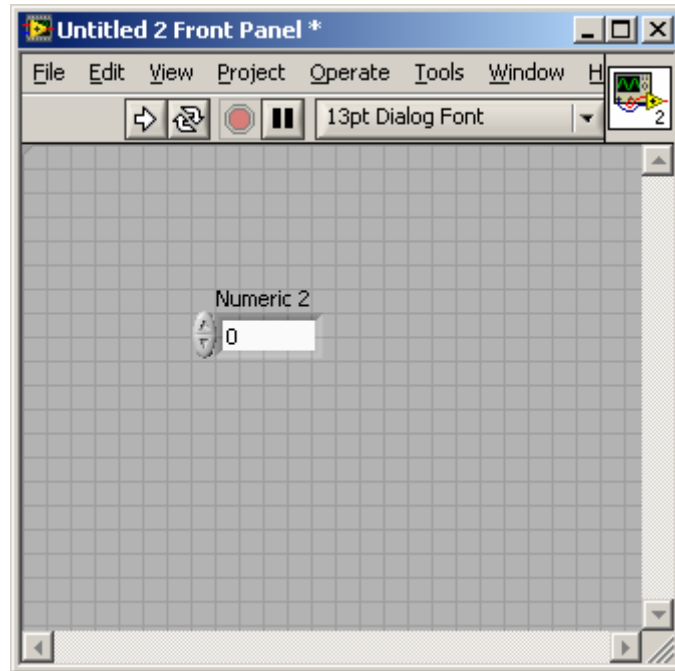
- Hover your cursor over the *Num Ctrl* button. The *Numeric Controls* window appears on top of the Controls window.



Note: If you're having trouble locating a specific control, left click the *Search* button to search by name via an automated search.

- Left click the *Num Ctrl* button that appears on the Numeric Controls window. The windows vanish and your cursor takes the shape of a hand. You are now holding the footprint of a Numeric Control which will later be used as your position control.

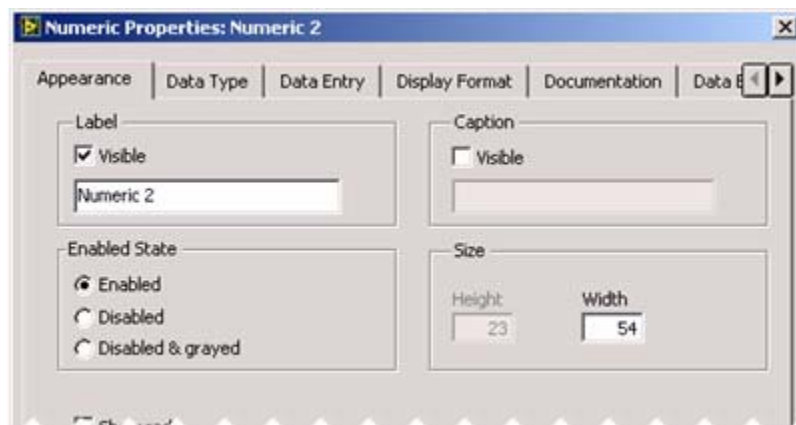
- Left click anywhere within the front panel to place the Numeric Control.



Note: The Numeric Control's value can be changed regardless of the state of the program (running/not running).

3.1.2 Editing Data Type

- Right click the Numeric Control and select *Properties* from the pop-up menu. The *Numeric Properties* window (seen below) opens into the *Appearance* tab.



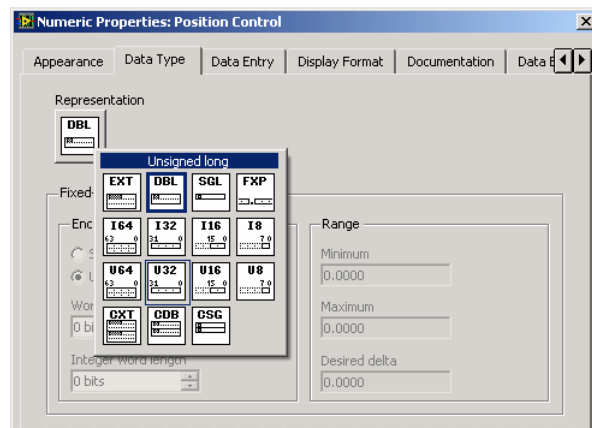
- This Control will be used to command the stage to a position, so change the text within the *Label* text field from "Numeric" to "command".
- Next, click the *Data Type* tab.

- Left click the data type representation button, and select *DBL* from the menu that appears.



Note: It's important to match the data type representation of the control to the data type expected by any functions using the control.

- Click the *OK* button near the bottom of the Numeric Properties menu to accept the changes and exit the menu.

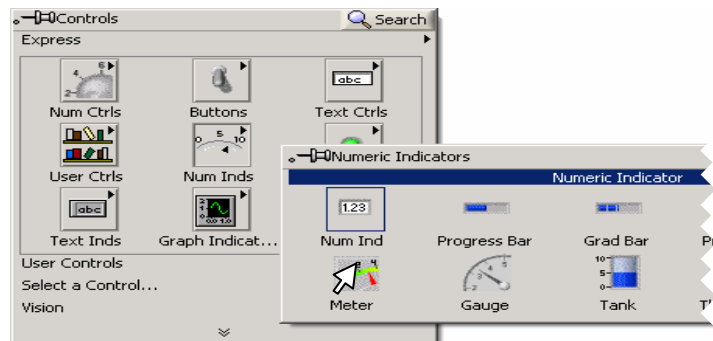


3.1.3 Create the two Numeric Indicators

Error Checking with Indicators

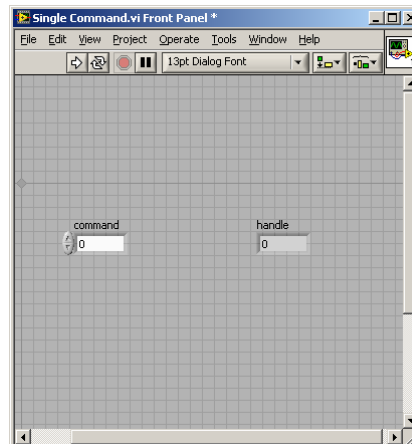
You will now create the *Numeric Indicator* used to display the handle. A handle must be acquired in order to communicate with the EO-Drive. Though it is not typically necessary to see the handle, beginning programmers should use the handle indicator as verification that the EO-Drive was loaded successfully.

- Right click over any empty area of the Front Panel and hover your mouse over the *Numeric Indicators* button. Two menus should be open now, the newest being the Numeric Indicators menu.
- Left click the upper-left most icon; the *Numeric Indicator*.
- Left click again to place the Numeric Indicator.
- Edit the properties of the indicator. Make its data type *I32*, and change its label to "handle".



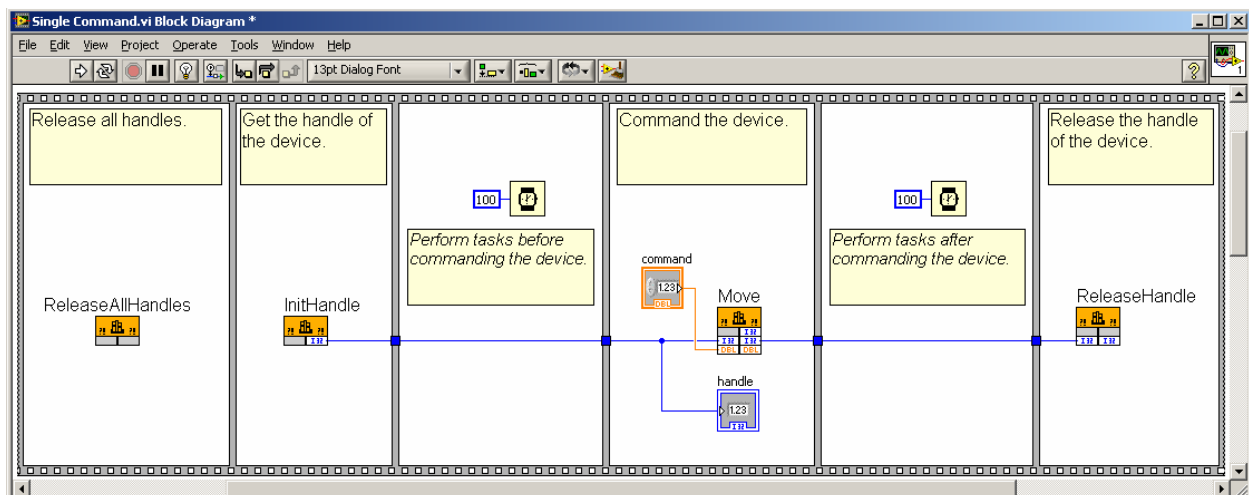
Section 3.1 Completed

We have completed the goal of Section 3.1. Our Front Panel now contains all the elements required to proceed into editing the Block Diagram. Also, the properties of all our controls and indicators have been set appropriately for their future applications.



Section 3.2 - Preparing the Block Diagram

By the end of this section, you will have a Block Diagram equal to the one depicted below. Once again, experienced LabVIEW users are encouraged to assemble the Block Diagram according to the picture. The dll function calls (yellow icons) appearing in this image are: (from left to right):
1. EO_ReleaseAllHandles **2. EO_InitHandle** **3. EO_Move** **4. EO_ReleaseHandle**.

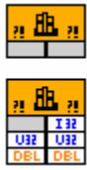


(Image of the finished Block Diagram following completion of Section 3.2)

Introduction to the Block Diagram

The Block Diagram of a LabVIEW VI contains the code to activate the Front Panel. The components and connections of the Block Diagram are essentially the inner-workings of the virtual machine. Code execution generally flows from left to right. In the image above, execution would begin with the dll function call to ReleaseAllHandles.

DLL (Dynamically-Linked Library) Function Calls in LabVIEW



The Call Library Function Nodes appearing in the image above, and at left, call functions from the EO-Drive.dll. Simply put, the EO-Drive.dll is a container of tools, or functions, used to communicate with the EO-Drive. In LabVIEW these functions are accessed via Call Library Function Nodes. At left are images of two separate Call Library Function Nodes. The lower one calls a function with input/return values. The user is encouraged to read the LabVIEW documentation for a more detailed explanation of Call Library Function Nodes.

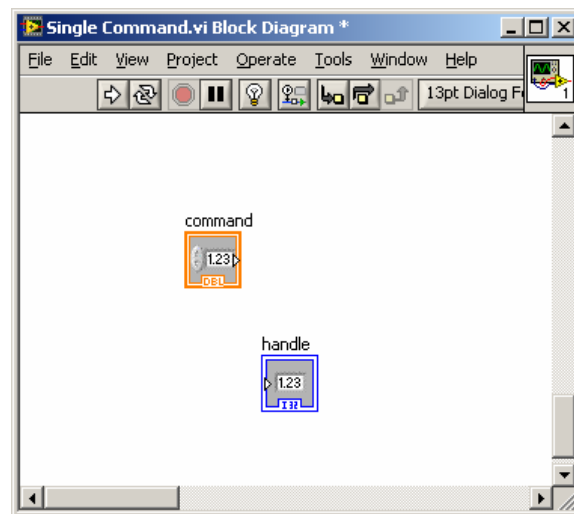
3.2.1 Accessing and Understanding the Block Diagram

- Hold *Ctrl* and press *E* to access the Block Diagram from the Front Panel. Alternatively, you can toggle between windows using the menu bar item: Window>>*Show Front Panel*.

Block Diagram: Controls and Indicators

Each time a control or indicator is created on the Front Panel its Block Diagram representation is created here. The two elements you see correspond to a Front Panel Control and Indicator of the same name which we created in the previous section (your icons may appear larger and more box-like; the difference is purely aesthetic).

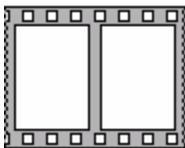
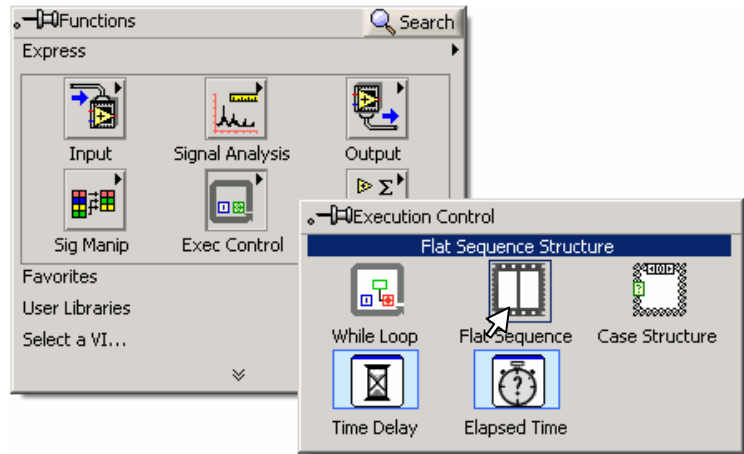
It is important that your Block Diagram matches the one depicted below. The arrangement, and icon appearance, of the components is irrelevant at this point. Pay particular attention to the data types (U32, I32, and DBL) and whether the arrow appears on the left side (Indicator) or right side (Control) of the component.



Note: Hold *Ctrl* then press *E* to switch back and forth between the Front Panel and Block Diagram. Alternatively, from the Front Panel menu bar, select: Window>>*Show Block Diagram*.

3.2.2 Create the Flat Sequence Structure

- Right click any empty area of the Block Diagram. The *Functions* menu appears.
- Hover the mouse over the *Exec Control* icon. A second window appears on top of the first.
- Left click the *Flat Sequence* icon. The windows vanish and your cursor is replaced with a box traced with a dashed line.
- Left click any empty space within your Block Diagram, move the mouse about four inches diagonally and left click again to finish creating a Flat Sequence Structure of a size equal to the box you just dragged (alternatively, you could have left clicked and held to create the structure. Upon releasing the left button the Flat Sequence Structure appears.)



The Flat Sequence Structure

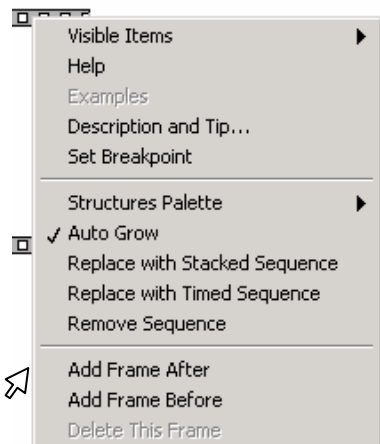
Flat Sequence Structures help the programmer to control the execution of the program. The execution flow of a frame of the Flat Sequence Structure is guaranteed to not trigger until the nearest preceding frame has completely finished.

Adding Components to the Flat Sequence Structure

Any component that was overlapped by the creation box has been added to the Flat Sequence Structure. Components can be drag-and-dropped in and out of the Flat Sequence Structure. Try putting some in, and then drag the Flat Sequence Structure around. You'll see that the components inside move with the box. However, if you drop the Flat Sequence Structure on top of outsiders; either the component, or the Flat Sequence Structure, will appear hovering above the other (try not to let this happen on your Block Diagram).

3.2.3 Add Frames to the Flat Sequence Structure

- Right click the top, or bottom, grey border of the Flat Sequence Structure.
- Select *Add Frame After* from the list that appears. A second frame appears after this one.



- The finished Block Diagram (as depicted in the image at the start of Chapter 3, Section 2) contained a Flat Sequence Structure with six frames. Continue adding frames to your Flat Sequence Structure until you reach six.
- Let's continue to match the image of the finished Block Diagram. Drag the 'command' Control into the fourth frame (the Flat Sequence Structure will auto-expand to fit the control if need be.)
- Drag the "Handle" Indicator to the fourth frame.
- Verify your diagram appears very similar to the image seen above. The numbered boxes distinguish what is meant by "frames" of the Flat Sequence Structure.

3.2.4 Add a Timer

Good Timing

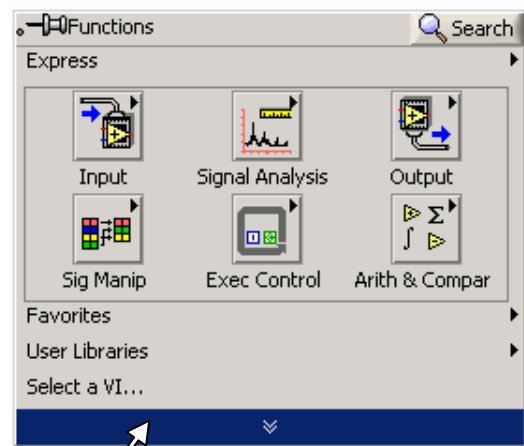
The diagram we are building makes use of Timers to synchronize data flow. In more complex programs, Timers are often placed before and/or after a command to control the command rate; however, this program uses the Timers only as a means of mimicking time spent performing tasks before and after the command. We will use a Millisecond Timer.

The *Wait (ms)* timer



The Millisecond Timer halts execution of the Block Diagram for a number of milliseconds equal to the numerical value passed to its input. This type of Timer requires an input. The input could come from a Numeric Control, allowing the user to alter the wait time from the Front Panel. However, to keep the example simple, this tutorial uses a Numerical Constant.

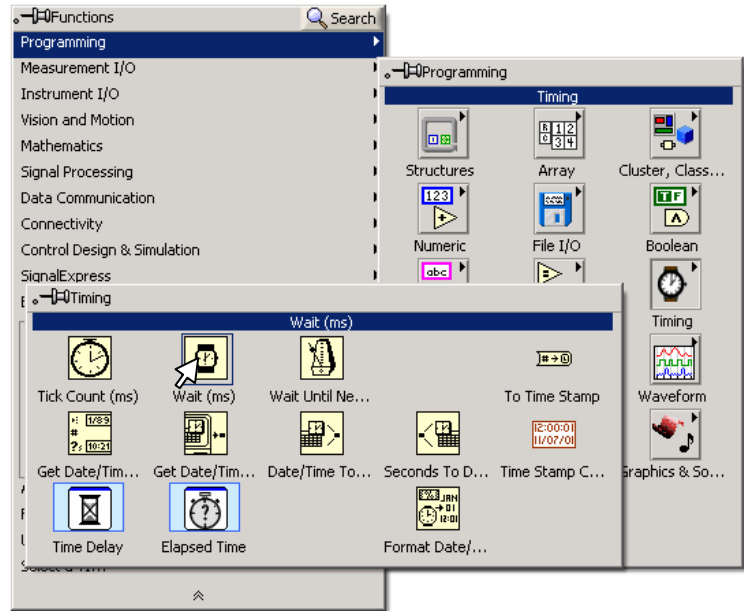
- Right click any empty area within the Block Diagram to open the *Functions* menu.
- Left click the bottom item of the menu (a double down arrow) to expand the menu. We can now see the full list of LabVIEW function categories.
- Hover your cursor over the *Programming* category. A new menu appears containing all sub-categories associated with programming.
- Hover your cursor over the *Timing* icon. A third window appears containing an assortment of timing functions.

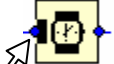


- Left click the *Wait (ms)* icon. The three windows vanish, and you should now be holding the icon for a timer.

- Bring the icon inside the third frame of the Flat Sequence Structure and left click to place the timer there.

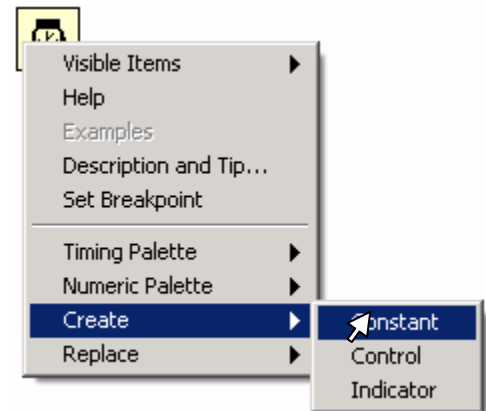
- Expand the third frame of the Flat Sequence Structure to add some working room. To expand the frame; bring your mouse cursor within the frame so that eight small resize boxes appear along the border. Left click and horizontally drag a left or right side box to your desired size.



- Hover your cursor over the Timer. The input (a blue dot) should appear on the left side of the Timer. 

- Hover your cursor over the input (left terminal) of the Timer and right click. A menu opens (as seen to the right).

- Hover your cursor over the *Create* category. This gives us a list of items to create pre-wired to the input of our Timer.



- Left click *Constant*. The windows vanish and a blue box containing the number zero will be created to the left of the Timer. This is a Numeric Constant, and it should be wired to the input of your Timer (if it isn't, wire it now. Refer to *Chapter 2: Wire Connections* for wiring assistance.)
- Left click once on the number '0' and replace it with the number "100". The third frame of your Flat Sequence Structure should now match the third frame as depicted in the image presented at the start of Chapter 3, Section 2.



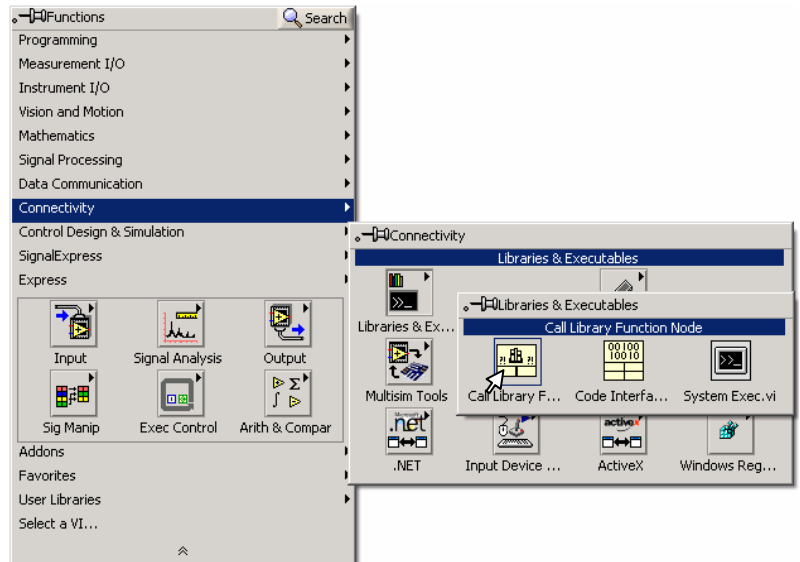
Note

Don't bother creating the second Timer at this point. Near the end of section 3.2 you will be asked to create the Timer based on the skills you've acquired.

3.2.5 Creating a Call Library Function Node

- Right click an empty area within your Block Diagram to access the Functions menu.
- Once again, left click the bottom item (a double down arrow) to expand the list of function categories.

- Hover your cursor over the *Connectivity* category. A new window opens.
- On the *Connectivity* menu, hover your mouse over the *Libraries and Executables* icon.




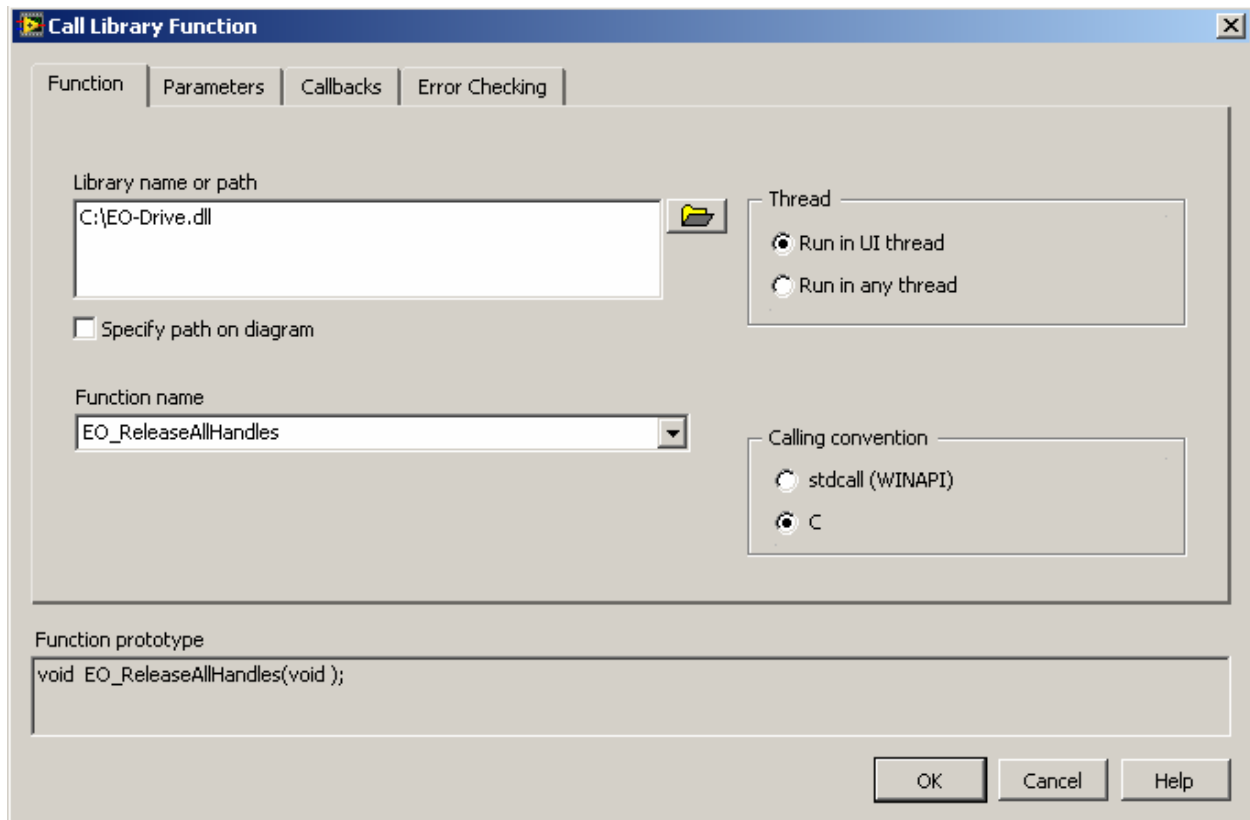
- Left click the *Call Library Function Node*. The windows vanish and you are now holding a footprint.

- Place the  Call Library Function Node within the First frame of the Flat Sequence Structure.

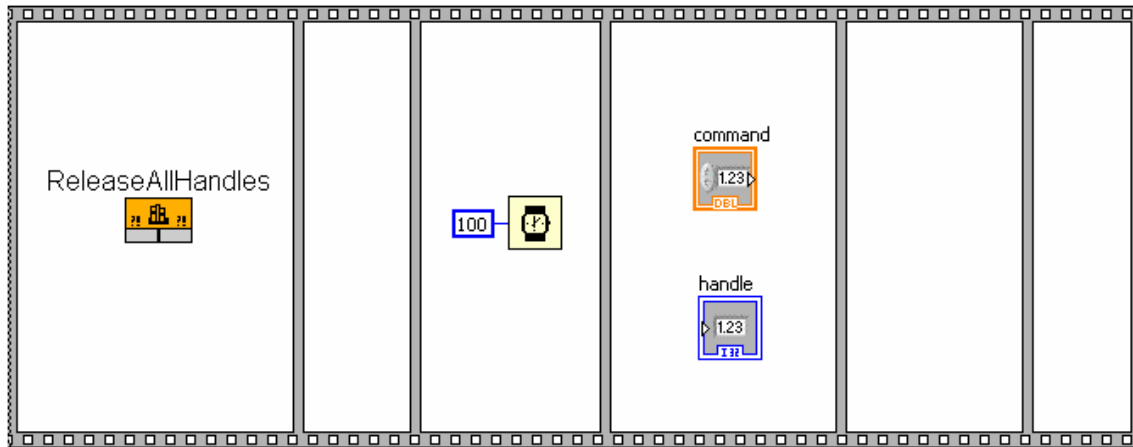
Call Library Function Node: Configuring to release all Handles.

To communicate with the EO-Drive we must be able to identify the EO-Drive by its Handle. Before we can identify the EO-Drive by its Handle we must first tell the dll to acquire a Handle for the EO-Drive. You are now configuring an EO-Drive.dll function which instructs the dll to release any Handles it currently has. This is an important task to do before ending the VI. Failure to release a Handle may cause problems for VI's used later that attempt to communicate with the same EO-Drive.

- Right click the newly created component, and select *Configure* from the menu that appears (or double-left click the component). Within the next few steps, we will set your Call Library Function Node to match the one seen below.
- Left click the *browse* icon. 
- Locate the file "EO-Drive.dll". The file was automatically installed onto your hard drive when you first installed the EO-Drive software.
- With the EO-Drive.dll selected as the Library to access, the *Function name* list has been automatically populated with all the usable functions. Left click the arrow at the right of the list to display all available functions.



- Select *EO_ReleaseAllHandles* from the list. The *Function* tab of your Call Library Function Node should now exactly match the image above.
- The function, *EO_ReleaseAllHandles*, doesn't require that we add or edit any parameters. All other tabs should be left at their default settings. Left click *OK* to accept the changes we've made.
- A dialog box will appear to inform you that the parameters will be automatically set to void. Click *OK*. This Call Library Function Node is complete.
- Verify that your Block Diagram closely resembles the one depicted below.



Note

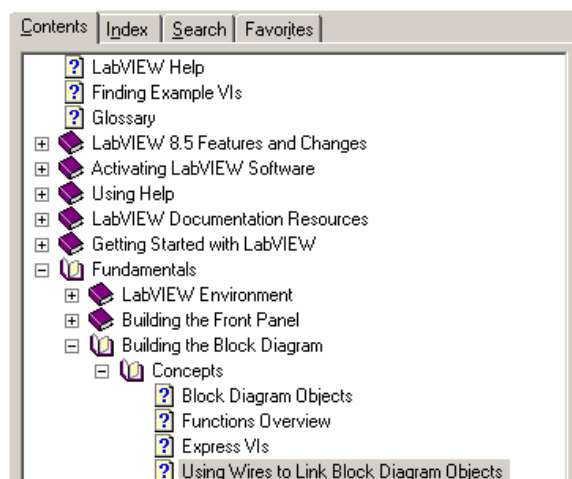
Your Call Library Function Nodes may be set to a more compact display type than seen above. To display the name of the function being called, right click the Call Library Function Node and hover your mouse over the *Name Format* category. Here you can toggle between *Names*, or *No names*.

3.2.6 Acquiring a Handle

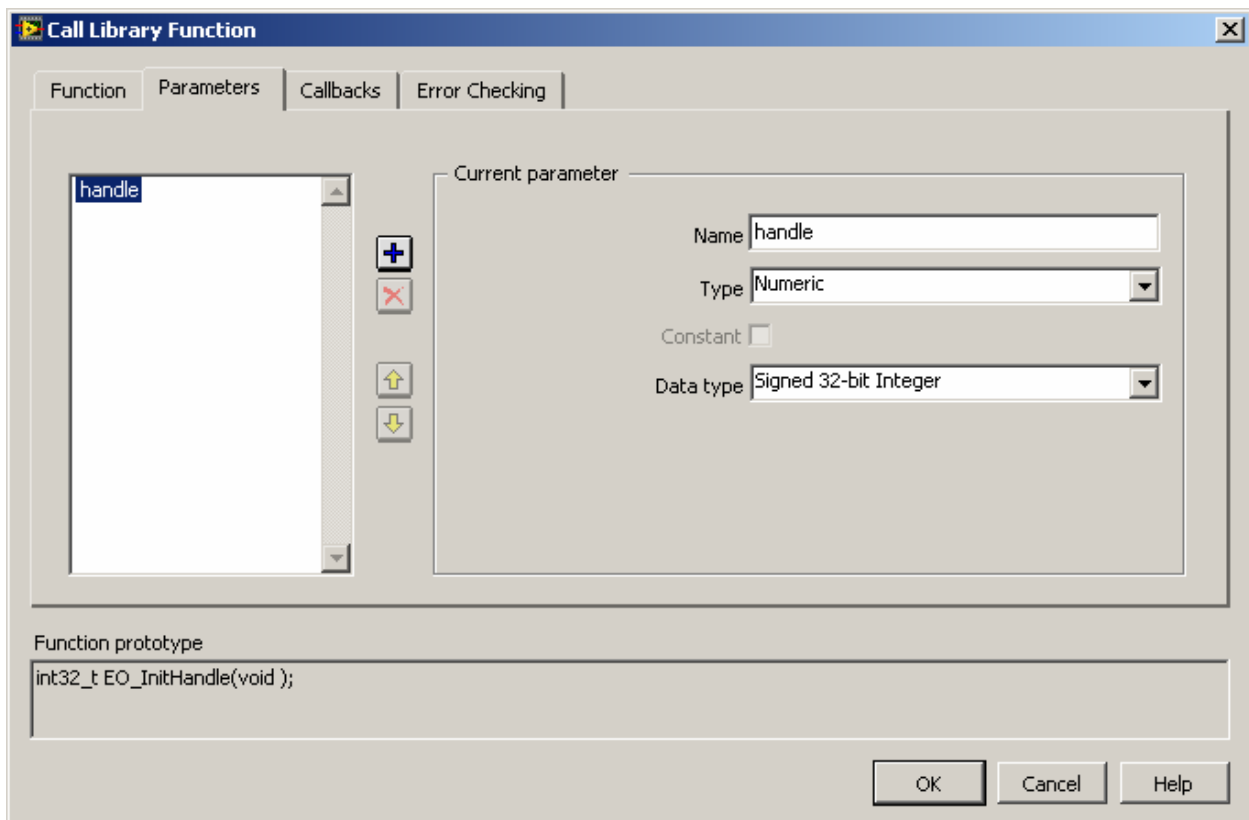
You will now create the Call Library Function Node which calls a function to acquire a *Handle* for your EO-Drive. By the end of this section you will have a functioning VI that will, among other things, grab a Handle for your EO-Drive and display it on the Front Panel via the “Handle” Numeric Indicator.

Accessing the LabVIEW Help Document on Wire Connections

You will soon be creating wires to connect the components you’ve created. This would be an excellent time to review the LabVIEW Help documentation regarding wire connections. LabVIEW Help documents can be accessed via the *Help>>Search the LabVIEW Help...* menu item. The image below demonstrates locating a document by navigating the *Contents* tab of the LabVIEW Help viewer.



- Create another Call Library Function Node either by repeating the steps described above, or by copying and pasting.
- Place it into the second frame of the Flat Sequence Structure.
- Enter its *Configure* menu.
- Direct the *Library name or path* to the EO-Drive.dll using the browse button, as done previously.
- Under *Function name*, select *EO_InitHandle*.
- Left click the *Parameters* tab at the top of the Configure menu.

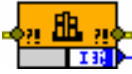


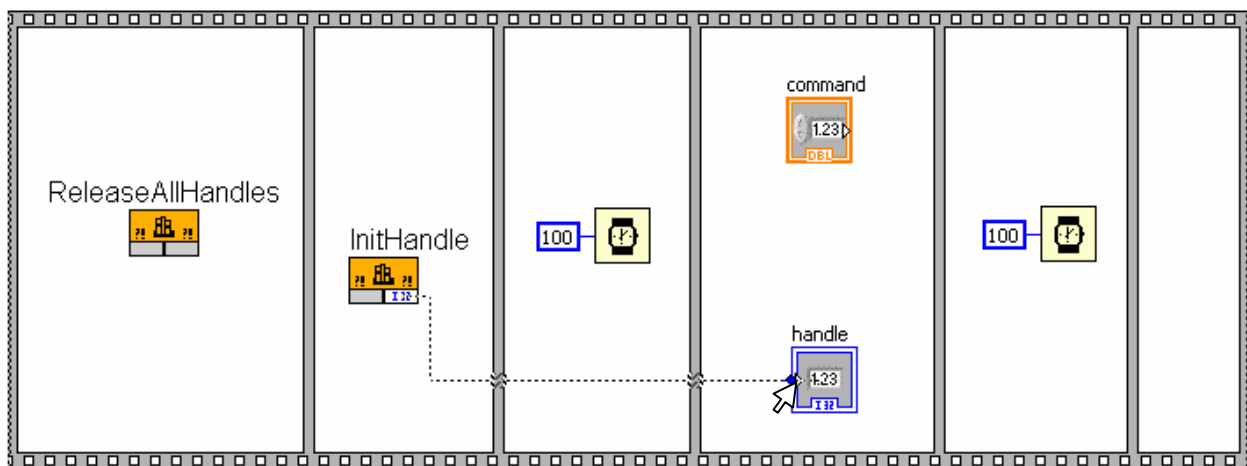
- In the *Current parameter* area, replace the name “return type” with the name “handle”.
- Change *Type* from “void” to “Numeric”. Two new settings should appear; *Constant*, and *Data Type*.
- Change Data Type to “Signed 32-bit Integer”.
- Left click *OK* to accept the changes.



Note

The appearance of this Call Library Function Node should have slightly changed. If the *Names* format is checked, the *handle* parameter now protrudes from the bottom of the Call Library Function Node. With the *No Names* format checked, an I32 has been added to the output (right) side of the Call Library Function Node (hover your cursor over the blue “I32” output and LabVIEW will show you the name.)

-  From the Block Diagram, left click the blue I32, or *Handle* output, on the EO_InitHandle Call Library Function Node. You are now dragging a wire out of this output.









- Left click the input of the “Handle” Numeric Indicator. A wire has been automatically created from the output of the EO_InitHandle function to the input of the Numeric Indicator.

Wire Connections and Tunneling


You’ll notice LabVIEW has a special way of bridging Flat Sequence Structure frames; a blue box has been created at every junction. The blue box is called a *tunnel*; and it allows data to flow in, out, and between, frames. To learn more about this, read the LabVIEW Help file on Flat Sequence Structures, or read the enclosed glossary definition for *tunnels*.

3.2.7 Running the VI

- Turn on the EO-Drive.
- Switch back to the Front Panel (*Ctrl + E*.)
- In the top left corner of the Front Panel, there should be an unbroken *Run*  icon. If your *Run* icon appears broken , left click it to receive a list of errors in your program. LabVIEW will not allow the VI to run until all errors have been eliminated.

- Run the VI by left clicking the *Run* button. The VI runs for 100 milliseconds then. A number should now be present within the text field of the “Handle” Numeric Indicator.
- Switch to the Block Diagram of your VI. The Block Diagram offers a few additional ways to run the VI. These can be seen to the right of the *Run* icon.
- Left click the *Highlight Execution* icon . The bulb within the icon should light representing that your VI will now run in Highlight Execution mode.
- From the Block Diagram, and with Highlight Execution on, left click the Run icon to run the VI. LabVIEW will now slowly step through every phase of your program and present to you a visual representation of what is being done.
- Try *single-stepping*  through the entire VI. Single-stepping puts you in complete control of the speed of execution, the VI only advances for each click of the single-step button. When you want to stop, left click the red stop button.
- Make sure your VI is stopped and Highlight Execution is turned off before continuing. If your VI is running the Run icon is a solid black arrow . Press the Stop icon to stop execution . If Highlight Execution is on, its toolbar icon will be a lit light bulb. Left click the icon to shut off Highlight Execution.

3.2.8 Writing to the EO-Drive DAC

- Create a Call Library Function Node.
- Place it within the fourth frame of the Flat Sequence Structure.
- Enter its Configure menu.
- Set the *Library name or path* to the location of the EO-Drive.dll.
- Set the *Function name* to “EO_Move.”
- Switch to the Parameters tab.
- Change the name of the “return type” parameter to “Error code.”
- Change the *Type* of the Error Code parameter to “Numeric.”
- Leave the *Data type* of the Error Code parameter as “Signed 32-bit Integer.”
- Left click the *Add a parameter*  button two times.

- Check the *Function prototype* of your Call Library Function Node. It will be located near the bottom of the Parameters tab of the Configure menu. It should read:

```
int32_t EO_Move(int32_t arg1, int32_t arg2);
```

Interpreting the Function Prototype

The function EO_Move is now configured to receive two arguments (arg1 and arg2), both of type “int32_t”, and will return an Error Code as a Signed 32-bit Integer. Compare this to the actual prototype for the function EO_Move as it appears within the EO-Drive_1_0.doc. You will notice that we are not finished configuring the function parameters...

- Change the name of “arg1” to “handle”.
- Leave the *Type* as Numeric, leave the *Data type* as Signed 32-bit Integer, and leave the *Pass* as Value.
- Change the name of “arg2” to “position”.
- Leave the *Type* as Numeric, but change the *Data type* to 8-byte Double.
- Leave the *Pass* as Value.
- Once again, check the Function Prototype. It should reflect the changes we’ve made. Compare this to the actual prototype of the EO_Move function as it appears within the EO-Drive_1_0.doc.

```
int EO_Move(int handle, double position)
```



(The EO_Move function as it appears within the EO-Drive_1_0.doc)

```
int32_t EO_Move(int32_t handle, double command);
```

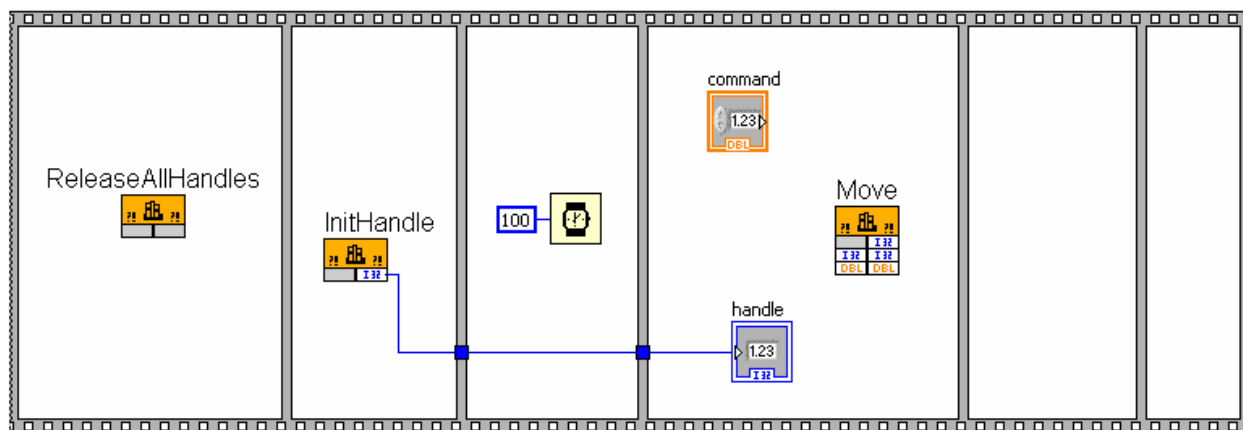
(Your EO_Move function as it appears in the Function prototype area of the Configure menu)

Interpreting the Function Prototype; Part 2

The declaration *int32_t* is equivalent to declaring the variable simply as *int* in most programming languages. Your Call Library Function Node is now properly configured to call the function EO_Move.

All parameters must appear in the exact order depicted above. If necessary, use the up  and down  arrow buttons to rearrange the order of the parameters.

- Left click *Ok* to accept the changes. The Call Library Function Node has become taller, compensating for the added parameters.




Note

Avoid congested Block Diagrams. Don't allow components to overlap wires, and make sure wire connections are clearly depicted as entering the proper input. Expand the frames of the Flat Sequence Structure if you need more room to work.

Interpreting the Function Prototype; Part 3

Unlike the Call Library Function Node for the EO_ReleaseAllHandles function, the EO_Move Call Library Function Node requires you to input data. Also, the function “returns” (outputs) an error code (or returns “0” if no error occurred).

3.2.9 Release the EO-Drive handle

- Create a Call Library Function Node.
- Place it within the sixth frame of the Flat Sequence Structure.
- Enter its Configure menu.
- Set the *Library name or path* to EO-Drive.dll.
- Set the *Function name* to “EO_ReleaseHandle.”
- Switch to the Parameters tab.
- Leave the “return type” parameter as “void”
- Left click the *Add a parameter*  button.
- Change the name of “arg1” to “handle.”
- Leave the *Type* as “Numeric”, and leave the *Data type* as “Signed 32-bit Integer.”
- Leave the *Pass* as “Value.”

Your function prototype should now appear as:

```
void EO_ReleaseHandle(int32_t handle);
```

- Left Click *Ok* to accept the changes.

3.2.10 Connect the wires, and create the second Timer

It is now time to create the remaining wire connections. The following instructions include every step necessary to accomplish this. However, if a wire is accidentally placed, or overlaps other components, refer to *Chapter 1: Wire Connections* for supplementary assistance.

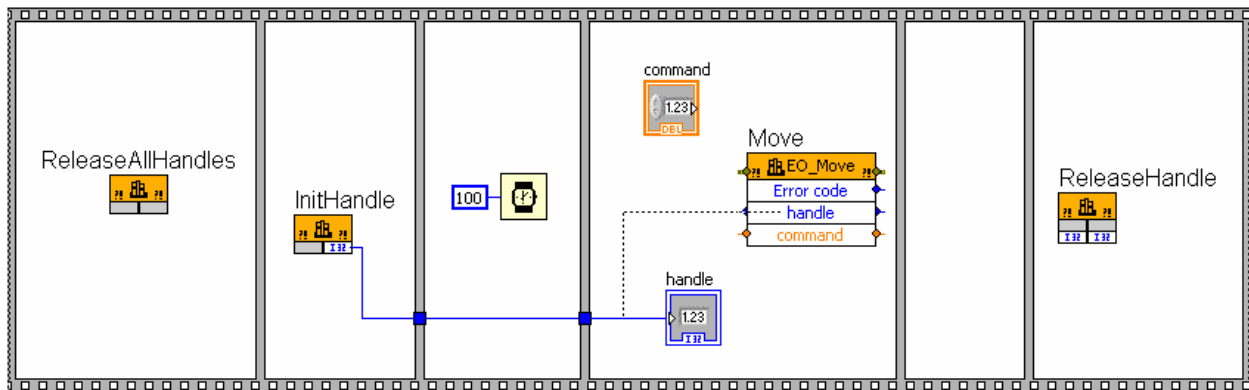
- Configure the EO_Move Call Library Function Node to display Names (right click the component and select *Names* in the *Name Format* category.)
- Expand the frames of your Flat Sequence Structure so that no component is within an inch (25mm) of an edge. This will give you enough room to make the wire connections.



Note

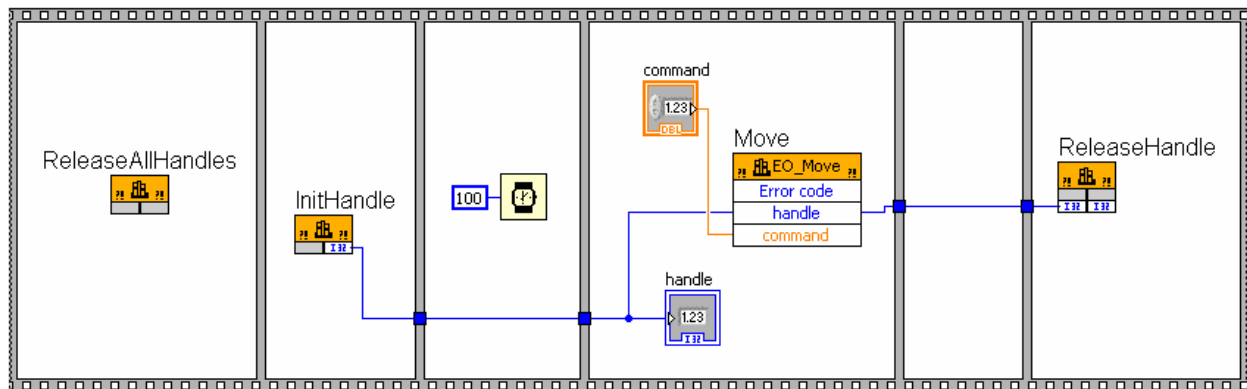
Make Coherent Wire Connections. After connecting a wire, left click to select a section of it. Vertically traveling wires can be moved horizontally, horizontal wires can be moved vertically. Wires can be selected and moved one pixel at a time with the arrow keys. Avoid overlapping components with your wires whenever possible.

- Left click the left side of the *handle* parameter on the EO_Move Call Library Function Node. You are now drawing a wire, connected at one end to the *Handle* input of the EO_Move function.



- Left click on the existing wire that passes into this frame. The wire passing into this frame comes from the EO_InitHandle function, and it is being passed to a Numeric Indicator. The EO_Move function now receives the same Handle as the Numeric Indicator.

- Connect the output of the “command” Numeric Control to the *command* input of EO_Move.
- Connect the *handle* output of EO_Move to the input (left) side of EO_ReleaseHandle. This instructs LabVIEW to pass the handle through EO_Move and into EO_ReleaseHandle. Alternatively, you could take the handle from before it is passed into EO_Move.



- Create another Timer that will halt execution for 100 milliseconds. Rather than copying and pasting the Timer into existence, recall how to create it via the functions palette. Place this Timer within the fifth frame of the Flat Sequence Structure (don't forget to set the number of milliseconds to wait.)

Section 3.3 – Command the EO-Drive

The Block Diagram is complete. Take a look at the *Run* arrow; it should be the normal image of an unbroken, white, arrow. However, if the Run arrow appears broken; the Block Diagram of this VI contains errors.

Troubleshooting a VI

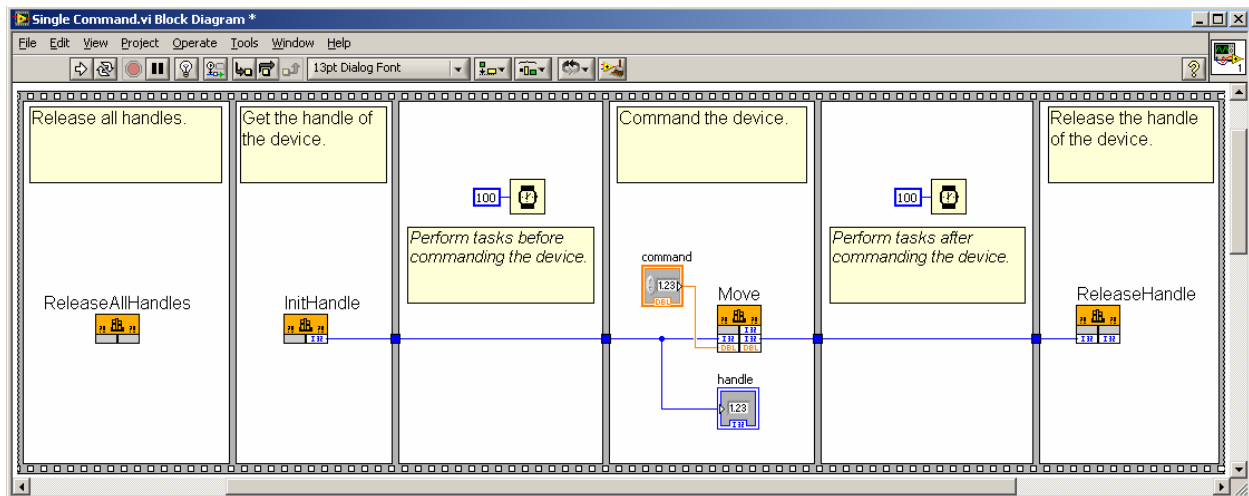
LabVIEW will not allow a VI to run while errors exist on the Block Diagram. Examples of such errors include an open/unconnected input or multiple outputs are wired together. Left click the broken Run arrow (seen at left) and LabVIEW will help you find the errors. If there is an open input, take a look at the image below to assist you in finding the correct connection. Also, this would be a good time to verify that all Indicators, Controls, and Function Parameters, are set to the correct Data Types.



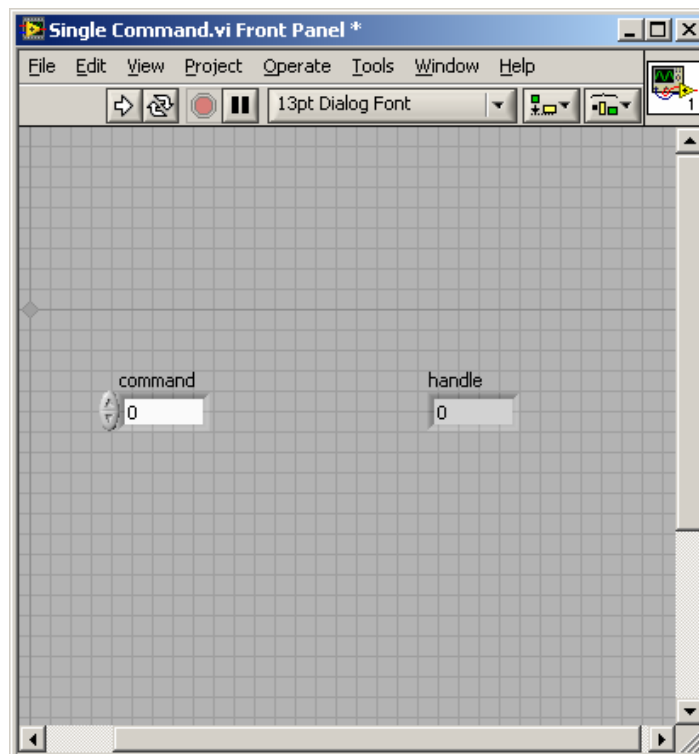


Note

Connecting together components of different Data Type Representations will not generate an error. Instead, LabVIEW automatically converts the data which is passed. Therefore, it is entirely up to you to verify that your Data Type Representations are correct.



(Image of the Finished Block Diagram)



(Image of the Completed VI's Front Panel after a single Run.)

3.3.1 *Enter Valid Command Parameters*

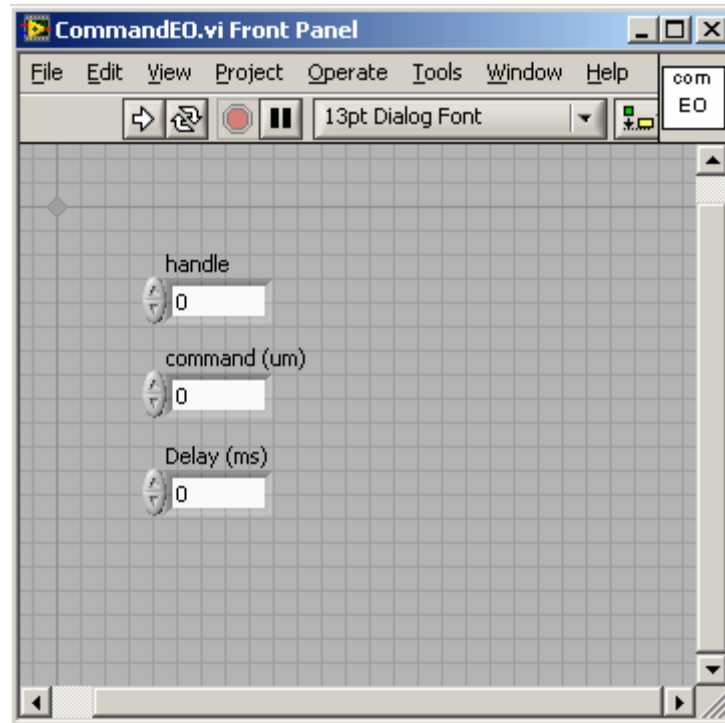
- Turn on the EO-Drive.
- Switch to the Front Panel of your VI.
- Enter a position to move to into the “command” control. Choose a micron value within the range of motion of your stage.
- Run the VI. The “Handle” indicator should show a non-negative number, and the stage should have moved to the specified position.

Project Completed

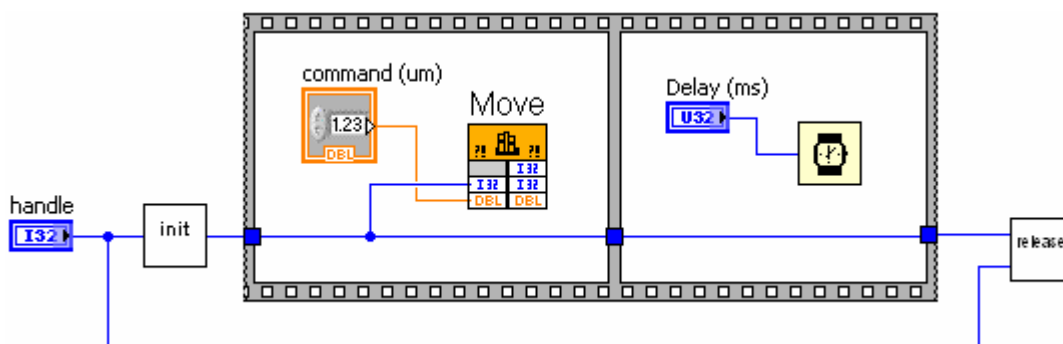
Completion of this Chapter has resulted in a VI that will write a position command to the EO-Drive. The following Chapters will build upon the skills acquired in this Chapter.

Chapter 4: Create a SubVI for Commanding the EO-Drive

The VI shown below will be created in Chapter 4.



(Front Panel of the finished Chapter 4 VI after a series of executions)



(Block Diagram of the Finished Chapter 4 VI)

Goal of Chapter 4

During this chapter, you will be instructed through the process of creating three distinct VIs, each of which will handle a separate portion of the same program. A few new LabVIEW concepts will be introduced along with several new LabVIEW components. The finished VI will accomplish a similar task as the VI created in Chapter 3; however, the emphasis here will be the use of SubVIs as a programming component.

Topics Include:

1. The use of SubVIs to simplify Block Diagram programming.
2. Editing the Connector Pane of the VI.
3. Responding to data using Case Structures and Comparison Functions.

EO-Drive.dll Functions Used Within This Chapter Include:

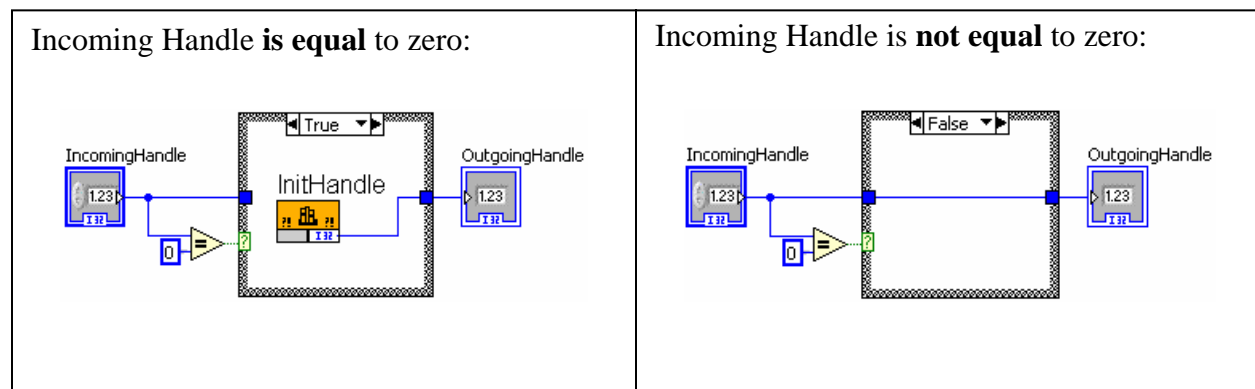
1. `void EO_ReleaseHandle(int handle);`
2. `int EO_InitHandle();`
3. `int EO_Move(int handle, double command);`

SubVIs Used Within This Chapter Include:

1. StandardInit.vi (*new*)
2. StandardRelease.vi (*new*)

Section 4.1 – Creating the Init SubVI

The two images below depict two different states of the same Block Diagram. This is the Block Diagram you will create in this section of the tutorial. It isn't necessary to see an image of the finished Front Panel; the Front Panel simply contains the "Incoming Handle" Numeric Control and "Outgoing Handle" Numeric Indicator.



Creating a SubVI to initialize the handle

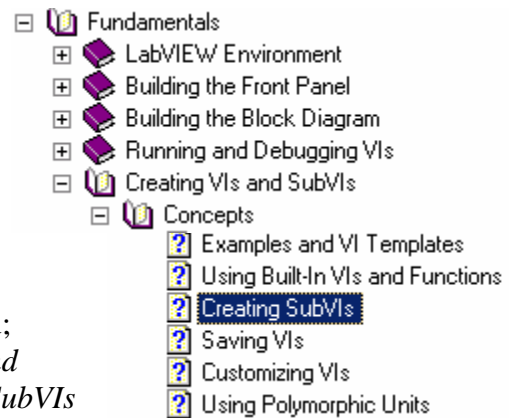
The "Incoming Handle" Control will receive an input from outside the VI. Similarly, the "Outgoing Handle" Indicator will pass data outside of the VI for use in other VIs. With this concept in mind, try interpreting what will happen when the VI is run. The function of this VI will be explained in depth as you progress through this section.

Recommended Reading: SubVI

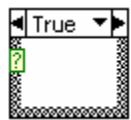
This VI will act as a SubVI; this means that you will design it to work as a component within other VIs you create. The SubVI is an important concept; it is therefore recommended that you read the LabVIEW Help document on SubVIs.

Accessing LabVIEW Help documents on SubVIs

To access the LabVIEW Help document on Creating SubVIs: Left click the *Help* Menu item at the top of your LabVIEW window; select *Search the LabVIEW help...* from the window that appears; (the LabVIEW Help window should open) select the *Contents* tab from the left-most menu; expand the *Fundamentals* folder, expand the *Creating VIs and SubVIs* folder, expand the *Concepts* folder. Select *Creating SubVIs* from the list. The document will display in a window to the right.



The Case Structure (Block Diagram)

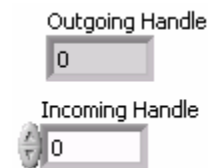


The *Case Structure* (seen at left) resembles a Flat Sequence Structure in both appearance and operation. However, the frames of the Case Structure are inseparably stacked one on top of another. Consequently, only one frame, or “case”, is visible at a time on the Block Diagram.

The key to the Case Structure is that only one of its frames (referred to as *cases*) will execute, the rest remain inactive for the duration of the containing VI/loop. Which case (frame) runs depends entirely on the value passed to the *Selector Terminal* (which appears on the left side of the structure). The Case Structure therefore allows the programmer to answer a question asked on the Block Diagram.


4.1.1 Create the Numeric Control and Indicator.

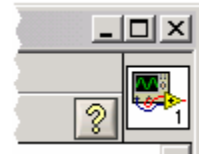
- Begin LabVIEW and start a new VI by selecting *Blank VI* from the *Getting Started* Window. The Front panel and Block Diagram of a new VI appear.
- On the Front Panel, create a Numeric Control.
- Change its data type representation to I32 (signed 32-bit integer).
- Change its label to “Incoming Handle”.
- Create a Numeric Indicator.
- Change its data type representation to I32.
- Change its label to “Outgoing Handle.”



4.1.2 Create the Terminals of the VI

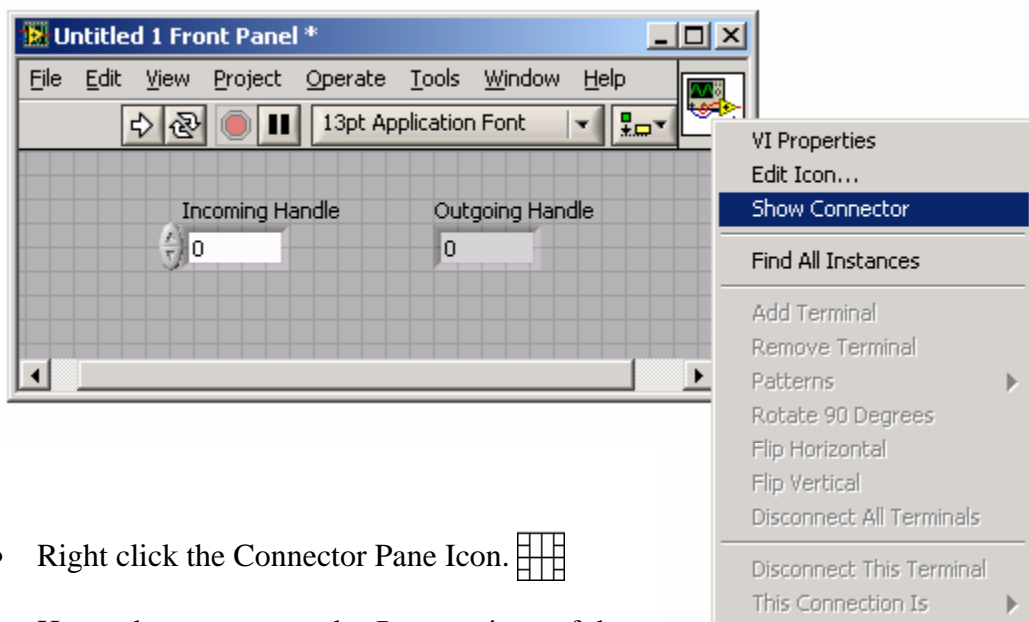
Every time you create a new VI, LabVIEW automatically prepares it to be used as a SubVI. Therefore, your VI has already been assigned a Block Diagram icon and a set of input/output terminals. So far these input and output terminals do not connect to anything within your VI.


- Right Click the icon  that appears in the top-right corner of your Front Panel. This is the default icon for a new VI.
- Select *Show Connector* from the menu that appears (see image below). The Icon should switch from the graphic of an oscilloscope to a group of boxes. This is the Connector Pane of your VI.

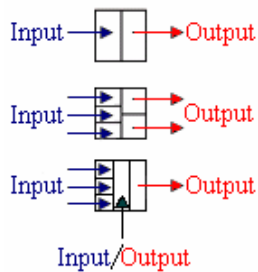
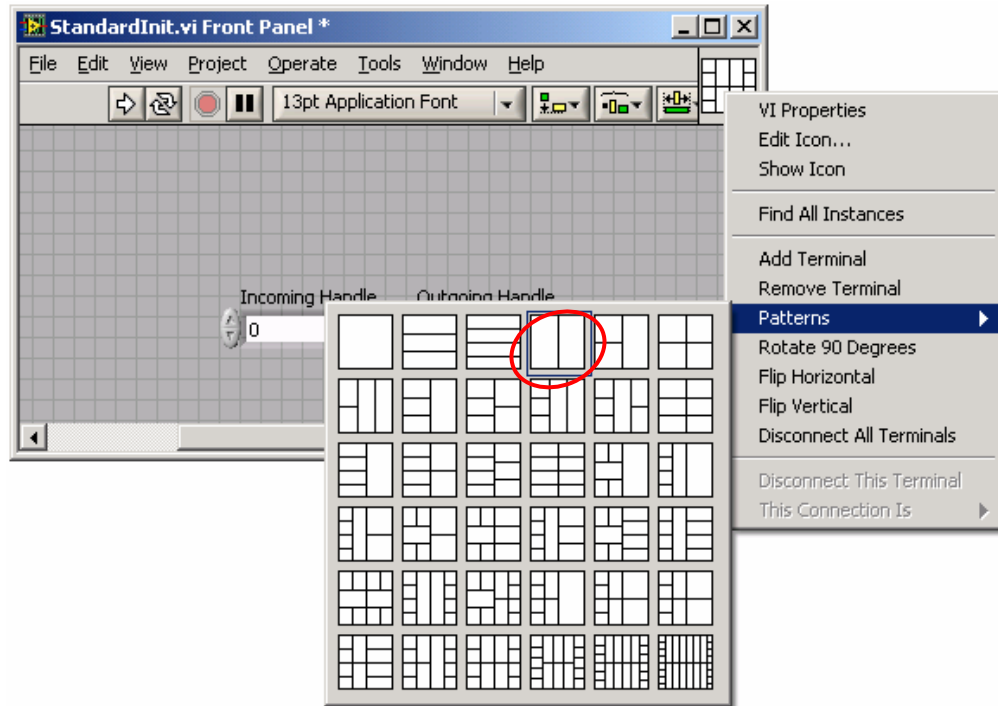


Note

You cannot Show Connectors from the Block Diagram. You will not have access to the options shown in the image below unless you are on the Front Panel.



- Right click the Connector Pane Icon. 
- Hover the cursor over the *Patterns* item of the menu that appears (seen below.)
- Select the fourth pattern from the left on the top row. In the image below, a red circle has been added to indicate which pattern to choose.

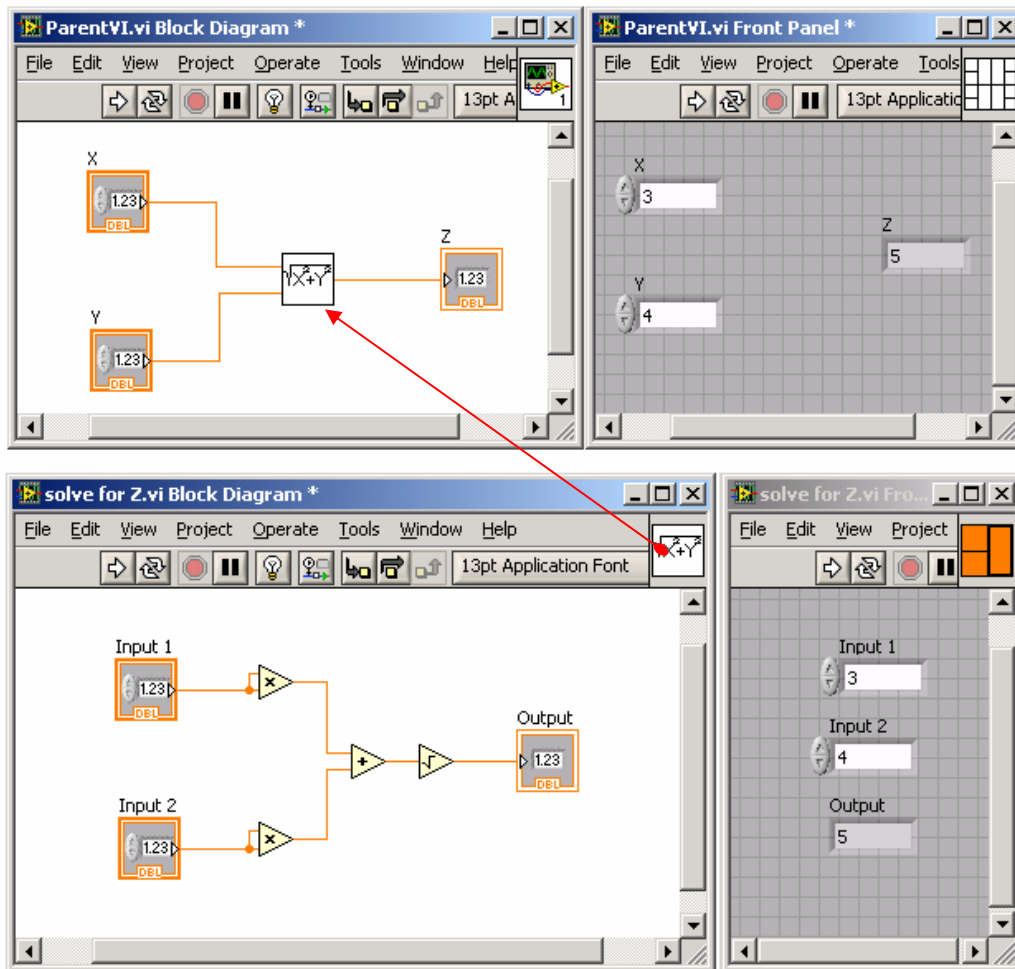


The Connector Pane

The Connector Pane specifies the input and output terminals of the VI you're creating. If this concept sounds obscure, keep in mind that some VIs are meant to be used within other VIs as SubVIs. Using the Connector Pane is actually quite simple and intuitive, but requires an understanding of the underlying concept of the SubVI. If you haven't already read the National Instruments documentation on SubVIs, please refer to the beginning of this chapter for assistance locating these documents.

Example Input and Output on the Connector Pane

Below are images of two separate VIs; *ParentVI.vi*, and *solve for Z.vi*. Both VIs are configured to accomplish the same task; to compute the length of the vector, *Z*. However, as can be seen, *ParentVI.vi* does none of the math. Instead, it passes the value of its Numeric Controls (*X* and *Y*) to *solve for Z.vi*. *Solve for Z.vi* passes the data through the appropriate mathematical operators, then stores the computed value inside its Numeric Indicator (*Output*). The *Output* Indicator is connected to the output terminal of the *solve for Z.vi* Connector Pane, which allows its value to pass out of *solve for Z.vi*. *ParentVI.vi* then retrieves the value of the *Output* control and passes the value to its Numeric Indicator (*Z*.)



4.1.3 Set the input and output of the VI

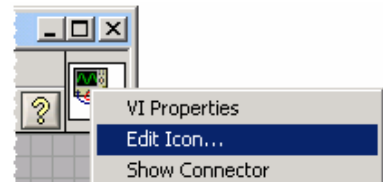
- Left click the input (left) half of your VI's Connector Pane. The side you click should turn black (as seen to the right.)
- Left click the Incoming Handle Control. The left half of your Connector Pane should turn blue (if it changes from black to a color other than blue, verify the Incoming Handle control is set to I32.)
- Left Click the output (right) half of your VI's Connector Pane.
- Left Click the Outgoing Handle Control. The right half of your Connector Pane should turn blue.




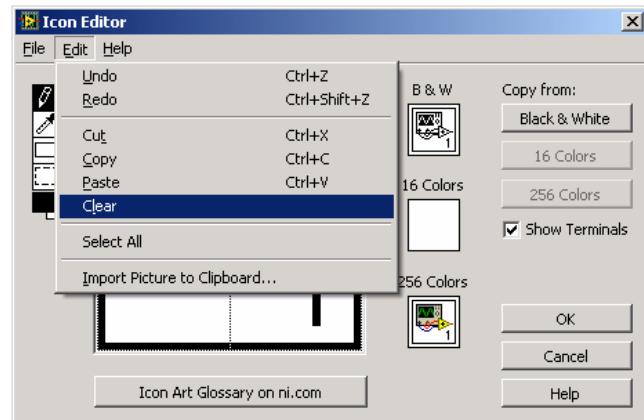
4.1.4 Edit the icon

This SubVI will appear within the Block Diagram of future VI's as an icon with input/output terminals. The icon of a SubVI should summarize its purpose; the purpose of this SubVI is to *initialize* a handle for use within other VI's.

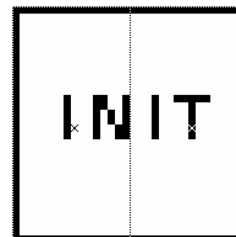
- Right click the Connector Pane icon.
- Select *Show Icon* from the menu that appears. The Connector Pane graphic should be replaced with the more familiar graphic of an instrument displaying a sine wave.
- Once again, right click the icon.
- Select *Edit Icon* from the menu that appears. The *Icon Editor* opens (as seen to the right.)



- Clear the image to white. This can be done either by painting white over the entire image, or by selecting *Clear* from the *Edit* menu.
- Select the  text tool from the toolbar on the left side of the Icon Editor.



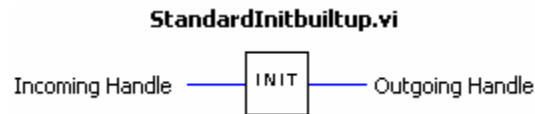
- Left click slightly left of the center in your icon. You can now apply text to this area.
- Enter: I N I T
- Draw a black border around the perimeter of the canvas using the pencil, line or unfilled square tool. Do not draw the border tightly around the text; it should encompass the entire drawing area.
- Check the *Show Terminals* option on the right side of the Icon Editor. Your image should appear very similar to the one seen at right.
- Click the *OK* button at the bottom-right of the Icon Editor.





Note

To view the inputs/outputs of a SubVI in a format similar to the image below, select *Help>>Show Context Help*, then left click the icon of the SubVI.




4.1.5 Configure a Call Library Function Node to Call *EO_InitHandle*

The Front Panel is finished. Switch to the Block Diagram (hold *Ctrl* and press *E*.)

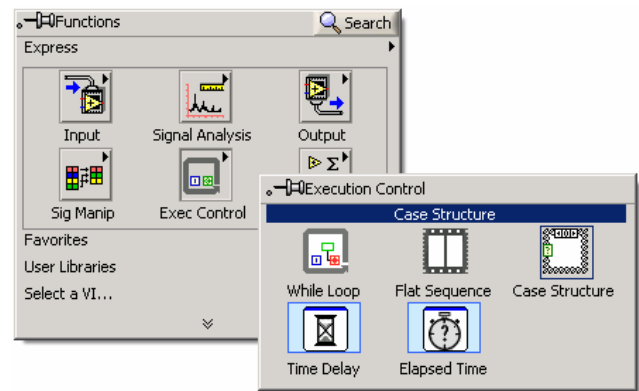
- From the Block Diagram, create a Call Library Function Node. Place the Call Library Function Node anywhere on the Block Diagram.
- Enter its *Configure* menu.
- Direct the *Library name or path* to the EO-Drive.dll using the browse button.
- Under *Function name*, select *EO_InitHandle*.
- Left click the *Parameters* tab at the top of the Configure menu.
- In the *Current parameter* area, replace the name “return type” with the name “handle”.
- Change *Type* from “void” to “Numeric”. Two new settings should appear; *Constant*, and *Data Type*.
- Change *Data Type* to “Signed 32-bit Integer”.
- Left click *OK* to accept the changes.

4.1.6 Create the Case Structure

You will now be instructed to create a Case Structure. The Case Structure creation process will closely resemble creation of the Flat Sequence Structure.

- Right click over any empty area of the Block Diagram. The *Functions* menu appears.
- Hover your cursor over the *Exec Control* icon which appears within the *Express* submenu of the Functions menu.
- Left click the *Case Structure* icon.  The two menus vanish.

- Left click any empty area of the Block Diagram.
- Move your cursor approximately three inches (75mm) diagonally. Be careful to avoid overlapping any other Block Diagram components.
- Left click again to finish creating a Case Structure (alternatively, you could have left clicked and held the button to specify the initial size of the Case Structure.)




- Any Block Diagram components that were overlapped by the Case Structure at the moment of its creation have been automatically inserted into the Case Structure. If this has happened, drag them out of the Case Structure. The Case Structure should be empty at this point.

4.1.7 Create the Equal Comparison Function

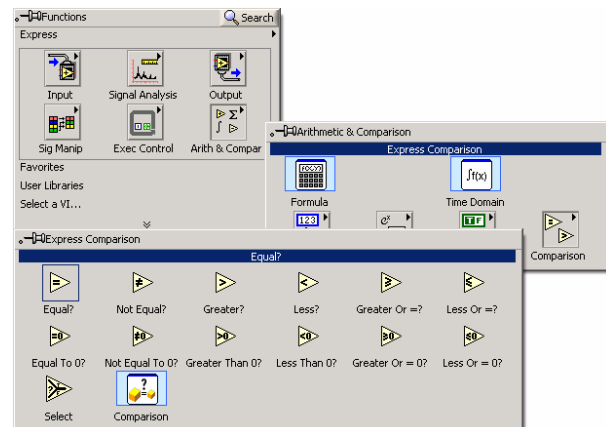
You will now create the Comparison function which is used to check for a valid handle.


The Equal? Comparison Function

The Equal? comparison function receives two separate data inputs, compares them, then outputs a value of True if the inputs are equal, or False if the inputs are unequal. Like every other LabVIEW function, the inputs are wired into the left side of the Equal? function, and the output is taken from the right side of the Equal? function.

- Right click an empty area of the Block Diagram. The Functions window appears.
- Hover the cursor over the *Arith & Compar* icon . The *Arithmetic & Comparison* window appears (seen below.)



- Hover your cursor over the *Comparison* icon. The *Express Comparison* window appears.



- Left click the *Equal?*  function. All Windows vanish and you are left holding an Equal? function.
- Left click an empty area to the left of the Case Structure to place the Equal? function.

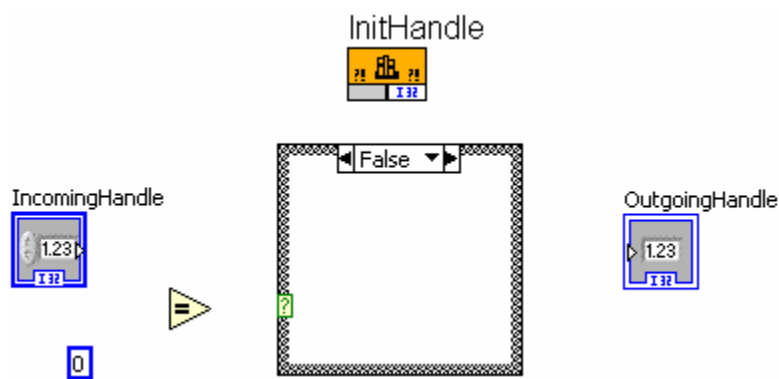
4.1.8 Create the Numeric Constant.

You may recall creating a Numeric Constant in the first chapter. However, the previous instructions followed a shortcut method for creating the Numeric Constant. The following instructions will find the Numeric Constant within the Functions menu.

- Right click any empty area of your Block Diagram. The Functions menu opens.
- Hover your cursor over the Arith & Compar icon. The Arithmetic & Comparison window appears.
- Hover your cursor over the *Numeric*  icon. The *Express Numeric* window opens.
- Left click the *Num Const*  icon. The windows vanish and you will be left holding a Numeric Constant.
- Left click an empty area of the Block Diagram that is to the left of the Equal? function. This places the Numeric Constant (with an initial value of “0”).

4.1.9 Arrange the Components to prepare for wiring

- Arrange the components so your Block Diagram appears similar to the image below.
- Leave the EO_InitHandle Call Library Function Node outside of the case structure for now.



4.1.10 Make the Comparison Wire Connections

It's now time to connect the inputs and output of the *Equal?* comparison function.


Using a Comparison function to control a Case Structure

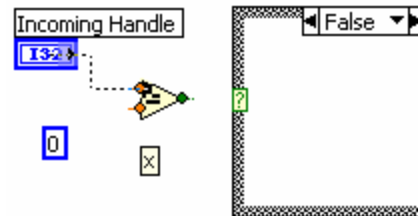
If this VI was used as a SubVI, the following could occur: a valid Handle is passed into the Incoming Handle Control from another VI. In this case, the Handle should be passed directly to the Outgoing Handle indicator and right back out of this VI. However, if no Handle is passed into this VI, an attempt will be made by this VI to get a Handle for the EO-Drive. How will this VI know when it has to acquire the Handle itself? Answer: if the Incoming Handle control is empty (i.e. the Incoming Handle control is equal to "0".)


- Left Click the output (right) side of the Incoming Handle control. You are now dragging a wire out of this control.
- Left click the top input of the *Equal?* comparison function. A wire has been created connecting the Incoming Handle control to this input.

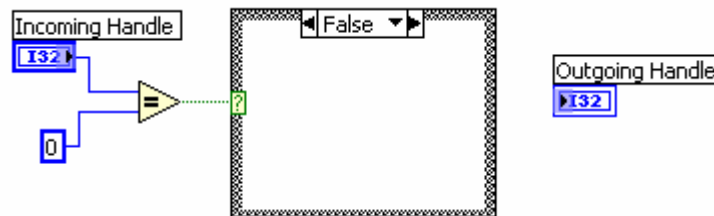
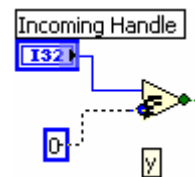


Note

The  in the image at right appears whenever the top input of a comparison function is highlighted. It's telling us that the top input is the X in the comparison equation: $is X = Y$?



- Left click the output of the Numeric Constant. You are now dragging a wire out of the Numeric Constant.
- Left click the bottom input of the *Equal?* comparison function to finish creating a wire between the Numeric Constant and the Y input of the comparison function.
- Left click the output (right) side of the *Equal?* comparison function. You are now dragging a wire out of this comparison function.
- Left click the Selector Terminal  of the Case Structure. LabVIEW automatically connects the wire to the input (left) side of Selector Terminal.

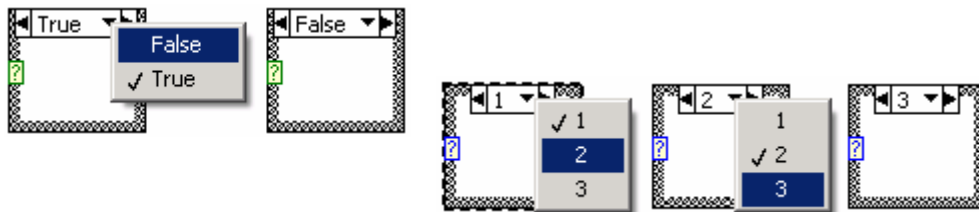


4.1.11 Make the Outgoing Handle wire connections

Upon creation, your Case Structure was automatically given two cases: True and False. Keep in mind that the title of these cases corresponds with the possible outputs of the Equal? function.

Understanding the Case Structure

Two separate Case Structures exist among the five images of cases below. The cases titled True, and False, represent one entire Case Structure; the cases titled 1, 2, and 3, are the second entire Case Structure.



Cycling through cases (frames) of a Case Structure

On the Block Diagram only one case of a Case Structure is visible at a time. To view other cases belonging to the same Case Structure; left click the downward pointing arrow at right of the case title. This action pulls down a list of existing cases. Left click the title of a case to view it.

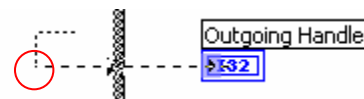
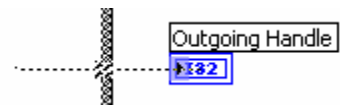
- Switch to the False case of your Case Structure.



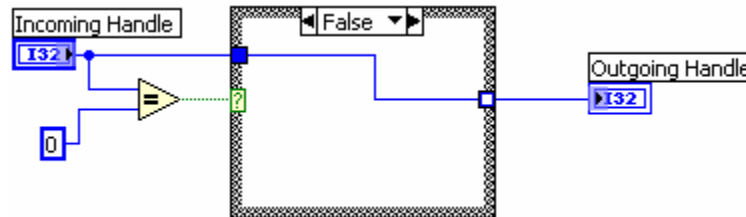
Note

If you find that your Case Structure is missing the True and/or False case, or other cases exist than the True and False cases; refer to the LabVIEW help file on Case Structures for help editing/deleting cases.



- Left Click the input (left) side of your Outgoing Handle indicator. You are now dragging a wire out of this indicator.
- Drag the wire into the Case Structure. You'll see the prototype of a *Tunnel* appear where the wire enters the Case Structure.
- Left click an empty area within the Case Structure. You're still dragging the wire, but you'll notice it has been pinned down at the point where you clicked. A red circle has been added to the image at right to illustrate where the wire was pinned.
- Drag the wire out the left edge of the Case Structure. Another Tunnel prototype appears where the wire exits the Case Structure.



- Left click the output of the Incoming Handle control. A wire has been created connecting the Incoming Handle control to the Outgoing Handle indicator through the False case of the Case Structure.



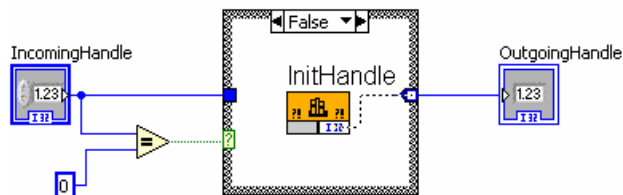
Case Structure Outputs

All Structure inputs/outputs pass through Tunnels , which often take on a distinct color according to the type of data passing through. However, the Tunnel passing the Incoming Handle wire through the output (right) side of the Case Structure is filled with white representing that it is incomplete: . This occurred because an output was also created in the True case of the Case Structure. At this point, the output in the True case isn't configured, and so, it can't be used.

4.1.12 Prepare the True case of the Case Structure



You will now create the code that will execute in the event that the value, “true”, is passed to the Case Structure Selector Terminal.


- Switch to the True case of your Case Structure. The wire passing through the Case Structure appears to have vanished. In reality, it still exists but is hidden within the False case.
- Drag the EO_InitHandle Call Library Function Node into the middle of the True case of your Case Structure.
- Left click any empty area of the Block Diagram to deselect the Call Library Function Node. This is a necessary step before you can create a wire at a terminal of this Call Library Function Node.
- Left click the “handle” output of the EO_InitHandle function. You are now dragging a wire out of the Call library Function Node.



- Left Click the output (right side) Tunnel of the Case Structure. The wire is created and the Tunnel fills in with blue.

4.1.13 Troubleshooting


The VI is finished; however, you shouldn't run it yet. Take a look at the Run arrow; if it is broken , there are errors in the VI; continue with this section for assistance troubleshooting the VI. If your Run arrow is not broken , skip the troubleshooting section.

- If your Run arrow appears unbroken , skip this section.
- Compare your Block Diagram to the images shown at the beginning of this chapter of the finished Block Diagram. Correct any differences you find between the two.



Note

Keep in mind that two cases exist within your Case Structure. Verify that both of the cases are configured correctly. If more than two cases exist, delete all except for the True, and False Cases. To delete a case of the Case Structure: right click the border of the Case Structure, select *Delete this case* from the menu that appears.

- Left Click the broken Run arrow  The Errors List window appears.
- Read through the list of errors. Read the Details for each error (there are often helpful recommendations here.)



Note

The LabVIEW Error List shows all outcomes of a mistake, not necessarily the mistake itself. Though several errors may be listed, they may all originate from one missing wire connection or unconnected input.

- Double left click any item of the Error List and LabVIEW will find it on the Block Diagram for you.
- If problems persist, restart this chapter with a new VI. In some cases, this may be the best way to discover the error you've overlooked.

Notes on section 4.1.14

The next VI you create will use this SubVI to obtain a handle for the EO-Drive.

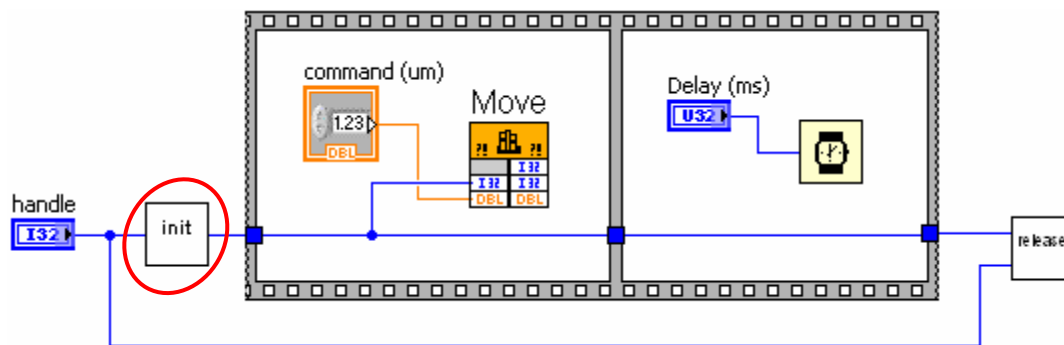
- The Init SubVI is complete. Save this VI in a new folder with the filename: StandardInit.

Handle With Care

This SubVI is not meant to be run alone. There is no provision in this VI for releasing the Handle it has obtained. Because of this, the VI will continue to hold the Handle *even after the VI has stopped running!* This would disable other applications which try to communicate with the EO-Drive. If this happens, you must exit and restart LabVIEW to force the release of all handles held by LabVIEW VIs.

The StandardInit SubVI

Shown below is an image of the finished Block Diagram following completion of Chapter 3. Circled in red is the portion you've created so far. It should now be clear to you how this portion of the VI works: If the Handle passed into the Init SubVI is equal to zero, the Init SubVI acquires the Handle itself. If the incoming Handle is not zero, it is passed to the output. In either case, a Handle is passed out for use within the parent VI.



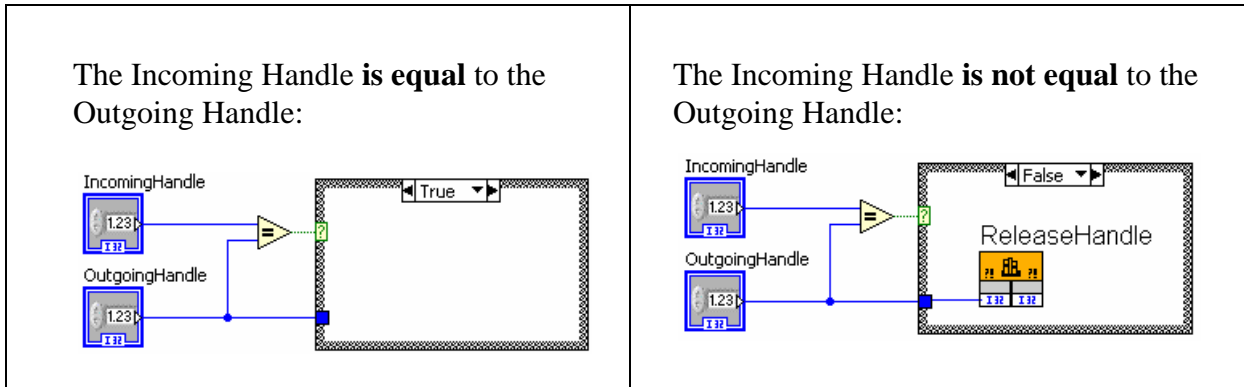
Section 4.2 – Creating the Release SubVI

The process of creating the Release SubVI will closely resemble the previous process by which you created the Init SubVI.

The Release SubVI



Below can be seen the Block Diagram of the Release SubVI which you will be creating in this section of the tutorial. Because a Case Structure is used within this VI, it is again necessary to show two separate states of the same Block Diagram. It is not necessary to see the Front Panel of this VI; the Front Panel simply contains the Incoming Handle and Outgoing Handle Numeric Controls.

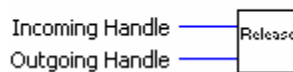


(Block Diagram of the finished Release SubVI)

Inputs of the Release SubVI

The Release SubVI has two inputs and no outputs. The inputs are the Incoming Handle and Outgoing Handle Numeric Controls (seen at left.) The names of the inputs refer to the input and output of the Init SubVI (you created previously) on the CommandEO VI Block Diagram.

StandardRelease.vi



Stand-Alone VI versus SubVI

The StandardInit, and StandardRelease, SubVIs can't do much on their own. They are meant to be used within other VIs which issue commands to the EO-Drive. A VI meant to stand alone (as its own application) must contain within its Block Diagram all the necessary elements to gather all required resources and must use the resources in some meaningful way, handle potential errors, and release the gathered resources back to a non-volatile state. This can, of course, be done with the assistance of SubVIs; the important point is that a VI contains a complete program, whereas a SubVI often only contains a supplementary set of instructions. However, it is sometimes helpful to design a VI to be capable of standing alone while also being ready for use as a SubVI.

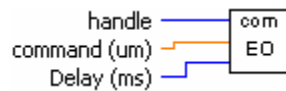
Stand-alone and SubVI Capability

The CommandEO VI that you are creating in this chapter will eventually be used within another VI in a later chapter as a SubVI. However, by the end of this chapter the CommandEO VI will be ready to function as its own useful application.

The CommandEO VI as a SubVI

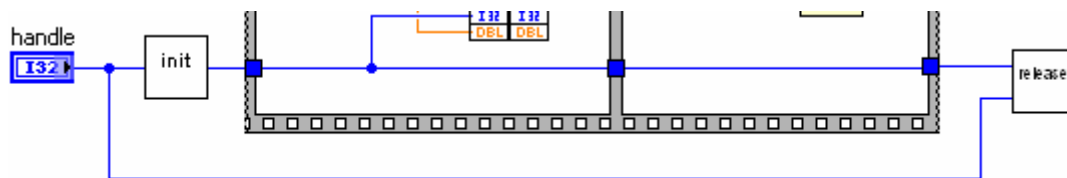
The purpose of the Release SubVI can only be understood with the previous fact in mind. Take a close look at the previous images of the finished Release SubVI, and at the image of the finished CommandEO Block Diagram. The Handle is only released when the Incoming Handle control is different from the Outgoing Handle control. Recall from creating the Init SubVI; this cannot be the case when the Incoming Handle is different than "0". This means the following: a Handle is

passed to the Init SubVI only when the CommandEO VI is being used as a SubVI; therefore, the Handle should not be released (by the Release SubVI) at the termination of the CommandEO SubVI.



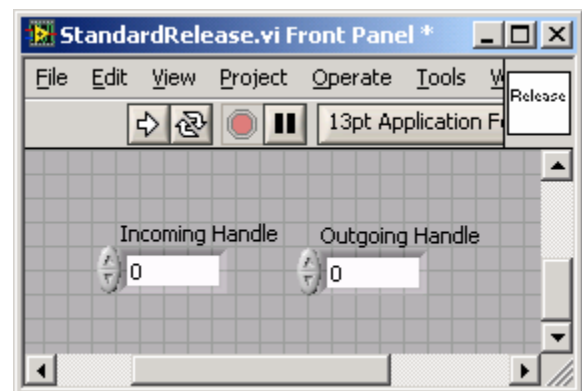
The CommandEO VI as a Stand-Alone Application

It is now apparent that the CommandEO VI does not initialize or release a Handle when used as a SubVI. However, when the CommandEO VI is used as a stand-alone VI, it will both initialize, and release, a handle. The Init SubVI knows to initialize a Handle because its “Incoming Handle” control is equal to zero. The Release SubVI knows to release the Handle because its “Incoming Handle” control is not equal to its “Outgoing Handle” control.




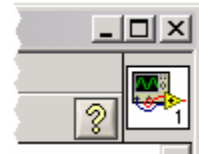
4.2.1 Create the Two Numeric Controls

- Begin LabVIEW and start a new VI.
- Save this VI in the same folder that contains StandardInit.vi, under the new filename: StandardRelease.
- On the Front Panel, create a Numeric Control.
- Change its data type representation to I32 (signed 32-bit integer.)
- Change its label to “Incoming Handle.”
- Create another Numeric Control.
- Change its data type representation to I32.
- Change its label to “Outgoing Handle.”




4.2.2 Create the Terminals of the VI

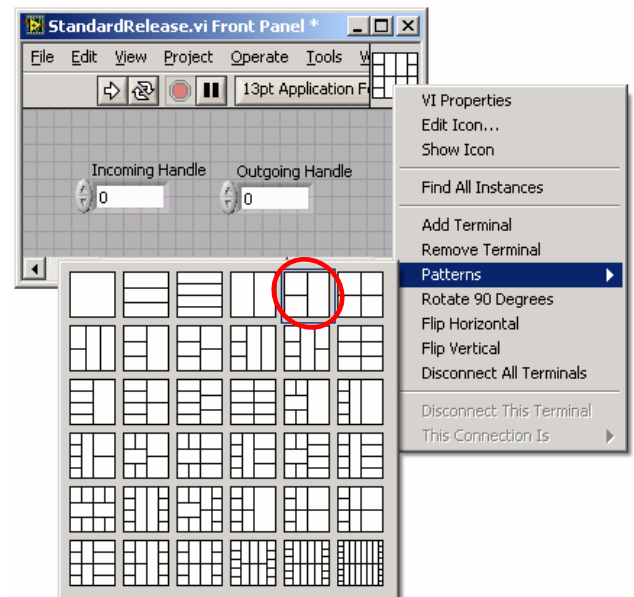
- Right Click the icon  that appears in the top-right corner of your Front Panel.
- Select *Show Connector* from the menu that appears. The Icon should have switched from the graphic of an oscilloscope to the Connector Pane of your VI.



Note

You cannot Show Connectors from the Block Diagram. You will not have access to the options shown in the image below unless you are on the Front Panel.

- Right click the Connector Pane Icon 
- Hover the cursor over the *Patterns* item of the menu that appears (seen at right.)
- Select the fifth pattern from the right edge, on the first row. In the image to the right, a red circle has been added to indicate which pattern to choose.




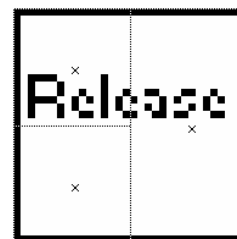
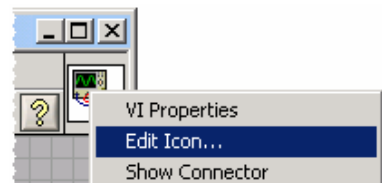
4.2.3 Set the input and output of the VI

- Left click the top input of your VI's Connector Pane. The top-left quarter of the Connector Pane should turn black (as seen on the right.)
- Left click the Incoming Handle control. The top-left quarter of your Connector Pane should turn blue (if it changes from black to a color other than blue, verify the Incoming Handle control is set as an I32.)
- Left Click the bottom input of your VI's Connector Pane.
- Left Click the Outgoing Handle Control. The bottom-left quarter of your Connector Pane should turn blue.



4.2.4 Edit the icon


- Right click the Connector Pane icon.
- Select *Show Icon* from the menu that appears. The Connector Pane graphic should have been replaced with the more familiar graphic of an instrument displaying a sine wave.
- Once again, right click the icon.
- Select *Edit Icon* from the menu that appears. The *Icon Editor* opens.
- Clear the image to white. This can be done either by painting white over the entire image, or by selecting *Clear* from the *Edit* menu.
- Select the  text tool from the toolbar on the left side of the Icon Editor.
- Left click close to the left border of the icon. You can now apply text to this area.
- Enter this word: Release
- Draw a black border around the perimeter of the icon using either the pencil, line, or unfilled square, tool.
- Check the *Show Terminals* option on the right side of the Icon Editor. Your image should now appear very similar to the one shown at right.
- Click the *OK* button at the bottom-right of the Icon Editor.



4.2.5 Configure a Call Library Function Node to Call EO_ReleaseHandle

The Front Panel is finished. Switch to the Block Diagram (hold *Ctrl* and press *E*.)

- From the Block Diagram, create a Call Library Function Node. Placement of the Call Library Function Node is not important at this point.
- Enter its *Configure* menu.
- Direct the *Library name or path* to the EO-Drive.dll using the browse button.
- Under *Function name*, select *EO_ReleaseHandle*.



- Left click the *Parameters* tab at the top of the Configure menu.
- Left click the *Add a Parameter*  button once.
- Left click the newly added parameter, “arg1”.
- In the *Current parameter* area; replace the name, “arg1”, with the name: “handle”.
- All other settings should be correct at their default values. Verify that the *Type* is “Numeric”, and the Data Type is “Signed 32-bit Integer”.

Your Function Prototype should now read:

```
void EO_ReleaseHandle(int32_t handle);
```

- Left click *OK* to accept the changes.

4.2.6 Create the Case Structure and the Equal? Comparison Function

- Create a *Case Structure* in an empty area of the Block Diagram. 
- Create an *Equal?* function. 
- Place the Equal? function in an empty area to the left of the Case Structure.

4.2.7 Arrange the Components in Preparation for Wiring

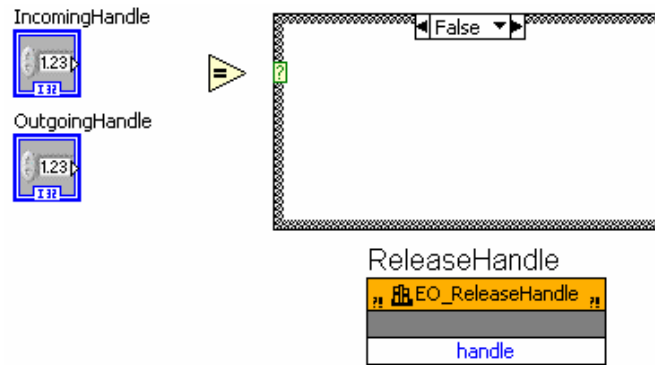
- Verify that your Block Diagram contains all components appearing in the image below.



Note

Don't forget that the Call Library Function Node has two distinct icon formats; Names, and No Names. The Call Library Function Node below is shown with the Names format selected.

- Drag each component of the Block Diagram to the approximate position relative to the Case Structure as seen below.

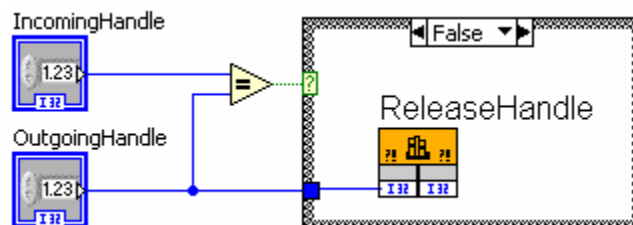


- Drag the EO_ReleaseHandle Call Library Function Node into the False case of your Case Structure.

4.2.8 Make the Wire Connections

You will now be instructed to make the required wire connections. However, you will no longer be given step-by-step instructions for doing so.

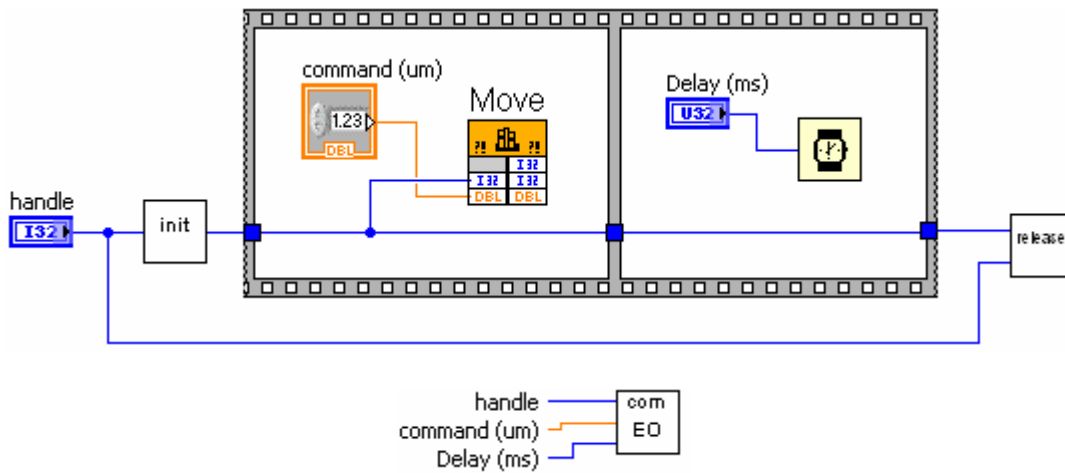
- Connect the output of the Incoming Handle control to the top input of the Equal? comparison function.
- Connect the output of the Outgoing Handle control to the bottom input of the Equal? comparison function.
- Connect the output of the Equal? comparison function to the Selector Terminal of the Case Structure.
- Connect the “handle” input of the EO_ReleaseHandle Call Library Function Node to the output of the Outgoing Handle control.
- Verify that your Block Diagram matches the image below before continuing. The True case of your Case Structure should be empty.



Section 4.3 – Create the CommandEO SubVI

Project Goal

In the following Section, you will create the CommandEO VI which incorporates the SubVIs you've created. As suggested by the title of this section, eventually you will be using the CommandEO VI as a SubVI.

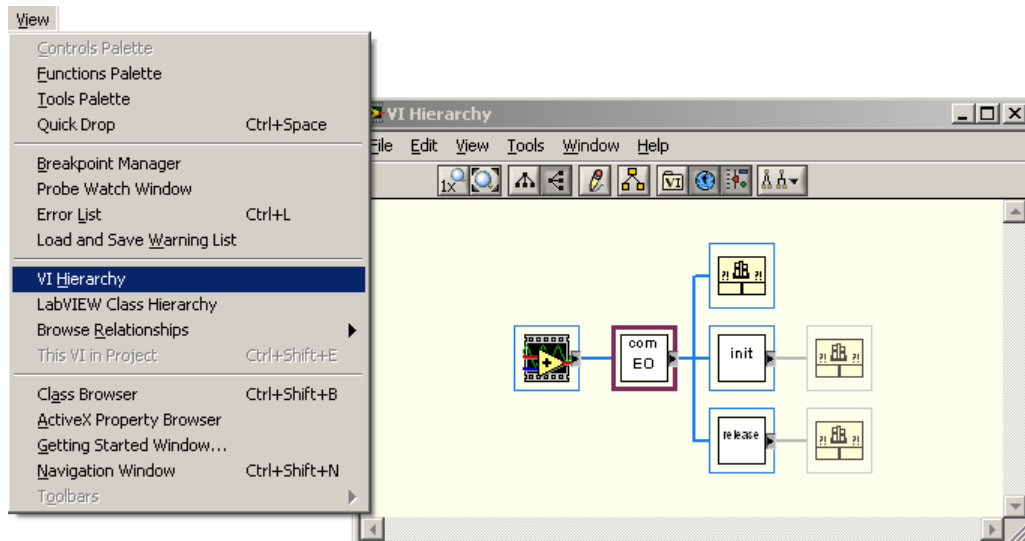


VI Hierarchy

The image below shows the hierarchy of the VIs created in this chapter. The following is an excerpt from the LabVIEW help document (*index* VI Hierarchy window):

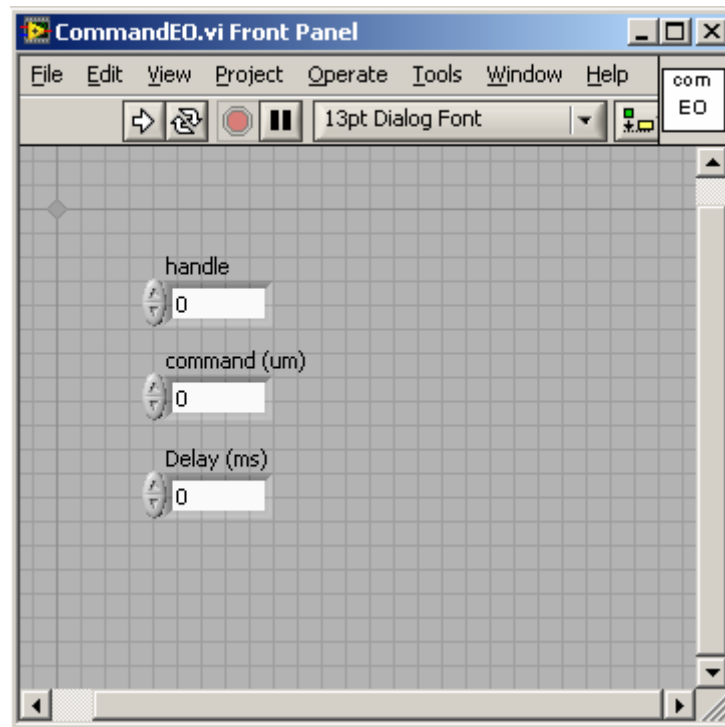
“Use this window to view the subVIs and other nodes that make up the VIs in memory and to search the VI hierarchy. This window displays all open LabVIEW projects and targets, as well as the calling hierarchy for all VIs in memory, including type definitions and global variables.”

The VI Hierarchy window can be accessed at any time from the *View* menu.



4.3.1 Start a New VI, Add the Front Panel Components

- Start a New VI, save it as: CommandEO.
- From the Front Panel, create three Numeric Controls.
- Name the Controls: command (um), Delay (ms), handle.
- Change the Representation of the “command (um)” control to DBL.
- Change the Representation of the “Delay (ms)” control to U32.
- Change the Representation of the “handle” control to I32.




4.3.2 Create and Configure the Flat Sequence Structure, and Timer

- From the Block Diagram, create a Flat Sequence Structure.
- Add a frame to the Flat Sequence Structure for a total of two frames.
- Create a *Wait (ms)* Timer in the second frame of the Flat Sequence Structure.
- Drag the “Delay (ms)” control inside the second frame of the Flat Sequence Structure, and place it to the left of the Wait (ms) Timer.

- Drag the “command (um)” control inside the first frame of the Flat Sequence Structure, and near the left edge.
- Resize the frames of the Structure, and position the other Block Diagram components to match the image below.



4.3.4 Create the Function Call for EO_Move

- Create a Call Library Function Node in the first frame of your Flat Sequence Structure.
- Enter the Configure menu of the Call Library Function Node.
- Under the Function tab, change the Library name or path to the address of the EO-Drive.dll
- From the *Function name* pull-down menu, select EO_Move.
- Switch to the Parameters tab.
- Left click the “return” parameter; under Current parameter, change the Name to “Error Code”, the Type to “Numeric”, and the Data type to “Signed 32-bit Integer”.
- Add two new parameters 
- Left click the first of the new parameters, arg1, and configure the Current parameter area to the following: change the Name to “handle”, Type to “Numeric”, Data type to “Signed 32-bit Integer”, and Pass to “Value”.
- Left click the second parameter, arg2, and configure the *Current parameter* area to the following: Change the Name to “command”, Type to “Numeric”, Data type to “8-byte Double”, and Pass to “Value”.

Verify that your Function Prototype appears as follows:

```
int32_t E0_Move(int32_t handle, double command);
```

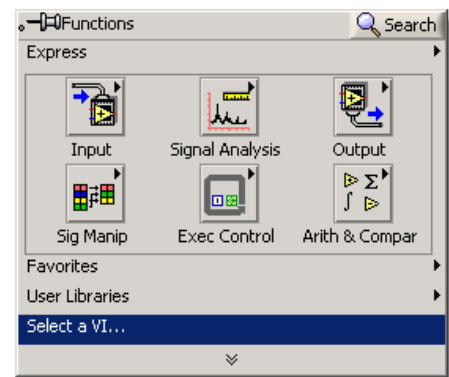
- Left click OK to accept the changes and exit the Call Library Function Node configuration menu.

4.3.5 Add the Init and Release SubVIs

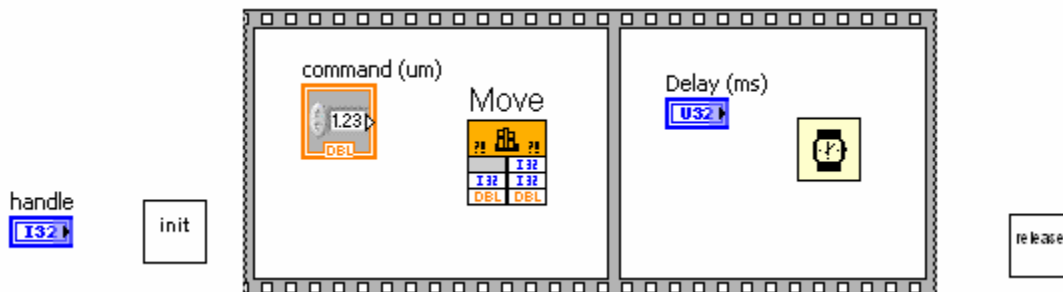
You will now add the Init, and Release, SubVIs. As you will see, the SubVIs can be added to the VI like any other component you've added so far.

- Right click an empty area of the Block Diagram. The Functions palette appears.

- Left click the *Select a VI* menu item. A new window appears with the title: *Select a VI to Open*

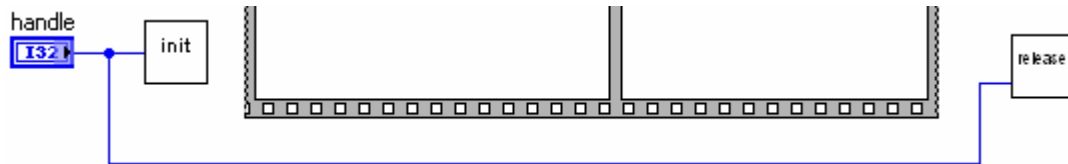


- Use the Select a VI to Open window to locate your StandardInit.vi.
- Left click your StandardInit.vi; it should appear in the *File name:* field near the bottom of the window.
- Left click the OK which appears at the bottom-right of the window. The window vanishes, and you are left holding the icon of your StandardInit.vi
- Left click an empty area at the left of the Flat Sequence Structure to place the StandardInit.vi
- Follow the same process to locate and place the StandardRelease.vi to the right of the Flat Sequence Structure.

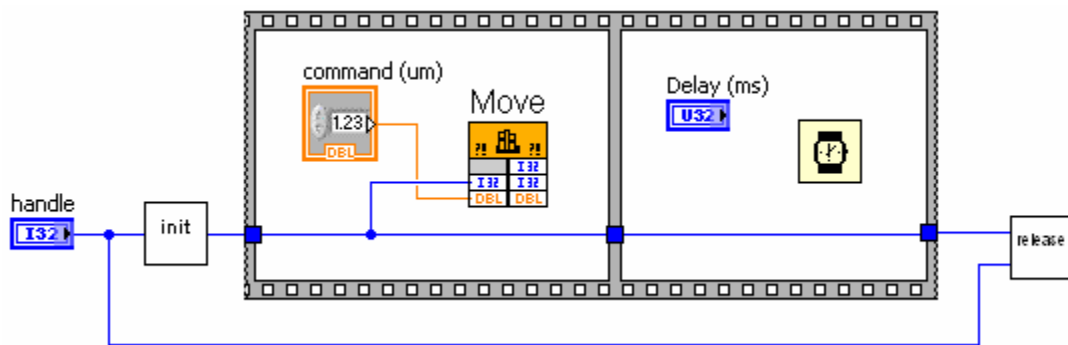


4.3.6 Make the Wire Connections

- Connect the output of the “command (um)” control to the “command” input of the EO_Move Call Library Function Node.
- Connect the “Handle” control to both the StandardInit SubVI input, and the StandardRelease SubVI top input (Incoming Handle). The wire connecting to the Release SubVI should not pass through the Flat Sequence Structure (as seen in the image below.)




- Connect the output of the StandardInit SubVI to the Handle input of the Call Library Function Node, as well as the bottom input of the Release SubVI (Outgoing Handle).



- Finally, connect the output of the “Delay (ms)” control to the input of the Wait (ms) Timer.

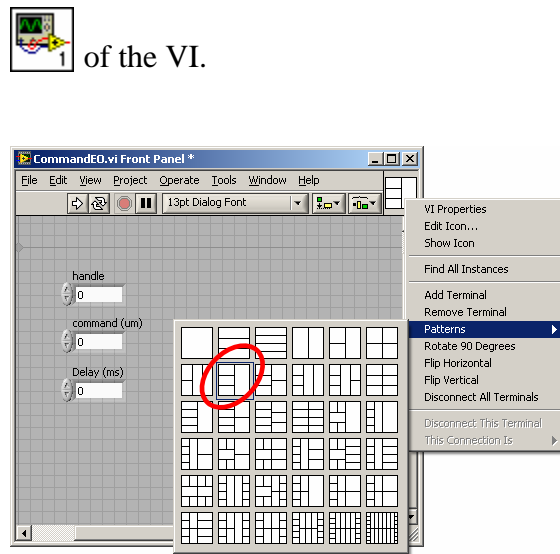
4.3.7 Create the Terminals of the VI


- From the Front Panel, right click the icon  of the VI.
- Select *Show Connector* from the menu that appears. The Icon should have switched from the graphic of an oscilloscope to the Connector Pane of your VI.



Note

You don't have the option to Show Connectors from the Block Diagram. You will have to switch to the Front Panel to have access to the options shown in the image at right.



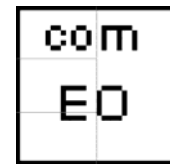
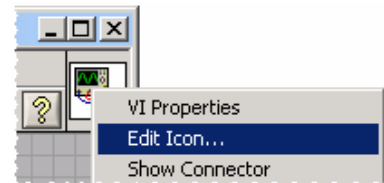
- Right click the Connector Pane Icon .
- Hover the cursor over the *Patterns* item of the menu that appears (seen above.)
- Select the pattern which is second-down from the top, and second-inward from the left edge. In the image above, a red circle has been added to indicate which pattern to choose.

4.3.8 Set the input and output of the VI

- Left click the top input (left side) of your VI's Connector Pane. The selected input of the Connector Pane should turn black.
- Left click the "handle" Control to connect it to the Connector Pane input. The Connector Pane input should turn blue (if it changes from black to a color other than blue, verify the handle Control is set as an Int32).
- Left click the middle input of your VI's Connector Pane. The selected input of the Connector Pane should turn black.
- Left click the "command (um)" control. The Connector Pane input should turn orange.
- Left click the third, and final, input of your VI's Connector Pane.
- Left click the "Delay (ms)" control. The input should turn blue.

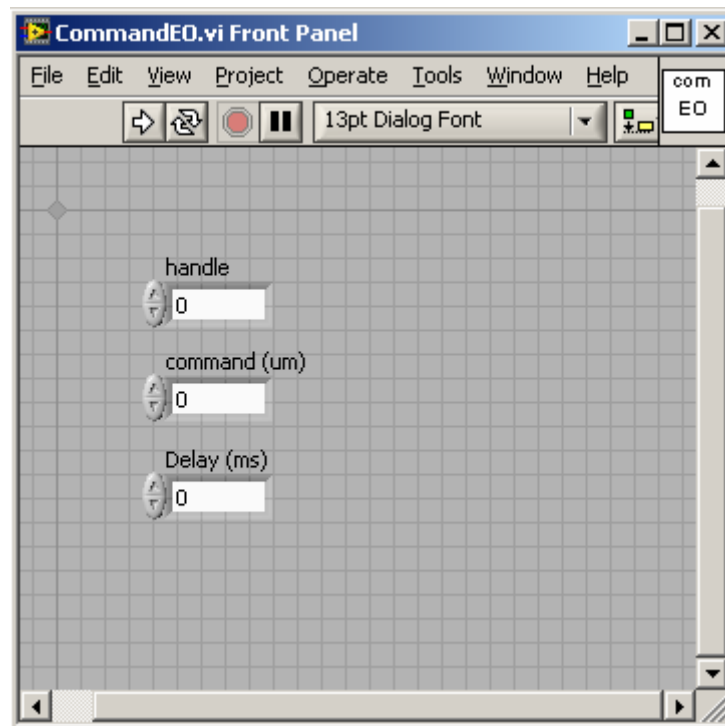
4.3.9 Edit the Icon

- Right click the Connector Pane icon.
- Select *Show Icon* from the menu that appears. The Connector Pane graphic will be replaced with the graphic of an instrument displaying a sine wave.
- Once again, right click the icon.
- Select *Edit Icon* from the menu that appears. The *Icon Editor* opens (as seen below.)
- Edit the Icon graphic to appear similar to the image at right.

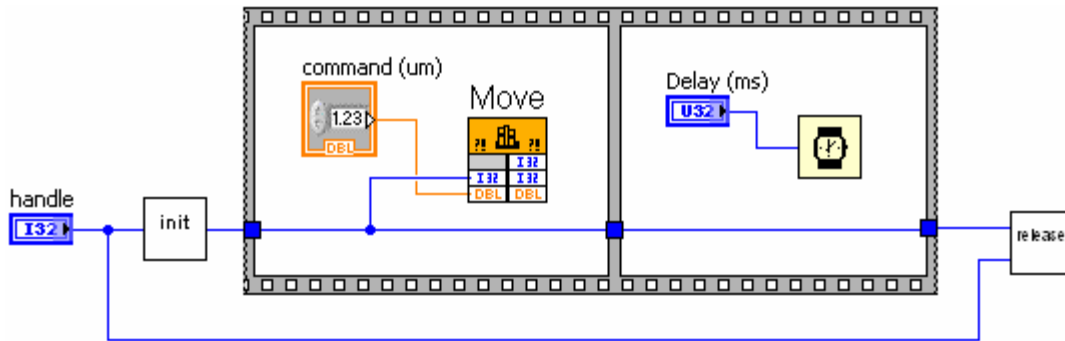


Section 4.4 – Run the CommandEO VI

You now have a VI that will write a position command to the EO-Drive. Though this VI functions similarly to the VI created during Chapter 3, this VI is much better suited to serve as a SubVI within the VI you will create in Chapter 5.



(Front Panel of the finished Chapter 4 VI after a series of executions)



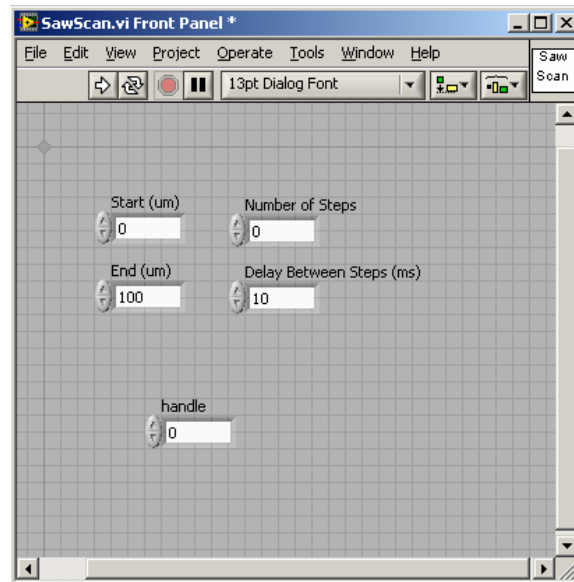
(Block Diagram of the Finished Chapter 4 VI)

4.4.1 Prepare to Run the VI

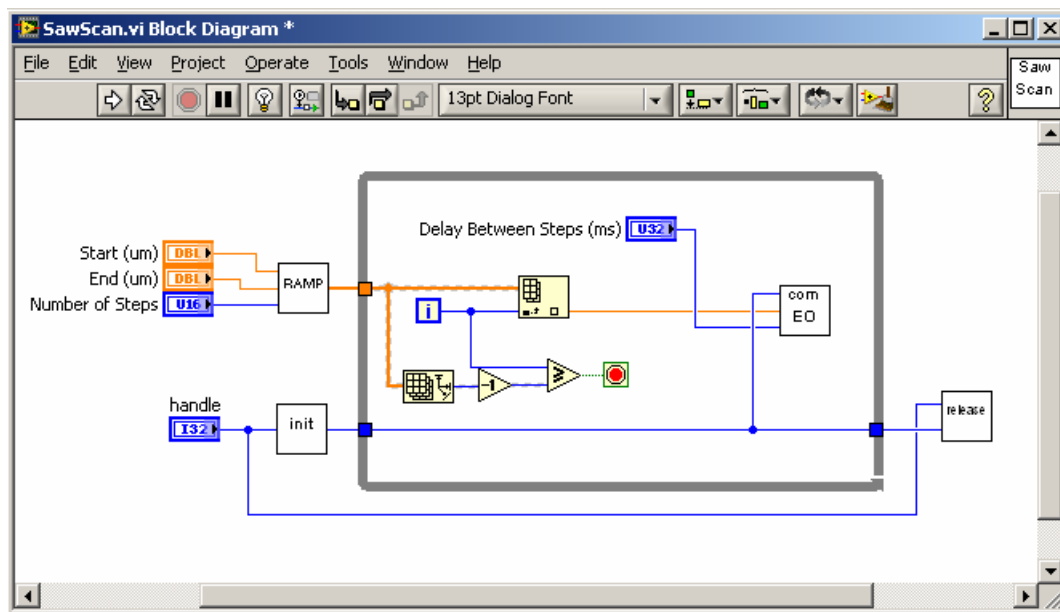
- Save the VI as: CommandEO
- Turn on the EO-Drive.
- Switch to the Front Panel of your VI.
- Enter a position to move to into the “Move to (um)” control. Choose a micron value within the range of motion of your stage.
- Set the “Delay (ms)” control to “10”
- Run the VI. The EO-Drive stage will move to the specified position.

Chapter 5: Create a Scanning VI

The VI shown below will be created in Chapter 5.



(Image of the finished Front Panel by the end Chapter 5)



(Image of the finished Block Diagram by the end of Chapter 5)

Goal of Chapter 5

Chapter 5 demonstrates how to use the skills acquired throughout this tutorial to create a simple scanning program. This Chapter consists of two sections; the first section programs a SubVI to calculate an array of values, the second section programs a VI to command each value of the array to the EO-Drive.

Topics Include:

1. Arrays
2. While Loops
3. For Loops
4. Shift register

EO-Drive.dll Functions Used Within This Chapter Include:

1. **void** EO_ReleaseHandle(**int** handle);
2. **int** EO_InitHandle();
3. **int** EO_Move(**int** handle, **double** command);

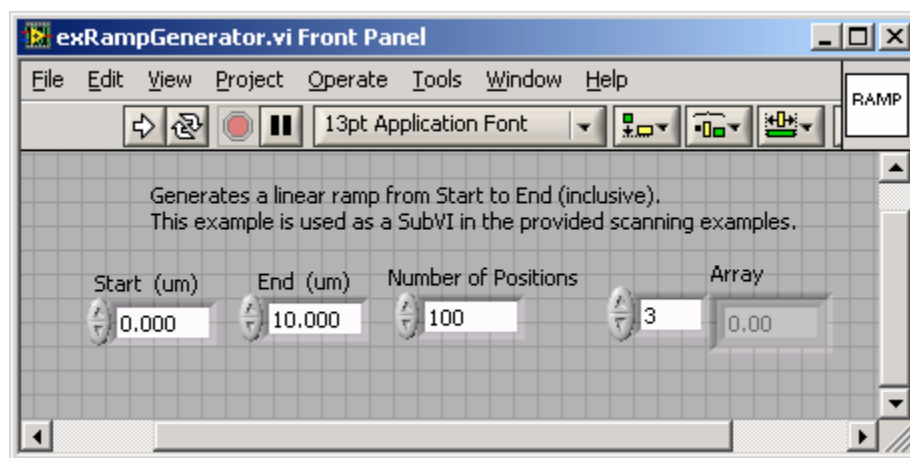
SubVIs Used Within This Chapter Include:

1. StandardInit.vi
2. StandardRelease.vi
3. RampGenerator.vi (*new*)

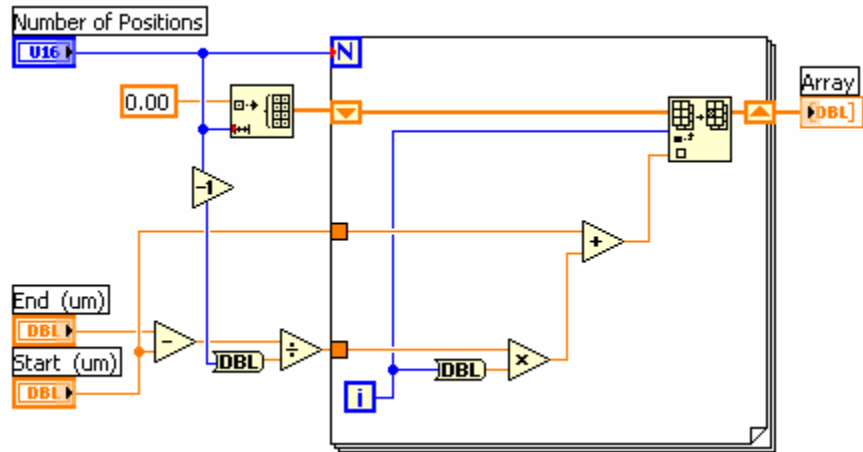
Section 5.1 – Create the Ramp Generator SubVI

Project Goal

Unlike other VIs you've created so far, this VI will not directly communicate with the EO-Drive. Instead, the sole purpose of this VI is to create an array of position values which can be subsequently commanded to the EO-Drive DAC.



(Image of the finished Front Panel of the ramp generating SubVI)



(Image of the finished Block Diagram of the ramp generating SubVI)

5.1.1 Create the Three Numeric Controls

- Begin LabVIEW and start a new VI by selecting *Blank VI* from the *Getting Started* Window. The Front panel and Block Diagram of a new VI appear.
- Save the VI as: RampGenerator.vi
- On the Front Panel, create a Numeric Control.
- Change its data type representation to U16 (Unsigned 16-bit integer.)
- Change its label to “Number of Positions”.
- Create a second Numeric Control.
- Change its data type representation to DBL.
- Change its label to “End (um)”.
- Create a third Numeric Control
- Change its data type representation to DBL.
- Change its label to “Start (um)”.

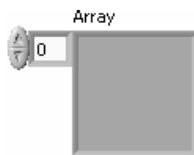
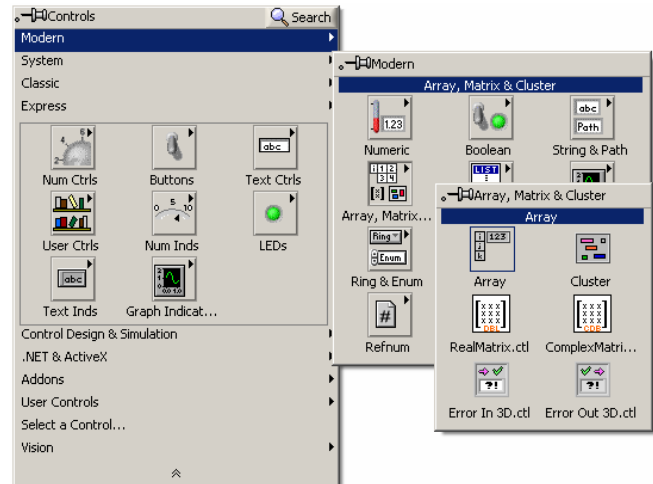
5.1.1 Create the Array Indicator

- Right click an empty area of the Front Panel to access the Controls palette.

- Expand the Controls Palette.



- Open the **Modern>>Array, Matrix & Cluster** sub-palette
- Select **Array**.
- Place the array on an empty area of the Front Panel



Arrays in LabVIEW

Arrays are used to link multiple controls/indicators of the same *representation* together for simpler data manipulation. As an example; imagine a VI that calculates five points for “Y” along the line: $Y=X+2$, in equal “X” increments from: $X = 1$, to $X = 5$. Typically, to display the “Y” value of the five points would require five separate Numeric Indicators. An Array could be used to contain all five of the indicators, and link them to each other. Inside the Array, the indicators are identified not by name, but by their position within the array (i.e. first, second, third, etc...).



Cycling Through the Values Contained Within an Array

The many values stored within an Array can be individually accessed according to their position in the Array by cycling the control at the left of the position value (in the image at right, the value of position 3 of the array is being viewed.) Continuing our example from before, position “0” of the Array would contain the value “3” ($Y=1+2$), position “1” would contain the value “4” ($Y=2+2$), and so on.

- On the Front Panel, create a Numeric Indicator with data type: Double.
- Drag and drop the Numeric Indicator into the empty grey box of the Array Indicator. Each position within the Array now contains its own instance of a Numeric Indicator with data type: Double.

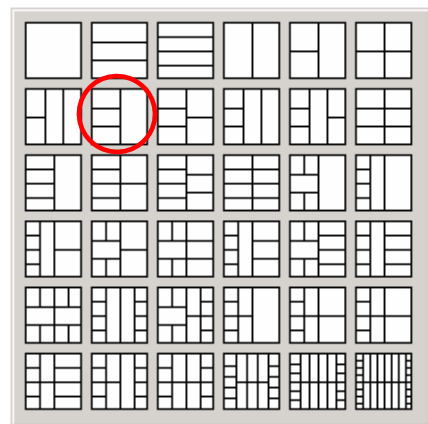
5.1.2 Create the Terminals of the VI

- From the Front Panel, edit the Connector Pane of your VI.
- Select the second pattern down, and to the right, from the top-left corner. In the image above, a red circle has been added to indicate which pattern to choose.



Note

You cannot Show Connectors from the Block Diagram. You will not have access to the options shown in the image at right unless you are on the Front Panel.



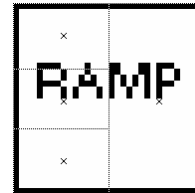
5.1.3 Set the input and output of the VI

- Left click the top input (left side) of your VI's Connector Pane. The top-left sixth of the Connector Pane should turn black.
- Left click the "Start (um)" control. The top-left sixth of your Connector Pane should turn orange (if it changes from black to a color other than orange, verify the "Start (um)" control is set to DBL.)
- Left click the second input of your VI's Connector Pane.
- Left click the "End (um)" control. The second input of your Connector Pane should turn orange.
- Left click the bottom input of your VI's Connector Pane.
- Left click the "Number of Positions" control. The bottom input of your Connector Pane should turn blue.
- Left click the right (output) half of your VI's Connector Pane.
- Left click the "Array" indicator.
- Your VI's Connector Pane should now resemble the image at right:



5.1.4 Edit the icon

- From the Front Panel, right click the Connector Pane icon.
- Select *Show Icon* from the menu that appears.
- Edit the icon of this VI to appear similar to the image at right:



5.1.5 Initialize an Array

You will now be instructed to create an Initialize Array function. The Initialize Array Function will create an array for your VI to fill with position values, incrementing from “Start (um)” to “Finish (um)”.

Array of values

The finished SawScan VI will command the EO-Drive to each of many individual positions. Instead of calculating the next position before each motion command, it is simpler to pre-calculate all position values within a SubVI. We can then pass the array of values to another VI that handles the EO-Drive movement. An Array is simply a conventional way to store large quantities of values.

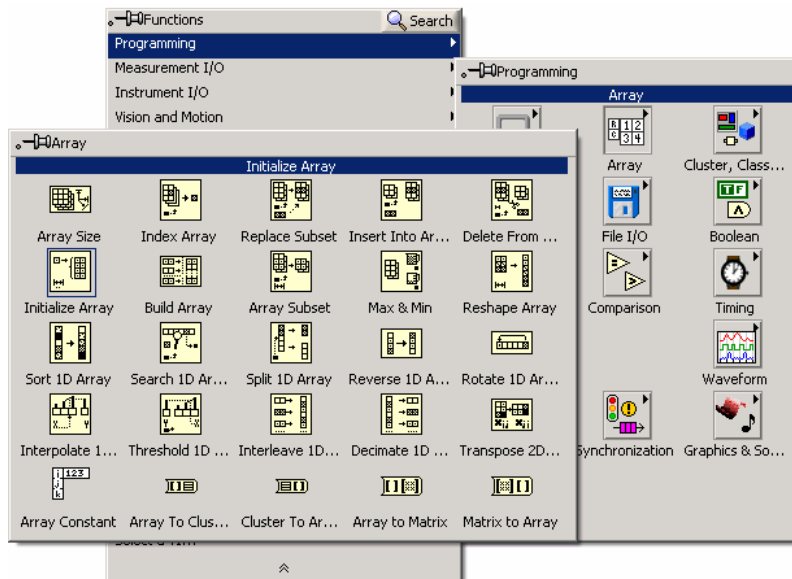
- The Front Panel is finished. Switch to the Block Diagram (hold *Ctrl* and press *E*.)
- Right Click an empty area of the Block Diagram to access the Functions Palette.
- Open the *Programming>>Array* category of the Functions Palette.



Note

In order to see the Programming sub-menu, you may have to left click the dashed down arrow to expand the Functions Palette.

- Left click the *Initialize Array* icon.



- Place the Initialize Array function over any empty area of the Block Diagram.

The Initialize Array Function

As mentioned before, the Initialize Array function creates an array for use within the Block Diagram. Using the Initialize Array function we can specify an initial value for each element of the array, as well as the total number of elements which will exist inside the array. The array is taken from the output (right) of this function.

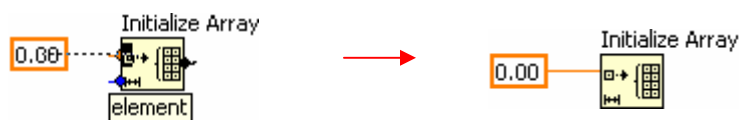
- Create a Numeric Constant with a value of “0”.
- Place the Numeric Constant slightly to the left of the Initialize Array function
- Change the Representation of the Numeric Constant to DBL.



Note

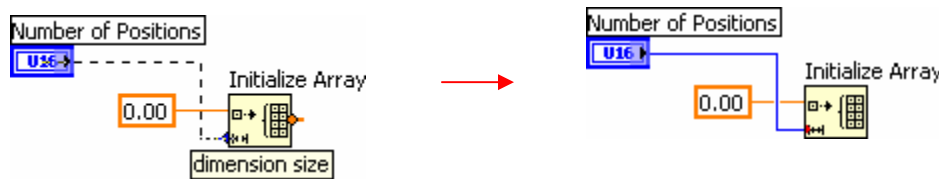
The *Adapt to entered data* option will automatically uncheck after you specify a Data Type Representation for the Constant. In this case, this option will not work for us.

- Create a wire connecting the Numeric Constant to the Initialize Array function’s top input (labeled *element*.)



- Drag the Number of Positions Control above, and to the left, of the Initialize Array function.

- Create a wire connecting the Number of Positions Control to the Initialize Array function's bottom input (labeled *dimension size*.)



5.1.6 Create the For Loop

In the following set of steps, you will be instructed to create a Loop structure. The concept of Loops in LabVIEW can be a difficult topic; use the LabVIEW Help documentation on *For* and *While* loops if you feel this tutorial inadequately covers this topic.

- Right click over any empty area of the Block Diagram to access the Functions palette.
- Open the *Programming>>Structures* category from the Functions palette.
- Left click the *For Loop* icon (seen below).



The For Loop

Similar to the Flat Sequence structure, the For Loop will contain components of your Block Diagram within a single frame, or *subdiagram*. Because of this, data must *tunnel* into, and out of, the For Loop.

The Loop Count Terminal




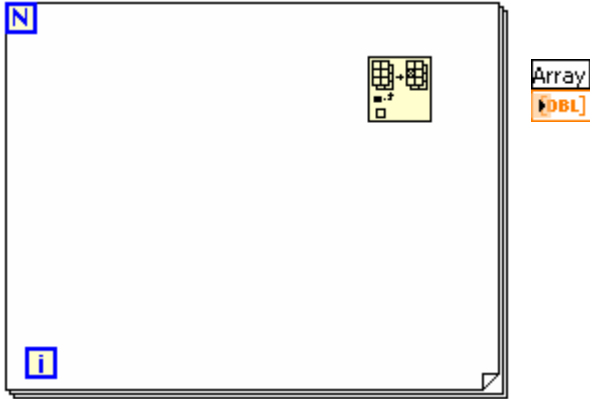
Every For Loop you create is automatically created with a Loop Count terminal in its top-left corner. The Loop Count terminal is used to specify a number of times to consecutively run the For Loop subdiagram. Wiring a numeric input, “n”, to the Loop Count terminal commands the For Loop to run “n” times.

- Left click an empty area of the Block Diagram to begin placing the For Loop.
- Drag your cursor diagonally a few inches and left click again to finish placing a For Loop. Any components that were overlapped by the For Loop during the creation process have been automatically added into For Loop. Remove the overlapped components by dragging them outside of the For Loop.


5.1.7 Create the Replace Array Subset Function

- Right Click an empty area of the Block Diagram to open the Functions Palette.
- Open the *Programming>>Array* category of the Functions Palette.

- Left click the *Replace Subset* icon. 
- Place the Replace Array Subset function inside the For Loop.



5.1.8 Create the Mathematical Operators

- Right Click an empty area of the Block Diagram to open the Functions Palette.
- Open the Arith & Compar>>Numeric category from the Functions Palette.
- Left click the *Subtract* icon . The windows vanish, and you are left holding a Subtract function.
- Left Click an empty area of the Block Diagram at left of the For Loop to place the Subtract function.
- Follow the same process to locate and place the Divide function outside and to the left of the For Loop.
- Create the Multiply and Add functions and place them within the For Loop.

The Loop Iteration Terminal



Every For Loop you create is automatically created with a Loop Iteration terminal. The Loop Iteration terminal simply outputs a numeric value equal to the number of times the For Loop has repeated (outputs a value of “0” during the first run). Recall the picture from the beginning of Chapter 5 of the finished Block Diagram; the Loop Iteration terminal is seen being used within the ramp-generating calculations.

5.1.9 Arrange the Components in Preparation for Wiring

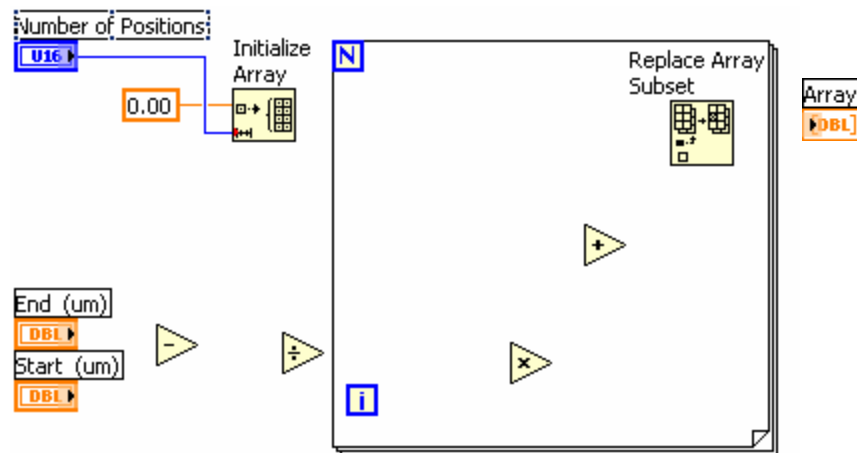
- Verify Your Block Diagram contains all components appearing in the image below.

- Drag each component of the Block Diagram to the approximate position relative to the For Loop, as seen below.



Note

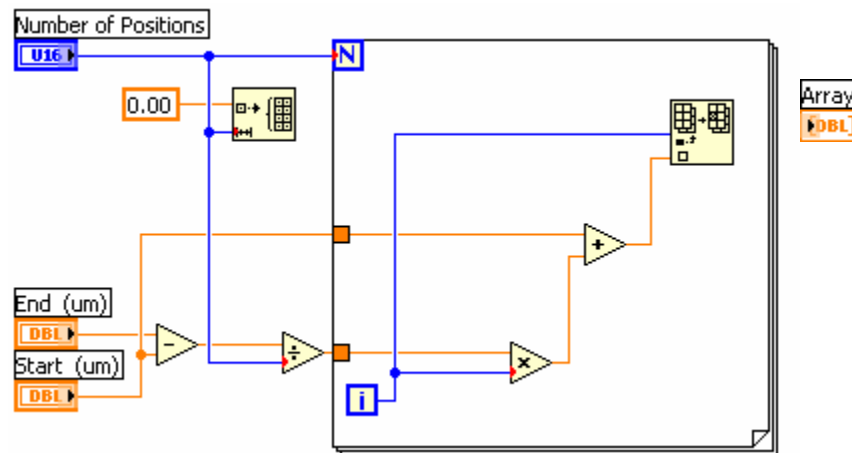
Multiple components can be selected, and dragged, together at a one time. Left click and hold to create a selection box around the components. Include/remove components into the group by holding the Shift key while selecting.



5.1.10 Make the Wire Connections

- Connect the output of the “End (um)” control to the top input of the Subtract function.
- Connect the output of the “Start (um)” control to the bottom input of the Subtract function.
- Connect the output of the Subtract the function to the top input of the Divide function.
- Connect the output of the “Number of Positions” control to the bottom input of the Divide function.
- Connect the output of the Divide function to the top input of the Multiply function. This wire will have to *tunnel* into the For Loop
- Connect the Loop Iteration output to the bottom input of the Multiply function.
- Connect the output of the Multiply function to the bottom input of the Add function.
- Connect the output of the “Start (um)” control to the top input of the Add function.
- Connect the output of the Add function to the Replace Array Subset function’s bottom input, labeled *new element/subarray*.

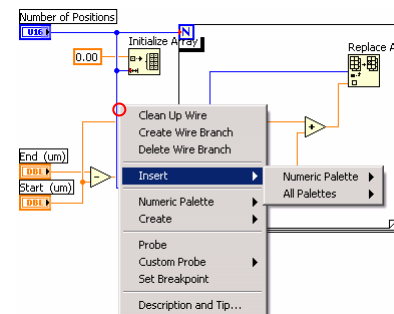
- Connect the Loop Iteration Output to the Replace Array Subset function's middle input, labeled "index".
- Connect the output of the "Number of Positions" control to the Loop Count input.
- Verify that your wire connections are correct by comparing your Block Diagram to the image below:



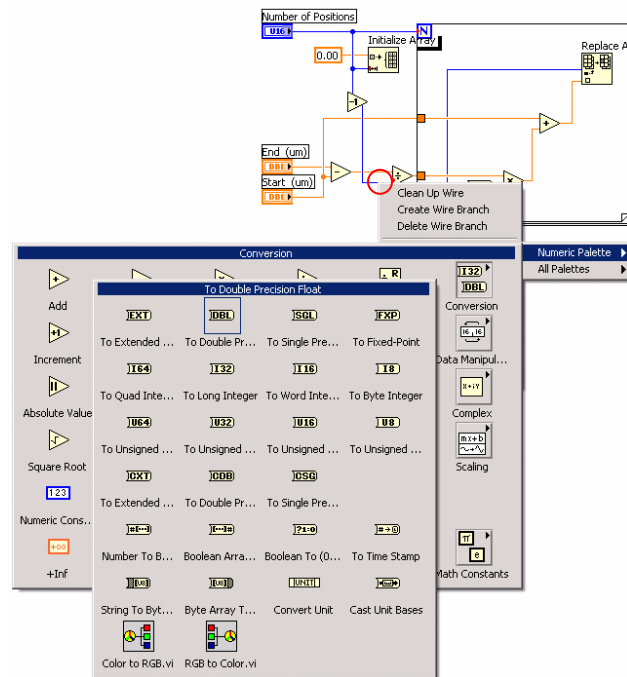
5.1.11 Add the Decrement Operator, and Both Data Type Converters

You must now add a *Decrement* Operator and Data Type Converter to the wire which connects the Number of Positions Control's output to the bottom input of the Division function. However, the Decrement Operator and Data Type Converter you are about to add must not affect the Data passed from the Number of Positions Control to the Initialize Array function.

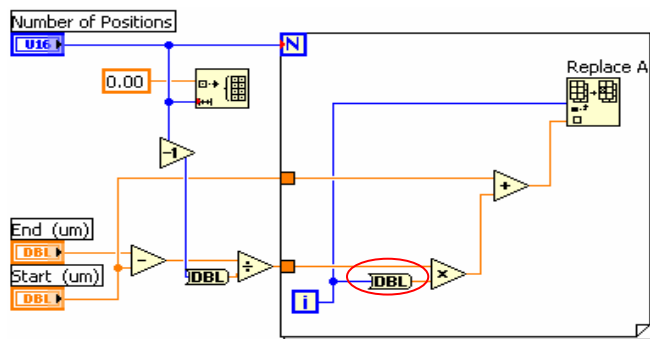
- Right click the wire (circled in red in the image below), which connects the "Number of Positions" output to the bottom input of the Division function. Where it runs vertically toward the Division function, at a point where the wire (from this point) connects on one end to only the Division function.
- Select *Insert* from the shortcut menu that appears.
- Select *Numeric Palette*. The palette containing Numeric Operators opens.
- Left click the Decrement icon. The Decrement function has been automatically inserted into the wire.
- Right Click the same wire at the horizontal section between the Decrement Operator and the Division function.



- Select Insert from the shortcut menu.
- Select the Numeric Palette.
- From the Numeric Palette; select Conversion. The Conversion Palette opens.
- Select *To Double Precision Float*. The windows vanish, and a Data Type Converter has been inserted into the wire.
- Right click the wire which passes data from the Loop Count output to the bottom input of the Multiplication function at a horizontal section.



- Select Insert from the shortcut menu.
- Once again, select the Numeric palette.
- From the Numeric palette; open the Conversion palette.
- From the Conversion palette; select the To Double Precision Float icon. LabVIEW has automatically inserted the Data Type Converter into the wire for you.

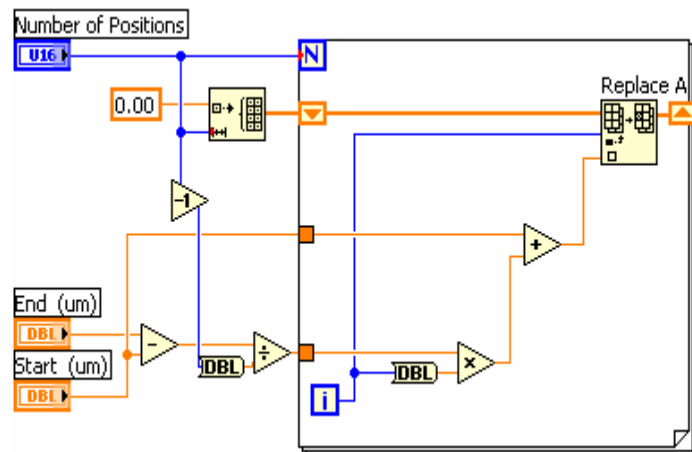
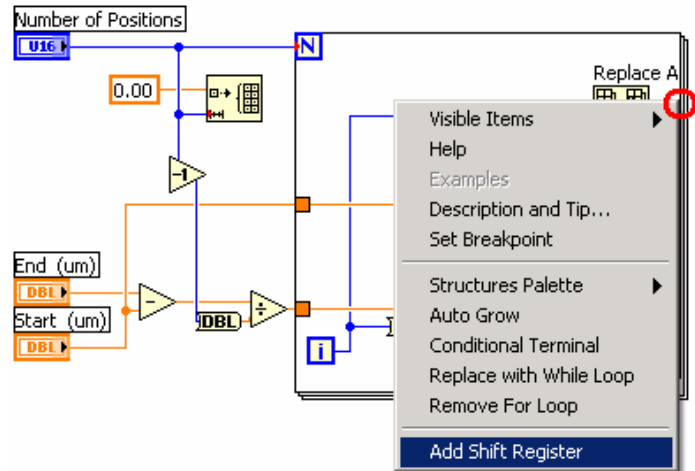


5.1.12 Add, and connect, the Shift Registers for the For Loop

Your Ramp SubVI is nearly complete. The current VI is ready to generate the ramp values; however, it fails to store the values anywhere useful. You will now insert a shift register to retain the data through successive iterations of the For Loop.

- Right click the right edge of the For Loop. A red circle has been added to the image below, right click at this point.
- Select *Add Shift Register* from the shortcut menu that appears. A Shift Register has been created on the right, and left, edge of the For Loop.

- Drag a Shift Register vertically to place it in line with the Replace Array Subset function. Notice the Shift Registers move together.
- Create a wire to connect the output of the Initialize Array function to the input of the left shift register.
- Create a wire to connect the output of the left Shift Register to the Replace Array Subset function's top input, labeled "array".
- Connect the output of the Replace Array Subset function to the input of the right Shift Register.
- Connect the output of right Shift Register to the input of the Array indicator



The Shift Register



The Shift Register consists of two separate pieces, together allowing the programmer to pass data from an iteration of a looping structure to the next. Upon completion of the For Loop the right-most Shift Register will output its most recent value. The value passed through Shift Registers can be of any type, including Array or Cluster.

Initializing a Shift Register



Before a Shift Register can be used, it must be initialized. The Shift Register at left has been initialized with an I32. The Value at the output of the left-most Shift Register upon the first iteration of the For Loop would be "0". In our Ramp SubVI, the Shift Register is initialized with an array of values; therefore, each

iteration of the For Loop must not only specify new values for the Shift Register, but must also specify which element of the array will hold the value.

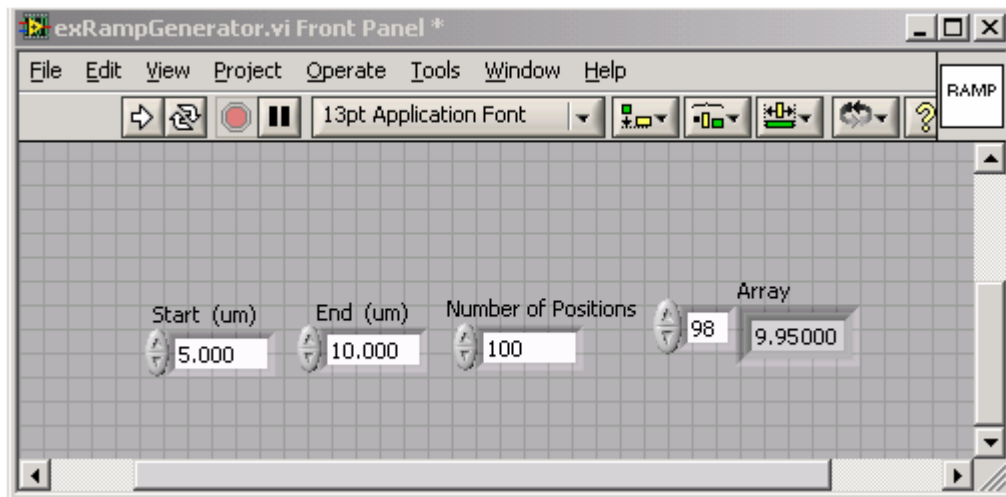
5.1.13 Test the Ramp VI

- Switch to the front Panel of your VI.
- Enter the number “10” into the “End (um)” control.
- Enter the number “101” into the “Number of Positions” control.
- Enter the number “5” into the “Start (um)” control.
- Run the VI.
- Notice the first element (0) of the Array now holds the value 5.

Viewing values stored in an Array

From the Front Panel, use the arrow controls beside the Array indicator to view the values of other elements. The values should be incrementing toward the “End (um)” value, 10. The amount of the increments should be:

$$(End - Start) / (Number\ of\ Positions - 1)$$

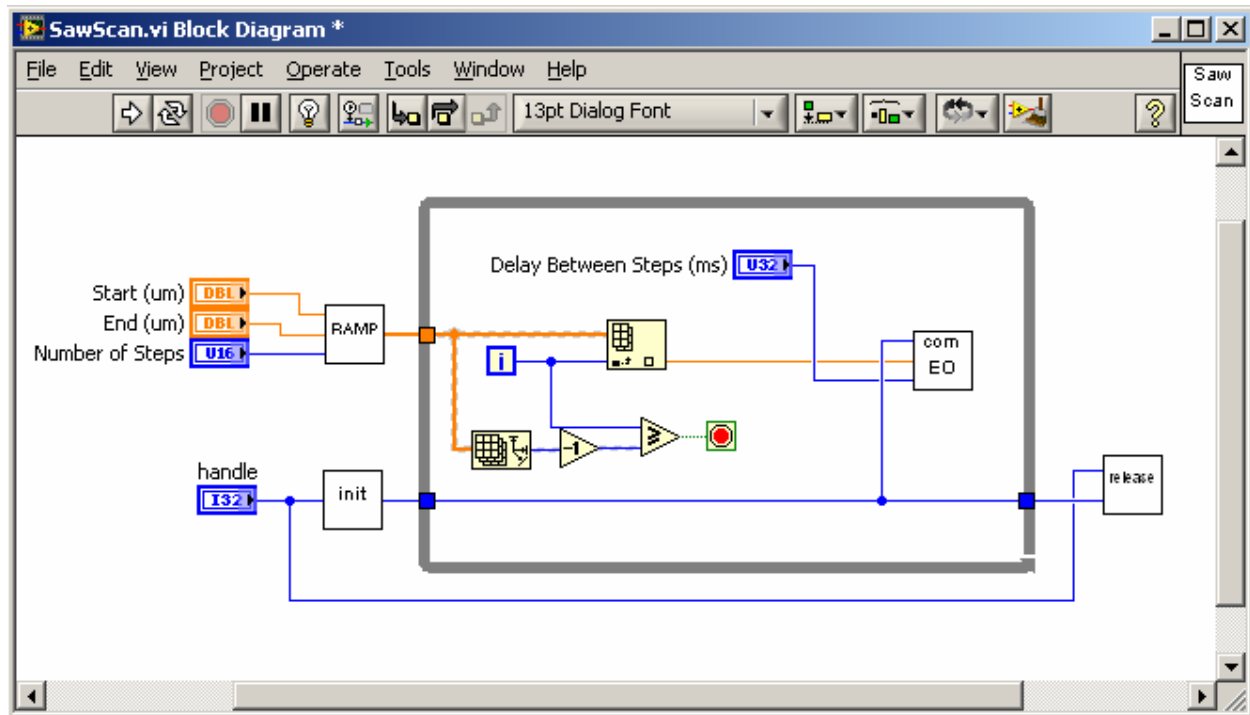


This equates to increments of 0.05 for the test values you entered previously.

Section 5.2 – Create the Saw Scan VI

Project Goal

It is now time to create the Saw Scan VI which uses the array generated by the Ramp SubVI to command an axis to each of several positions.



(Image of the finished Saw Scan Block Diagram)

5.2.1 Start a New VI, Add the Front Panel Components

- Start a New VI, save it as: SawScan.vi
- From the Front Panel, create five Numeric Controls.
- Name the controls: Start (um), End (um), Delay Between Steps (ms), Handle, Number of Steps.
- Change the Representation of the “Start (um)” control to DBL.
- Change the Representation of the “End (um)” control to DBL.
- Change the Representation of the “Delay Between Steps (ms)” control to U32.
- Change the Representation of the “Handle” control to I32.

- Change the Representation of the “Number of Steps” control to U16.
- Edit the Display Format of the “Start (um)”, and “End (um)”, control to display as a Floating Point with 4 digits of precision.

5.2.2 Create and Configure the While Loop

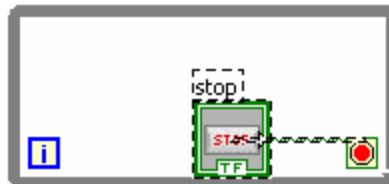
- From the Block Diagram, Right click an empty area to access the Functions palette.
- From the Functions palette, open the Express>>Execution Control category.
- Select *While Loop*.
- Place the While Loop on the Block Diagram. The While Loop is placed in the same manner as the Flat Sequence/Case Structure.

While Loop



The While Loop and For Loop are very similar in functionality. The main difference being; the For Loop repeats a specific number of times, but, the While Loop continues to repeat its subdiagram until commanded to stop. Like the For Loop, the While Loop possesses a Loop Iteration terminal; however, the Loop Count terminal is replaced by the “Loop Condition” terminal.

- If a Boolean Control, labeled “stop”, was automatically created with the While Loop; delete it now. Delete the wire that was connecting it to the Loop Condition input.

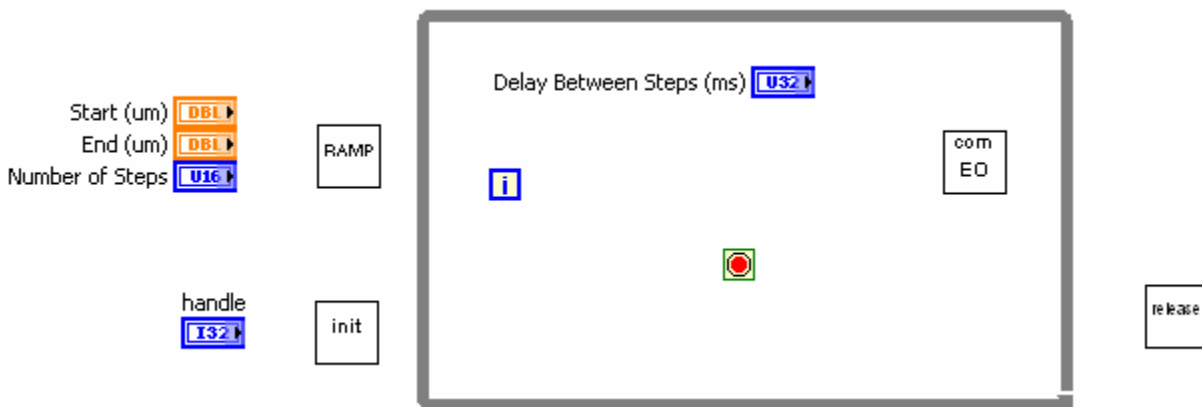


- Drag all other Block Diagram Components to the left side of the While loop.
- Resize the While Loop. Make it large enough to fit all the components that will later be dragged inside it (as seen in the image below, or at the beginning of this chapter.)

5.2.7 Add the Init, Release, and the Ramp SubVIs

- Right click an empty area of the Block Diagram. The Functions palette appears.
- Left click the *Select a VI* menu item. A new window appears with the title: *Select a Vi to Open*

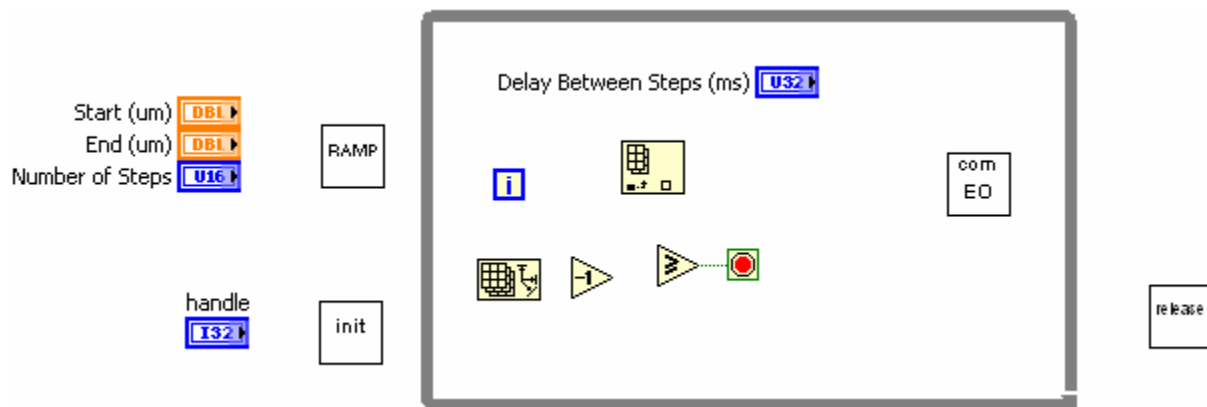
- Use the Select a VI to Open window to locate your StandardInit.vi.
- Left click your StandardInit.vi; it should appear in the *File name* field near the bottom of the window.
- Left click the OK which appears at the bottom-right of the window. The window vanishes, and you are left holding the icon of your StandardInit.vi
- Left click an empty area left of the While Loop to place the StandardInit.vi
- Follow the same process to locate and place the StandardRelease.vi to the right of the While Loop.
- Follow the same process to place the RampGenerator.vi SubVI outside, and to the left, of the While Loop.
- Finally, follow the same process to place the CommandEO.vi SubVI inside the while loop, and to the right of the other contained operators.



5.2.9 Create the Array functions, and Arithmetic & Comparison Operators

- Select the *Index Array* function from the Array palette (the Array Palette can be found in the Programming category of the Functions palette.)
- Place the Index Array function inside the While Loop, near the middle.
- Select the *Array Size* function from the Array palette.
- Place the Array Size function below, and to the left, of the Index Array function.
- Select a Decrement function from the Numeric palette (found in the Arithmetic & Comparison category of the Functions palette).

- Place the Decrement function slightly below and to the right of the Array Size function.
- Select the *Greater or Equal?* Function from the *Programming>>Comparison* category of the Functions palette (also found in the *Express>>Arithmetic & Comparison>>Comparison* category of the Functions palette).
- Place the Greater or Equal? function slightly to the right of the Subtract function.



The Loop Condition Terminal



Every While Loop you create is automatically created with a Loop Condition terminal. The only way to stop a While Loop is to pass a value of “True” to the Loop Condition terminal. How can we tell when to stop our SawScan VI’s While Loop? The answer is simple: the value of the Loop Iteration terminal will equal the size of the Ramp SubVI’s array only after each position command has been withdrawn from the array; therefore, the while loop should stop.

5.2.10 Arrange the Components in Preparation for Wiring

Your VI should now contain all the components seen in the images below. If this is not the case, browse the previous set of instructions for the step(s) you’ve overlooked or try creating the component(s) according to the images.

- Arrange the components of your Block Diagram to closely match the positions of the components as depicted in the image above.



Note

To view the label of a function (such as the array functions below): right click the function, and select *visible items>>label*.

- Verify that the Data Type Representations of all your Numeric Controls are correct before continuing.

5.2.11 Connect the Wires

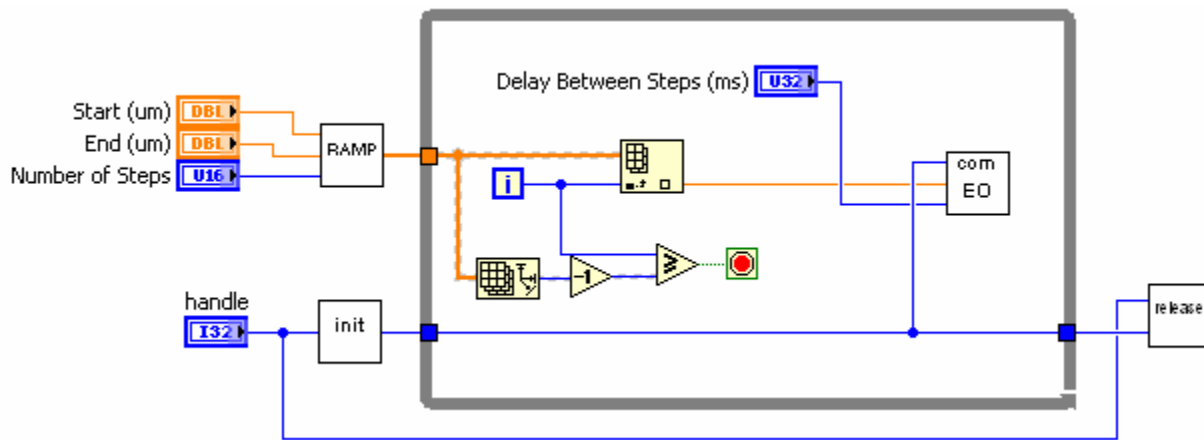
- Create a wire to connect the “Start (um)” control to the Ramp SubVI’s top input, labeled “start (um)”.
- Create a wire to connect the “End (um)” control to the Ramp SubVI’s middle input, labeled “end (um)”.
- Create a wire to connect the “Number of Steps” control to the Ramp SubVI’s bottom input, labeled “Number of steps”.
- Connect the output of the Ramp SubVI to the Index Array function’s top input, labeled “array”. The wire will tunnel into the While Loop to make this connection.
- Connect the output of the Ramp SubVI to the input of the Array Size function. Either tunnel into the While Loop to make the connection or start the wire at a point on the existing Ramp SubVI output wire which is already inside the While Loop.
- Connect the output of the Loop Iteration counter to the Index Array function’s bottom input, labeled “index”.
- Connect the output of the Loop Iteration counter to the top input of the Greater or Equal? comparison operator.
- Connect the output of the Array Size function to the input of the Decrement function.
- Connect the output of the Decrement function to the bottom input of the Greater or Equal? comparison operator.
- Connect the output of the Greater or Equal? comparison operator to the input of the Loop Condition terminal.
- Connect the output of the “Handle” control to the input of the Init SubVI.
- Connect the output of the “Handle” control to the Release SubVI’s top input, labeled “Incoming Handle”. Do not tunnel through the While Loop to make this connection.
- Connect the output of the Init SubVI to the bottom input of the Release SubVI, labeled “Outgoing Handle”. This wire must tunnel through the While Loop; into the left edge and out the right edge.
- Connect the output of the Init SubVI to the CommandEO SubVI’s top input, labeled “Handle”. Either tunnel into the While Loop to make this connection or start the wire at a point on the existing Init SubVI output wire which is already inside the While Loop.

- Connect the output of the “Delay Between Steps (ms)” control to the “Delay (ms)” input of the CommandEO SubVI.
- Connect the output of the Index Array function to the CommandEO SubVI’s top input, labeled “command (um)”.

Section 5.3 – Prepare the VI to Scan an Axis

Project Goal

The SawScan VI is complete. Compare your Block Diagram to the image below before proceeding. The VI is almost ready to run, but, first, you will need to enter valid parameters for the VI. After entering the parameters, you will set them as *default values*; as default values, they will be loaded with the VI every time it’s opened in the future.



(Image of the finished Block Diagram for the Saw Scan VI)

5.3.1 Set Default Values for the Front Panel Components

- Switch to the Front Panel of your SawScan.vi.
- Enter a micron Value into the “End (um)” control which is within the range of motion of your stage.
- Right click the “End (um)” control to access its shortcut menu.
- Select the Data Operations category from the “End (um)” control’s shortcut menu.
- Select *Make Current Value Default* from the list of Data Operations items.
- Enter the value “100” into the “Number of Steps” control and follow the same process as above to make this value the default value for this control.

- Enter the value “10” into the “Delay Between Steps (ms)” control and make this value the default for this control.
- All other controls can remain at their initial default value, “0”.

5.3.2 *Run the VI*

- Run the VI. The stage is commanded through a ramp according to the input parameters.

Project Completed

You have programmed a LabVIEW VI to command an EO-Drive through a scan. This VI makes use of the fundamental LabVIEW programming concepts, which you will require as you program and customize VIs of your own. We designed this tutorial to cover the biggest hurdles to getting started; however, there is still plenty left to learn about LabVIEW programming. From this point on, we suggest you refer to LabVIEW help documentation anytime you’re about to try something new.