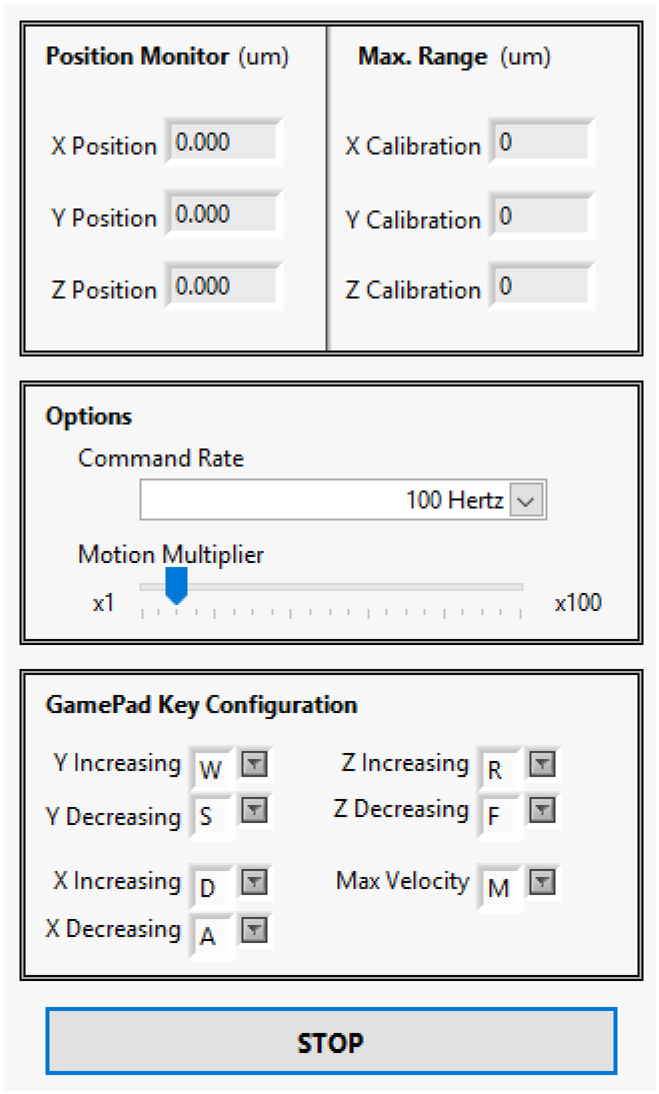
Application Note – NanoDrive® Control via a USB Game Pad

The Keyboard NanoDrive Control VI shown below will be explained in this Application Note.


Position Monitor (um)

X Position 0.000

Y Position 0.000

Z Position 0.000

Max. Range (um)

X Calibration 0

Y Calibration 0

Z Calibration 0

Options

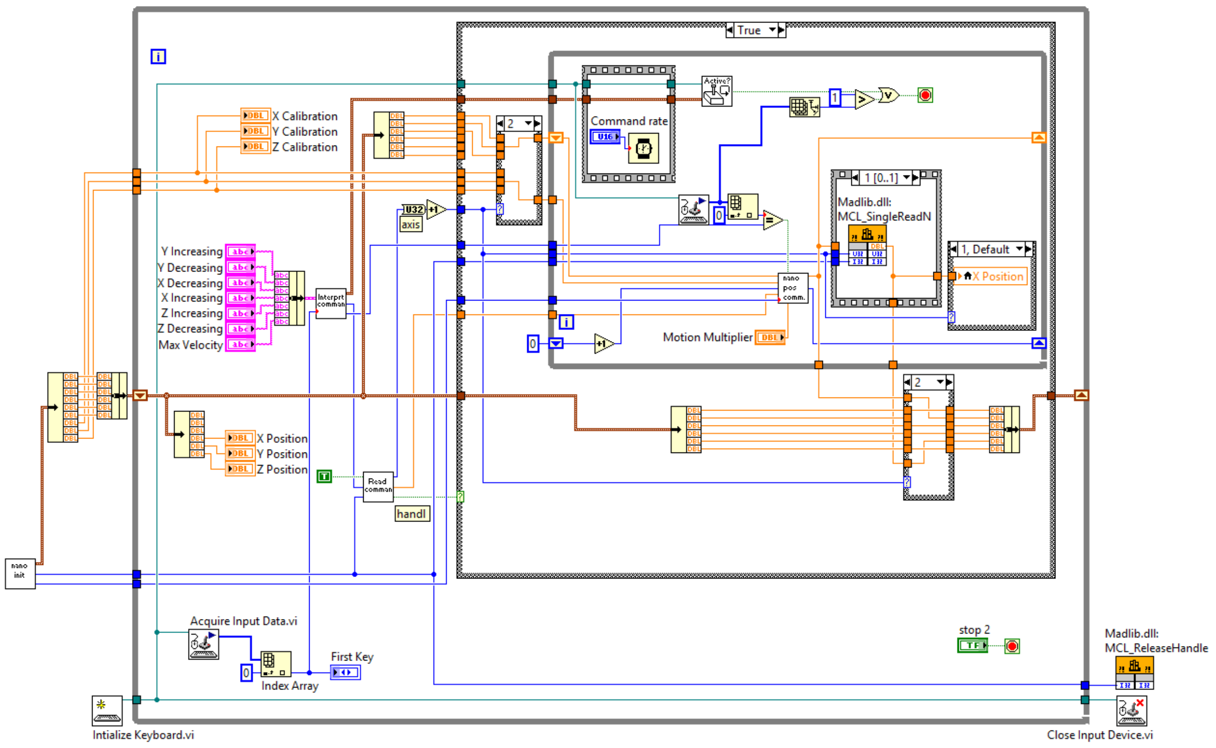
Command Rate: 100 Hertz

Motion Multiplier: x1 – x100

GamePad Key Configuration

Y Increasing	W	Z Increasing	R
Y Decreasing	S	Z Decreasing	F
X Increasing	D	Max Velocity	M
X Decreasing	A		

(Image of the finished Front Panel)



(Image shows the finished Block Diagram)

Game Pad Control in LabVIEW

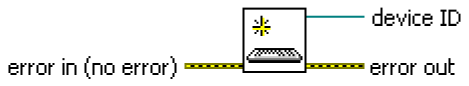
This Application Note will demonstrate creating a LabVIEW VI to respond to actions of the keyboard. The Logitech Game Pad, as well as most other Game Pad controllers, can be configured to replicate keyboard actions via the Game Pad. This means a VI which responds to keyboard actions is ready to be controlled by most Game Pad controllers available.

Topics Include:

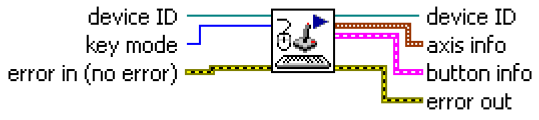
1. Intermediate LabVIEW programming concepts.
2. Initializing/Commanding/Reading a Mad City Labs' (MCL) product.
3. Handling clustered data in LabVIEW.
4. Opening a reference to and initializing the keyboard.
5. Acquiring device data and using the data on the Block Diagram.
6. Closing a reference to a device.

NI Originated SubVIs Used Within The Example VI Include:

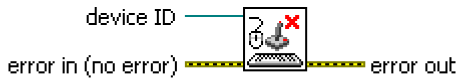
1. Initialize Keyboard.vi



2. Acquire Input Data.vi

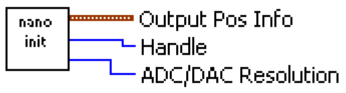


3. Close Input Device.vi

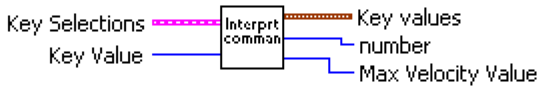


MCL Originated SubVIs Used Within The Example VI Include:

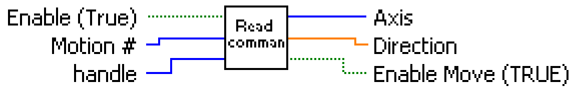
1. Init.vi



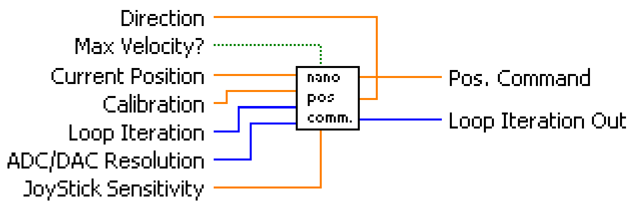
2. GetKeyboardInfo.vi



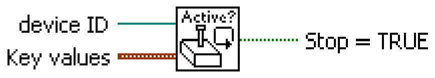
3. ReadCommand.vi



4. IssueCommand.vi

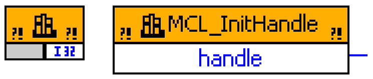


5. KeyPressed.vi



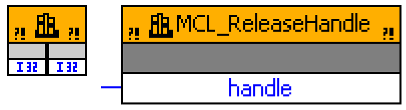
Madlib.dll Functions Used Within the Example VI Include:

1. int MCL_InitHandle();



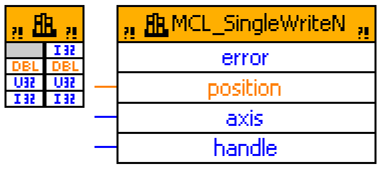
Requests control of a single Mad City Labs' Nano-Drive^{TM}. A handle is acquired for the device which serves as a device identification number within future commands to the Nano-Drive.

2. void MCL_ReleaseHandle(int handle);



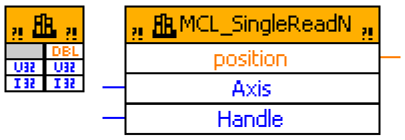
Performs some cleanup operations and releases control of the specified Nano-Drive^{TM}.

3. int MCL_SingleWriteN(double position, unsigned int axis, int handle);



Commands the Nano-Drive to move the specified axis to a position. Returns "0" upon success.

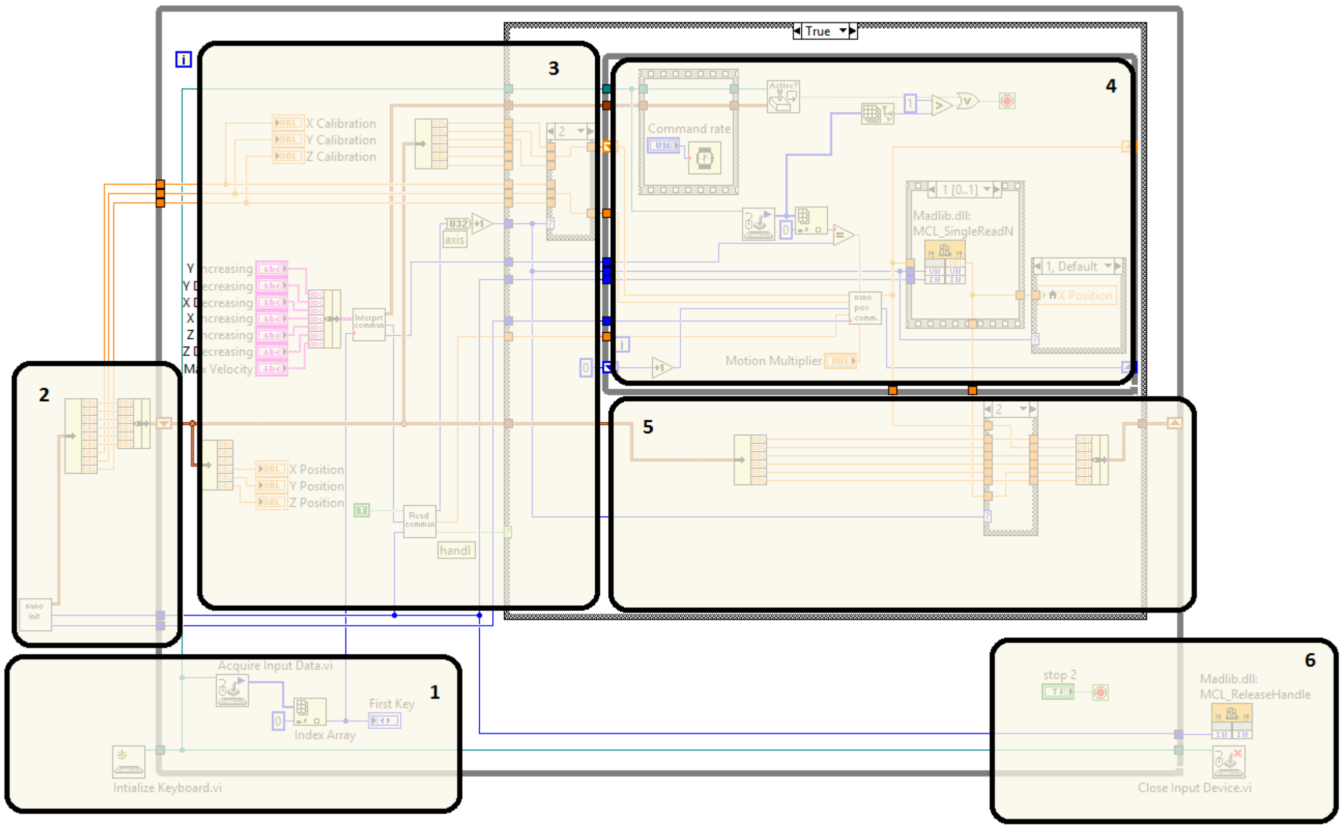
4. MCL_SingleReadN (unsigned int axis, int handle);



Reads the position of the specified axis. Returns a position value or the appropriate error code.

The Six Main Processes of the VI

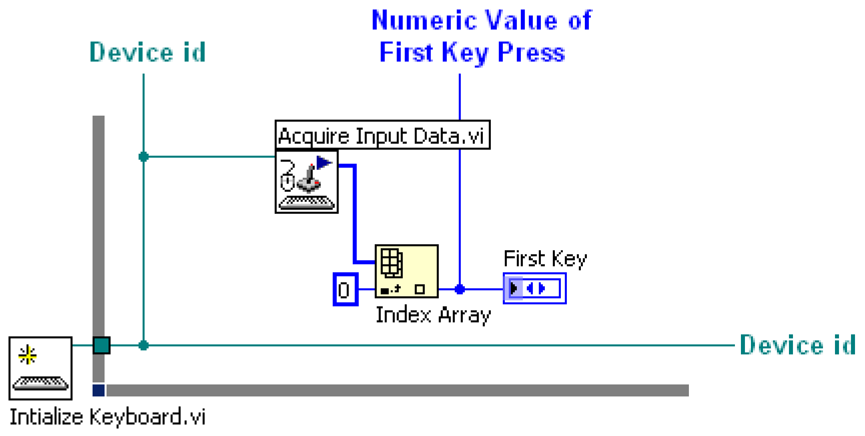
The components of the Block Diagram below have been deliberately placed neighboring components serving similar purposes. Therefore, it has been made possible to highlight six separate sections of the Block Diagram creating what will be considered the six main processes of this VI. The processes will be explained in an order according to the assigned numeric sequence as seen below.



The Six Processes of KeyboardNanoDriveControl.vi:

1. Initiate and Acquire Data from the Keyboard
2. Initiate the Nano-Drive®
3. Translate Data from the Keyboard into Motion Commands
4. Move the Nano-Drive®
5. Update Position Display Data
6. Allow User to End the VI

Process 1 – Initiate and Acquire Data from the Keyboard

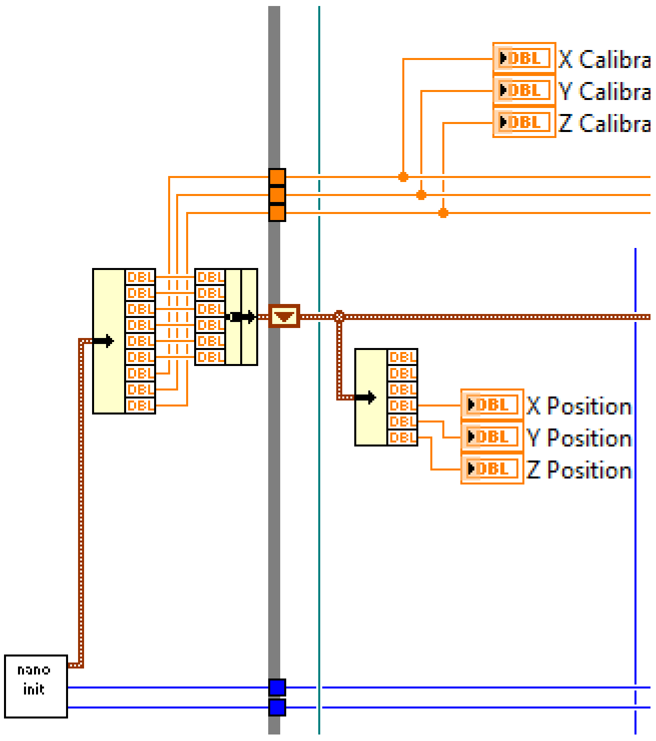


Initialize Keyboard.vi SubVI
This SubVI opens a reference to and initializes the keyboard for use within the Nano-Drive® VI. The "Device ID" obtained by this SubVI must be passed to any SubVIs/ functions which attempt to acquire data from the keyboard. The "Device ID" can be seen passing into the "device ID" input of the "Acquire Input Data.vi" SubVI, as well as passing out of the image to additional components of the Nano-Drive VI.

Acquire Input Data.vi SubVI
This SubVI retrieves data from the device referenced by the "Device ID". In this case the device is the keyboard and the data retrieved is stored within an array of U16 (Unsigned 16-bit) integers. The "Acquire Input Data.vi" SubVI fills the array with unique numeric representations for each keyboard button pressed.

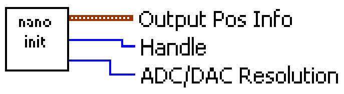
The Index Array Function
A value contained within the array generated by the "Acquire Input Data" SubVI (representing a keyboard button has been pressed) will appear within element '0' of the array. When multiple buttons are simultaneously pressed, elements '1', '2', '3', etc…, will hold the additional values. The "Index Array" function is used here to retrieve the value of the first element, element '0', of the array. Because this process exists within a rapidly looping structure, the VI maintains an accurate and rapidly updating account of which keyboard button it must respond to.

Process 2 – Initiate the Nano-Drive®



Init.vi SubVI

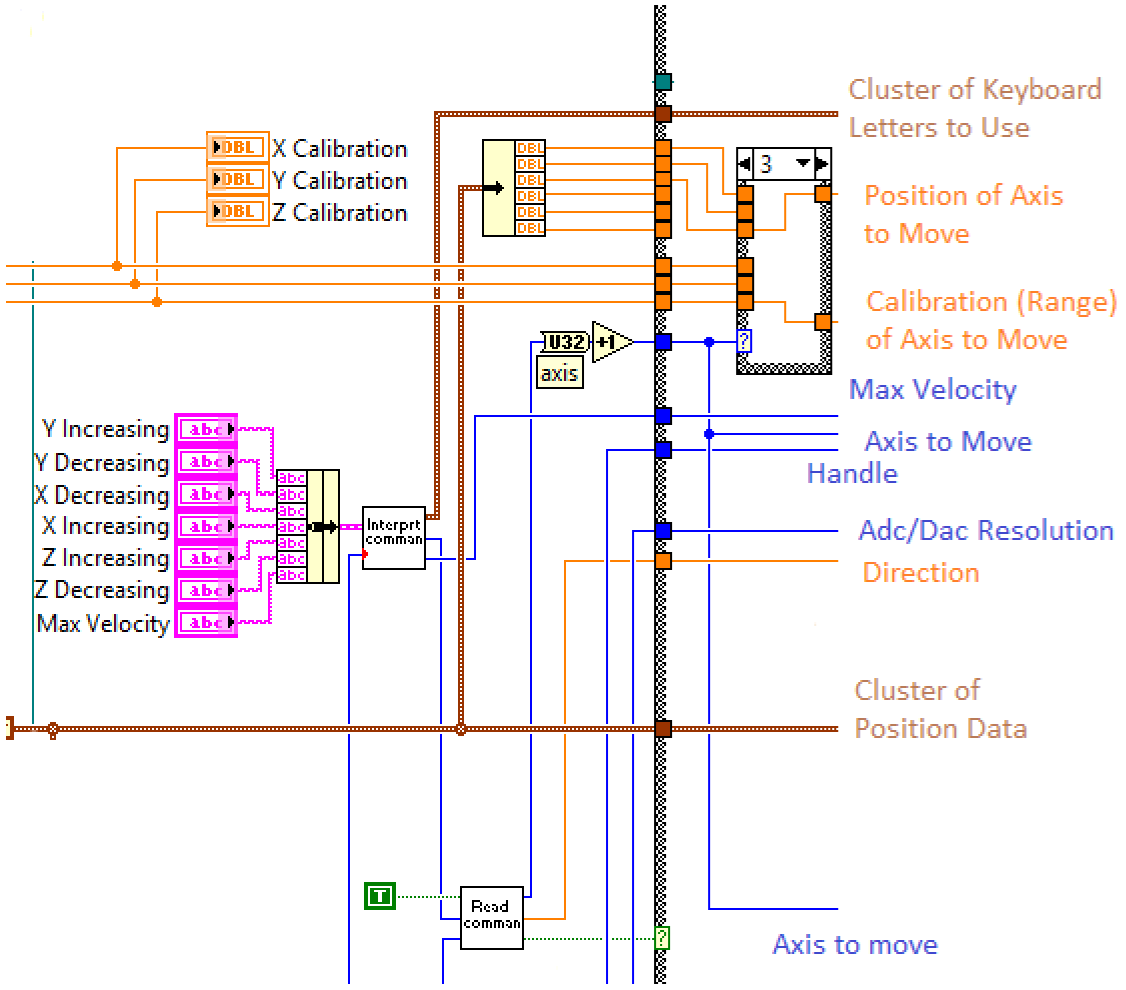
The Init.vi SubVI initializes and acquires a "Handle" for a Nano-Drive® controller. The SubVI then reads the position of each axis of the Nano-Drive® and outputs a "Cluster" containing this data. The SubVI also obtains the resolution of the Analog-to-Digital and Digital-to-Analog converters (i.e. 16-bit, or 20-bit, interface). Lastly, the SubVI acquires the Calibration of each axis (i.e. the range of the axis.)



Initializing Shift Registers with Position Data

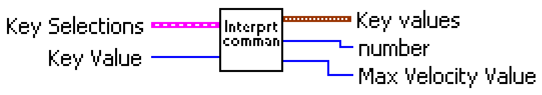
The output cluster from Init.vi containing the X, Y, Z positions is wired to a shift register. The cluster is unbundled inside the while loop to update the Front Panel "Indicators" with the current positions. The cluster has its values modified by later processes in the while loop; using the Shift Register makes sure that each while loop iteration has the updated position data to display to the indicators.

Process 3 – Translate Data from the Keyboard into Motion Commands



GetKeyboardInfo.vi SubVI

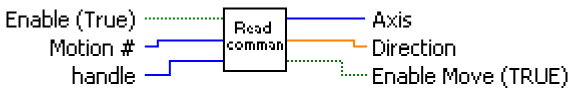
The "GetKeyboardInfo" SubVI allows the user to set which keyboard buttons will move the stage. This is done by choosing a value for each of the seven string controls (seen above). The GetKeyboardInfo SubVI compares the value retrieved from the keyboard to the values of the string controls. The GetKeyboardInfo SubVI determines if a motion command has taken place, and passes this data to the "ReadCommand" SubVI.



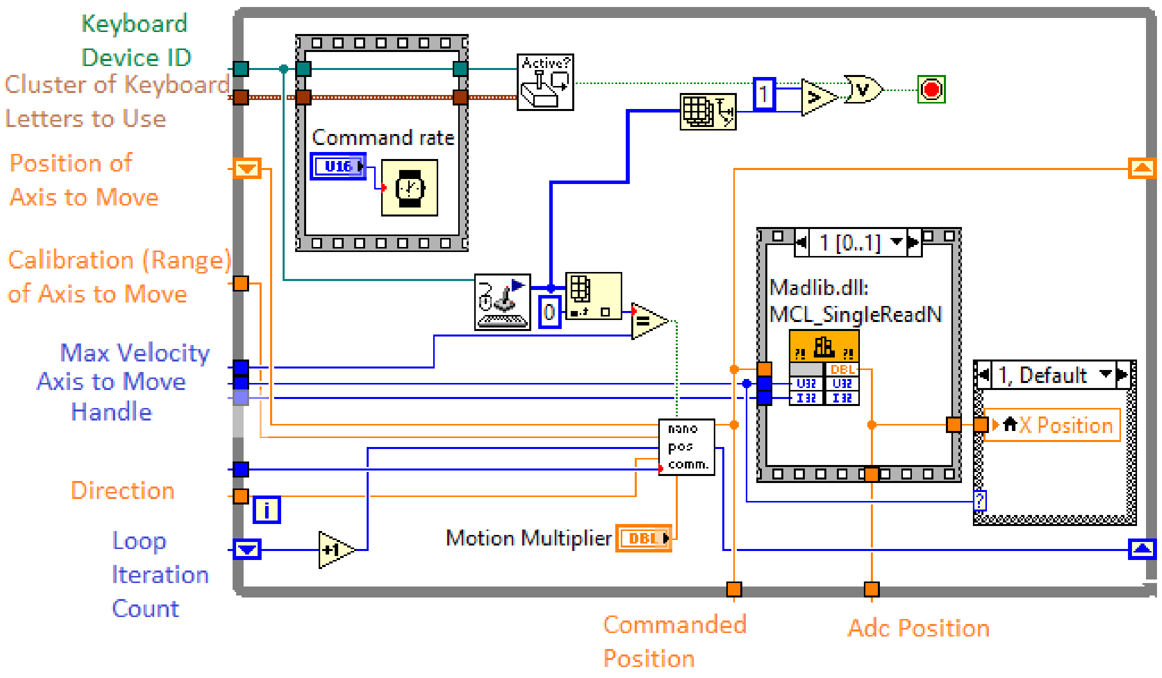
ReadCommand.vi SubVI

The "ReadCommand" SubVI receives a number from the "GetKeyboardInfo" SubVI into its "Motion #" input which corresponds to the axis and direction of the occurring motion command.

The ReadCommand SubVI decodes the number received at its Motion # input into motion command parameters usable by Madlib.dll functions.

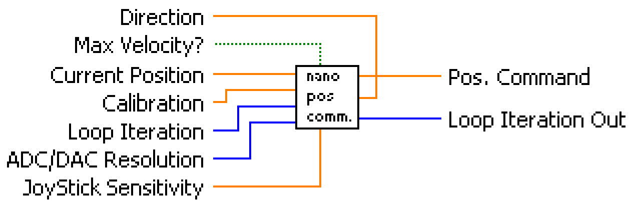


Process 4 – Move the NanoDrive®



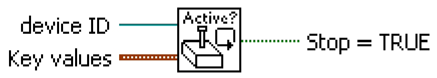
Issue Command.vi SubVI

The “IssueCommand” SubVI determines the position the commanded axis must move to. The decision is initially based upon two factors; the current position of the axis and the direction the axis has been commanded to move. However, as the “While Loop” rapidly cycles, the IssueCommand SubVI counts the iterations to create an acceleration ramp.

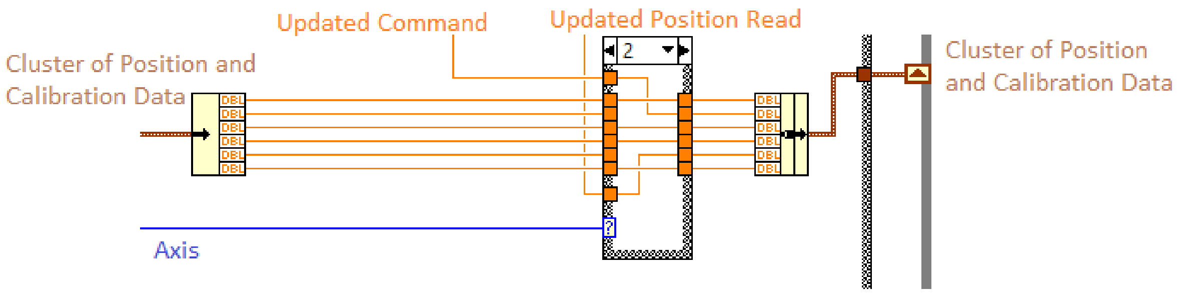


The KeysPressed.vi SubVI

The purpose of the “KeysPressed.vi” SubVI is simple; it does not allow the VI to progress until the user has released all relevant keyboard buttons; i.e., the Game Pad must be returned to its neutral position. When all relevant keyboard buttons have been released the Boolean value "TRUE" is passed to the "Stop Terminal" of the "While Loop".



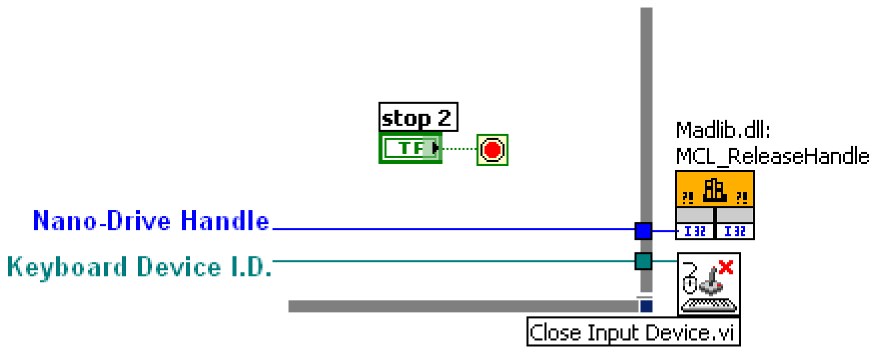
Process 5 – Update Position Display Data


Updated Command
Updated Position Read
Cluster of Position and Calibration Data
2
DBL
Cluster of Position and Calibration Data
Axis

Updating Position Display Data
Three cases exist for the "Case Structure" seen within the image above. The Case Structure reacts to the active axis of the Nano-Drive®; i.e., after an axis has been moved this Case Structure guarantees the corresponding position display on the Front Panel is updated.

Passing Data through Shift Registers
After exiting the "While Loop" of Process 4, and updating the position data, the VI returns to waiting for a command from the Game Pad/Keyboard. Position data is retained throughout the life of the VI via the "Shift Registers" which carry data through successive iterations of the larger While Loop.

Process 6 – Allow User to End the VI



The Stop Button
Pressing the grey "Stop" button stops the "While loop" from looping, beginning the end of the VI. Note that the image above shows the "Handle" of the Nano-Drive® and "Device I.D." of the Game Pad/Keyboard passing out of the While Loop into two separate functions. This occurs only after the Stop button has been pressed.

The MCL_ReleaseHandle and Close Input Device.vi Functions
After the user has pressed the grey "Stop" button to end the VI, the handle for the Nano-Drive® passes out of the "While loop" and into the "MCL_ReleaseHandle" function. Similarly, the "Device ID" used to obtain data from the keyboard is passed out of the While loop and into the "Close Input Device.vi" SubVI. These functions release control of the two devices.