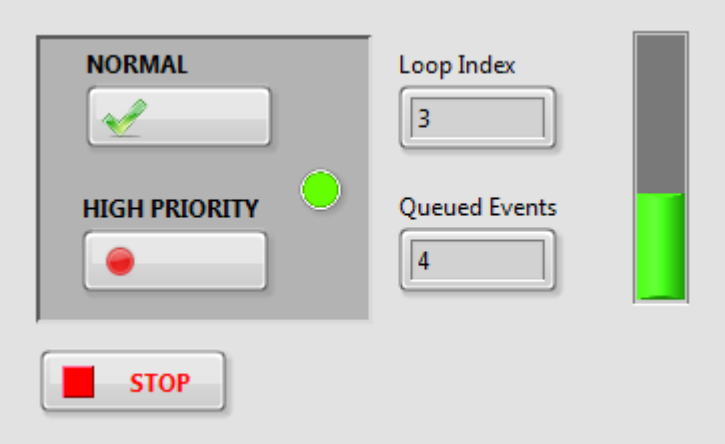
PHYC 500: Introduction to LabView

M.P. Hasselbeck, University of New Mexico

Exercise 15 (v 1.2) Producer-Consumer Event Structure

This exercise extends the producer and consumer queuing action to include events generated in the Event Structure. User inputs on the Front Panel are queued and sequentially processed, including a high priority option to put certain designated events at the front of the line.

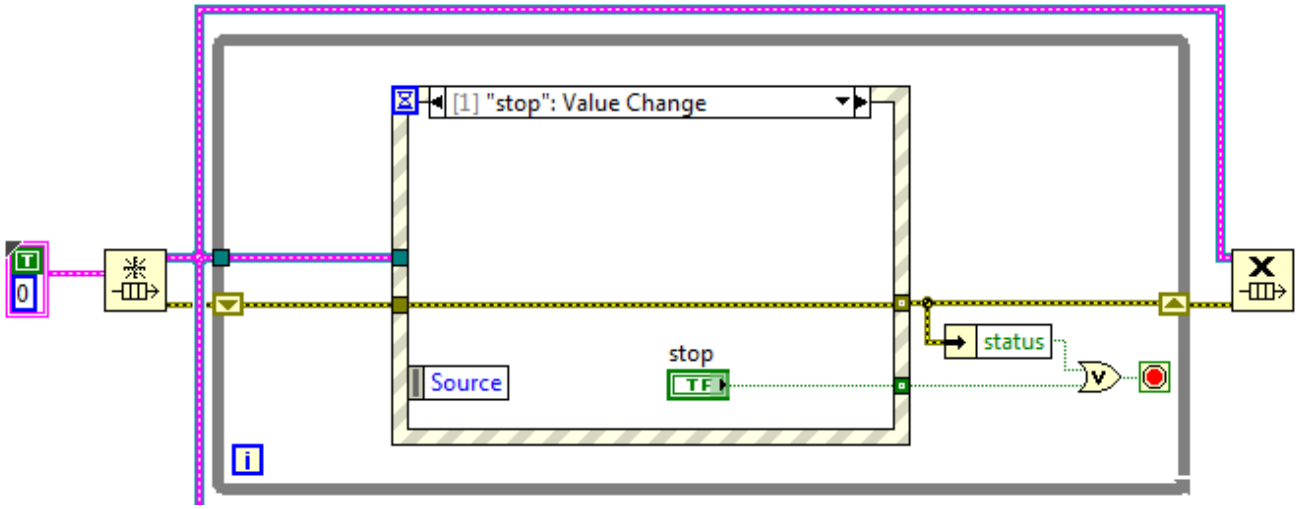
Open LabView and select Producer-Consumer-Events from the available templates. Go to the Front Panel and remove the 'Enqueue Element' Boolean control. Add two Boolean controls and label them 'Normal' and 'High Priority'. Adjacent to these buttons place a single LED; it will flash 3 times at 2 Hz with a different color depending on which button is pushed. Also place two numeric indicators with I32 representation. One keeps track of the queued events and the other updates the producer loop index. Wire the queued event counter to a vertical progress bar and set its scale to 0—10. This is how the Front Panel may look using the Silver palette with the VI running:



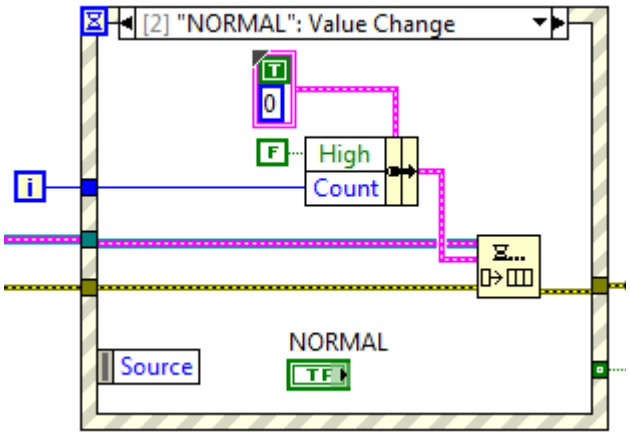
Go to the Block Diagram and edit the producer loop Event Structure. You will need three separate event cases to look for a value change when any of the three Front Panel buttons is pressed. Review Exercise 3 for the procedure to configure events. Drag the Boolean icons inside their corresponding event cases.

Delete the default string constant from the **element data type** input terminal on 'Obtain Queue'. This will be replaced with a two-element cluster containing a Boolean and I32 numeric. Proceed as follows: Create a Cluster Constant on the Block Diagram. Place a Boolean constant and numeric I32 integer constant inside the cluster constant. Right-click on the edge of the Cluster Constant, select Auto Sizing, and Arrange Vertically. Right-click again and select 'Make Type Def'. You should see a dark triangle on the upper-left corner of the icon indicating it is now a Type Def. Open the Type Def to show

its Front Panel and add corresponding labels 'High' and 'Count' to the two components. Apply changes and save as a custom control (.ctl) file. In the Block Diagram, connect the Type Def to **element data type** of 'Obtain Queue'. The Type Def gives the programmer flexibility for future expansion and customization, just as it does in the State Machine.

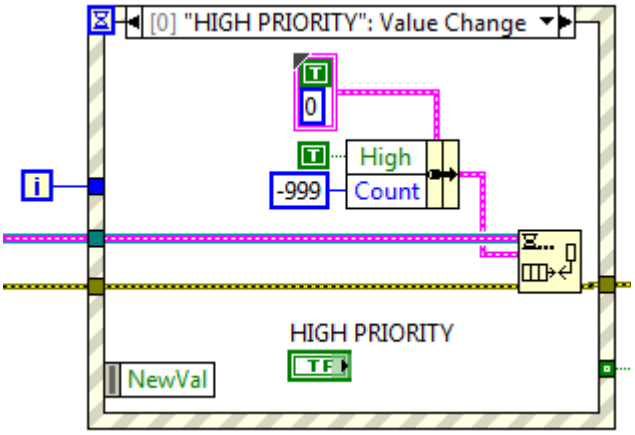


Go to the Normal event case in the producer loop and insert the cluster function 'Bundle by Name'. Make a copy of the Type Def. Cluster constant, drop it in the Event Structure, and connect it to the unwired **input cluster** terminal. Expand the bundle function to reveal the High and Count control components. Connect a FALSE constant to the Boolean input and the While Loop index to the numeric input. Connect the output to the **element** terminal of 'Enqueue Element'. This part of the Block Diagram should resemble the following:



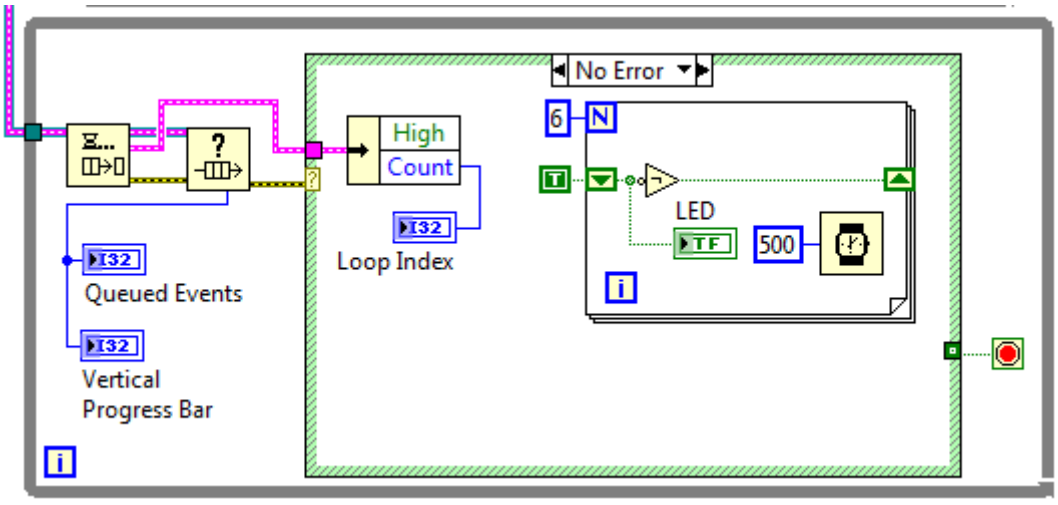
Next, build the High Priority event in the producer loop. It should be configured the same as the Normal event with the following changes: i) Boolean constant is TRUE, ii) Numeric Control is a constant –999, and iii) use instead 'Enqueue Element at Opposite End' function. The latter operation will put this event at the front of the queue, i.e. high

priority events get handled before the normal events in the queue.



The Stop case can remain as in the template.

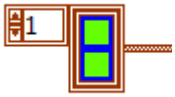
The consumer loop is configured similar to the consumer loop in Exercise 14. Place 'Get Queue Status' next to 'Dequeue Element' outside the Error Case Structure and wire **# of elements in queue** to the corresponding Front Panel indicators (numeric and vertical progress bar). You will have to do some re-arranging to make space. The remaining code goes inside the 'No Error' case. Since the **element** output from 'Dequeue Element' is a cluster, use the 'Unbundle by Name' function to separate the Boolean and numeric data. The numeric data shows the producer loop index or –999 if the High Priority button is pressed. Insert your existing code from previous assignments to flash the LED three times (use a 500 ms Wait).



The color of the LED is automatically set inside the program using a Property Node. Right-click on the LED icon and create a Property Node: Colors [4] and place it outside the For Loop (or While Loop if you used this instead) that generates periodic flashing. Right-click on the Property Node and select 'Change to Write'. The Property Node takes a

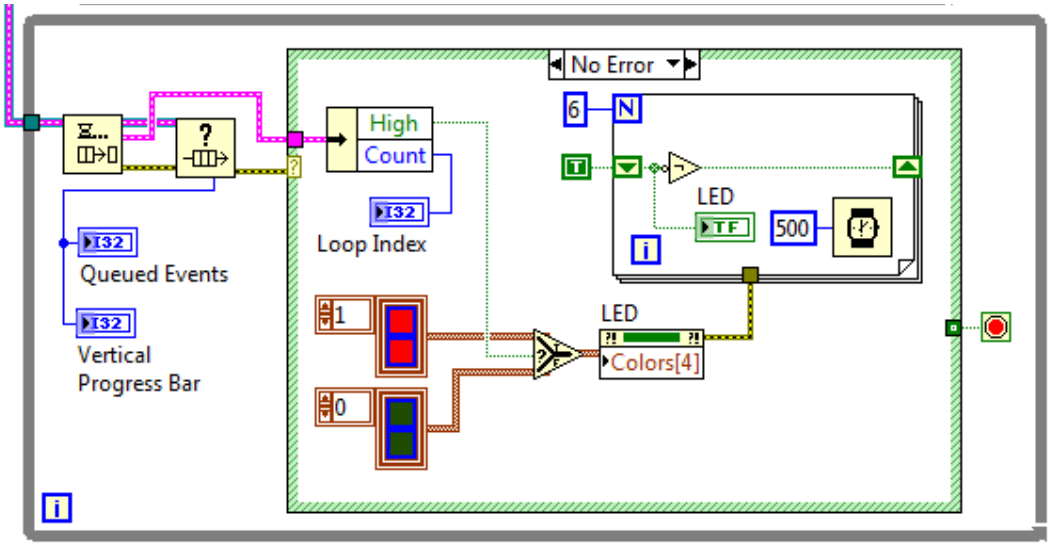
two-element array input with each array element containing a two element cluster. One cluster defines the TRUE state colors and the second defines the FALSE state colors of the LED.

In a blank space, create a Cluster Constant. Navigate to the palette Graphics & Sound: Picture Functions and select a Color Box Constant. Place two color constants inside the Cluster Constant. Right-click on the edge of the cluster and select AutoSizing: Arrange Vertically. Next, create an Array Constant and place the color cluster inside it. Element 0 of the array defines the color of the FALSE case and the element with index 1 is the TRUE case. Two colors are available in the cluster to allow custom shading (foreground and background), but since this is only cosmetic it's easier to make them both the same color. Use the paintbrush tool to make the TRUE state bright green and the FALSE state dark green. This is how the LED will flash when the Normal button is pressed. An example of the TRUE state array element for the green LED is shown here:



Duplicate the array. Use the paintbrush tool to set the TRUE state of the second array (Element 1) to bright red and the FALSE state (Element 0) to dark red. This color combination will alternately flash on the LED when the High Priority button is pressed.

The Boolean data obtained from the queue determines which color scheme appears on the LED. If it is TRUE, the LED flashes bright and dark red; if FALSE it flashes bright and dark green. This can be implemented with the Select function or a Case Structure; the desired array is then connected to the input terminal of the Property Node for the LED. You should see a thick, brown wire indicating a 2-D cluster array. An example of how to setup the Consumer Loop is shown below. Note the use of the Error cluster on the Property Node to eliminate a possible race condition: the For Loop will not run until the desired LED color is established. Once properly assembled, the VI can be tested.



This VI allows for a rapid succession of button clicks – the button events are accumulated (not lost) while the LED slowly flashes. The Producer-Consumer-Event structure maintains a responsive GUI while the VI is busy performing other tasks in parallel. Notice that neither of the While Loops are polling with a Wait function; the producer loop remains inactive until a button is pressed and the consumer loop is dormant until data appears in the queue. This makes for a very efficient use of computer resources. When the Producer Loop is stopped, an error propagates down the queue and stops the Consumer Loop.