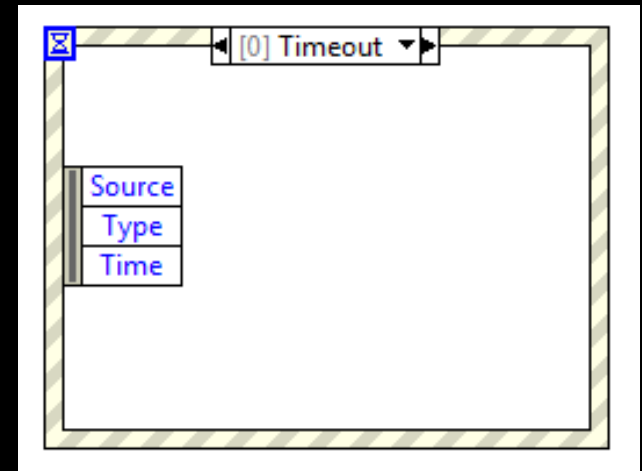


# Event Structures and Boolean Mechanical Action



# Event Structures

Event structures are used to trigger a section of code when the user performs some event or action on the front panel.

For example, we could have a button on the front panel that causes code to implement when the user pushes some button.

The most common event is “Value Change.”

# Event Structure vs. Polling

Polling is the act of your program constantly checking to see if some value has changed (maybe checking every 500 ms). This uses CPU power. 😞


Event structures act more like interrupts. It does not use any CPU resources until it needs to (e.g. the user makes some change on the front panel).

So you can design your programs to run very efficiently. **You should always use event structures instead of polling if you can!**

# Different Types of Events

- Timeout: *triggers an event if the user doesn't perform an action in some specified time. Works similarly to having a wait (ms) in a while loop.*
- Value Change: *user changes the value of some control*

## Less Common Events

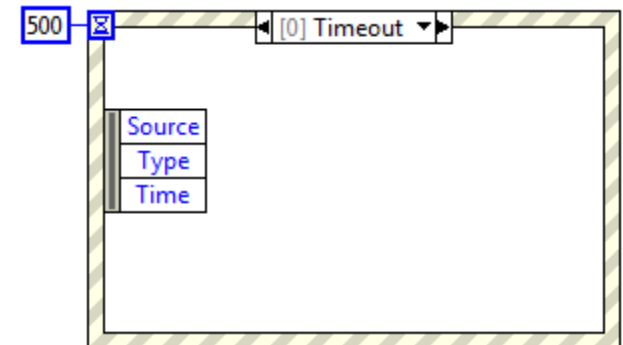
- Application Instance Close: *user hits the X at the top right to close the program – unfortunately, it doesn't work for abort button* 
- Key Down/Up: *user presses or releases a keyboard key*
- Mouse Down/Up: *user presses or releases the mouse button*

# Timeout Event

The timeout event only triggers if the user doesn't perform some action within a specified amount of time. In the picture, this specified time is 500 ms.

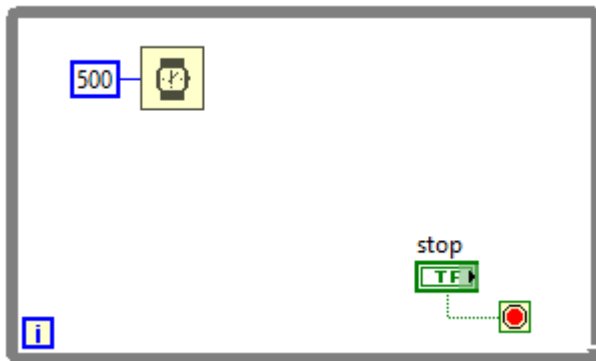
The timeout event is nice if you need to update some indicator or control on the front panel periodically. For example, you may update a graph periodically.

Note: The timeout event always shows up by default and is defaulted to an infinite timeout (the value being fed in is -1), but you do not have to have a timeout event and often times you won't. To remove, you right-click and choose "Remove this Event Case."

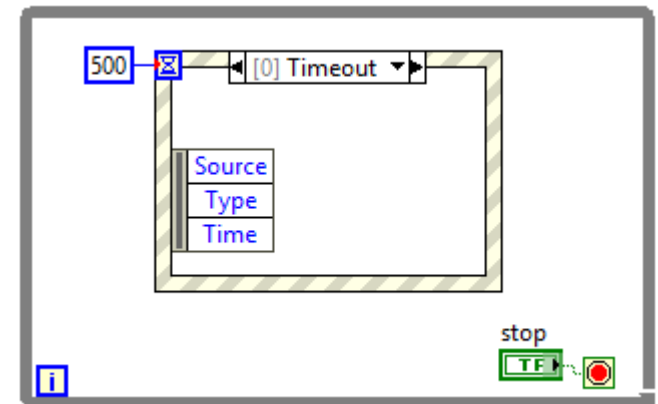


# Timeout Event

If the timeout event is inside a while loop (as shown below), then it behaves almost identically to using the wait(ms) function. Therefore, don't include a wait(ms) function additionally to a timeout function.

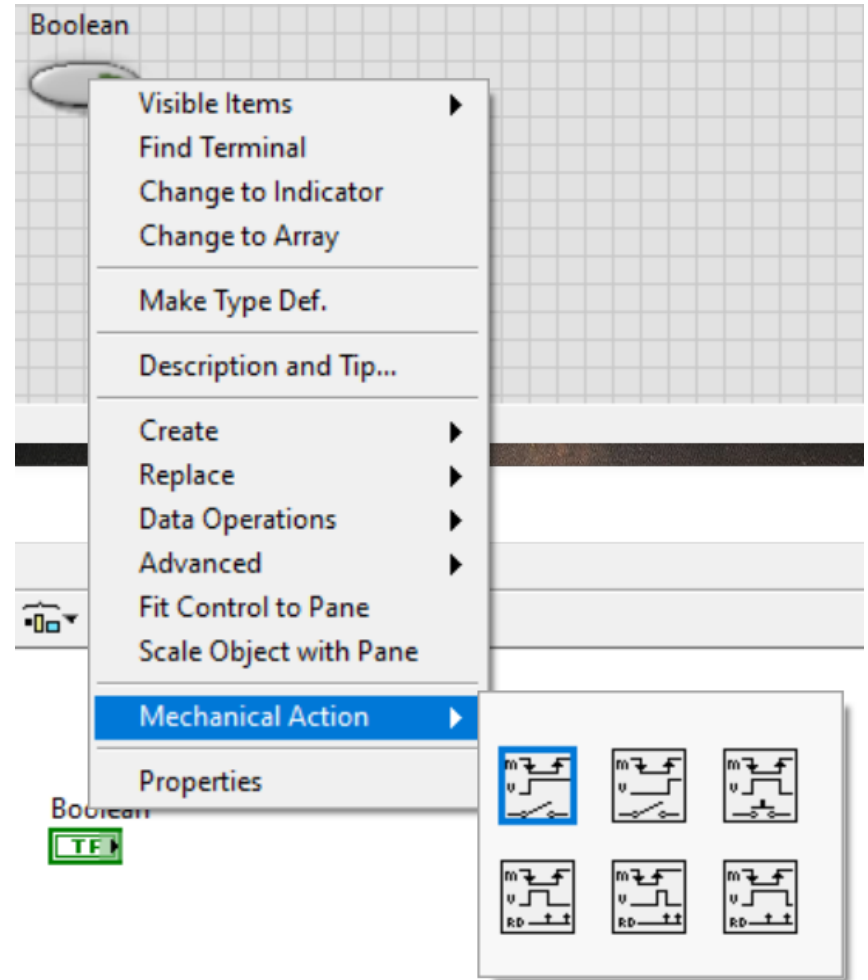


←Behave the same way→



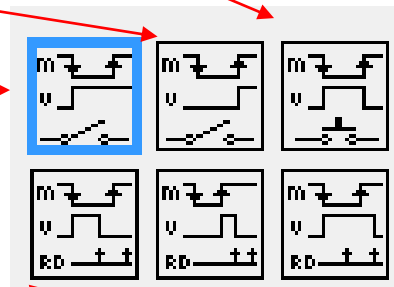
# Mechanical Action of Boolean Buttons

Before I continue on with Event Structures, let's switch to Mechanical Action of Buttons



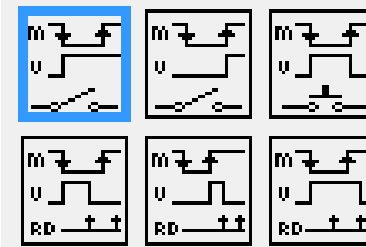
# Mechanical Actions of Boolean Objects

- Switch until released (door buzzer)
- Switch when released (mouse)
- Switch when pressed (light switch)
- Latch when pressed (circuit breaker)
- Latch when released (dialog box)
- Latch until released





# Mechanical Actions of Boolean Objects



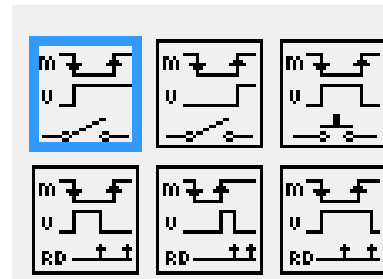
- **Switch when pressed**—Changes the control value each time you click it with the Operating tool, similar to a light switch. This behavior is not affected by how often the VI reads the control.
- **Switch when released**—Changes the control value only after you release the mouse button during a mouse click within the graphical boundary of the control. This behavior is not affected by how often the VI reads the control.
- **Switch until released**—Changes the control value when you click it and retains the new value until you release the mouse button. At this time, the control reverts to its default value, similar to the operation of a door buzzer. This behavior is not affected by how often the VI reads the control.
- **Latch when pressed**—Changes the control value when you click it and retains the new value until the VI reads it once. At this point, the control reverts to its default value even if you keep pressing the mouse button. This behavior is similar to a circuit breaker and is useful for stopping a While Loop or for getting the VI to perform an action only once each time you set the control.
- **Latch when released**—Changes the control value only after you release the mouse button within the graphical boundary of the control. When the VI reads it once, the control reverts to its default value. This behavior works in the same manner as dialog box buttons and system buttons.
- **Latch until released**—Changes the control value when you click it and retains the value until the VI reads it once or you release the mouse button, depending on which one occurs last.

See "[Mechanical Action.vi](#)"

# CLAD Question

**Q27:** Which Mechanical Action changes a Boolean when the button is pressed and returns it to its default value after LabVIEW reads the value?

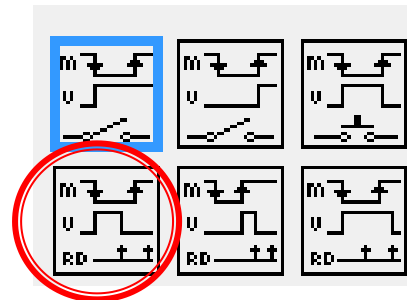
- A** Latch when pressed
- B** Switch when released
- C** Switch until released
- D** Latch when released



# CLAD Question

**Q27:** Which Mechanical Action changes a Boolean when the button is pressed and returns it to its default value after LabVIEW reads the value?

- A** Latch when pressed
- B** Switch when released
- C** Switch until released
- D** Latch when released



# Event Structures Come at a Cost to the Programmer

Although event structures are really nice, they can be difficult to implement.

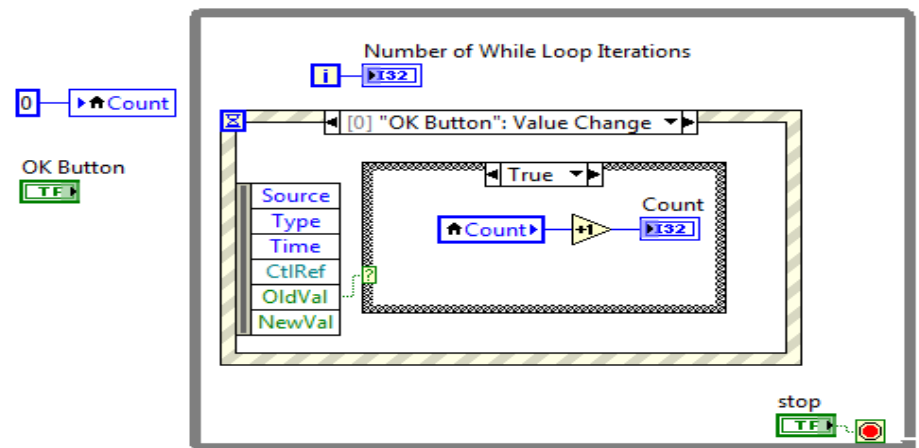
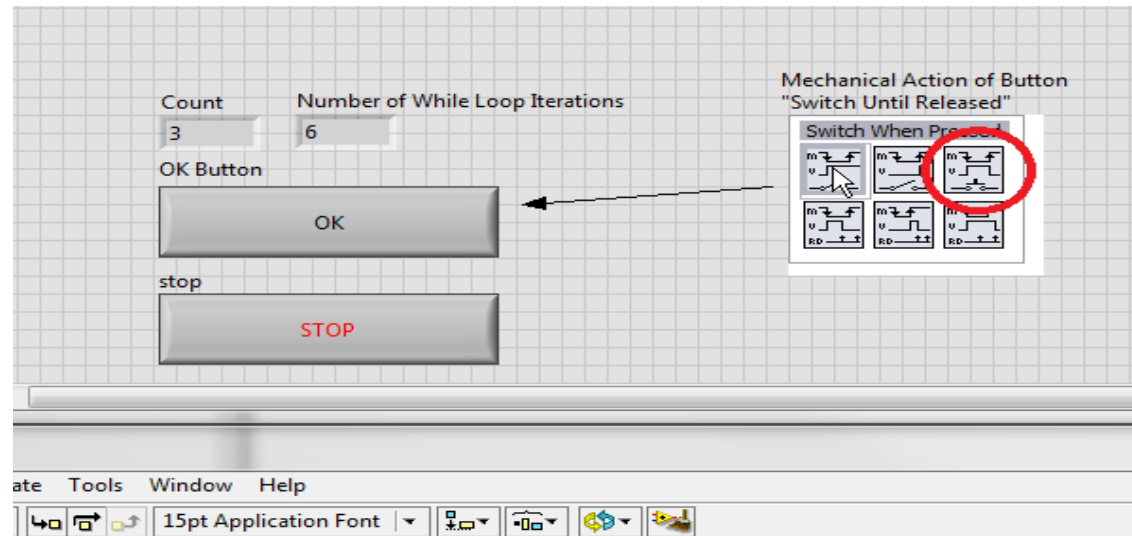
For example, if you do a Value Change event with a Boolean button that has a mechanical action that goes down and up ("*Latch Until Released*" is the actual name), then the event will trigger twice instead of once. It triggers once for the down and once for the up. See *next slide*.

# Event Structures – Handling Boolean Switch Value Change

To prevent the event structure from triggering twice from counting both the up and down motion of a single button press, I usually implement a **case structure inside the event structure**.

Note: The “false” case is empty.

Notice that the while loop iterates twice as many times as the number of button presses.

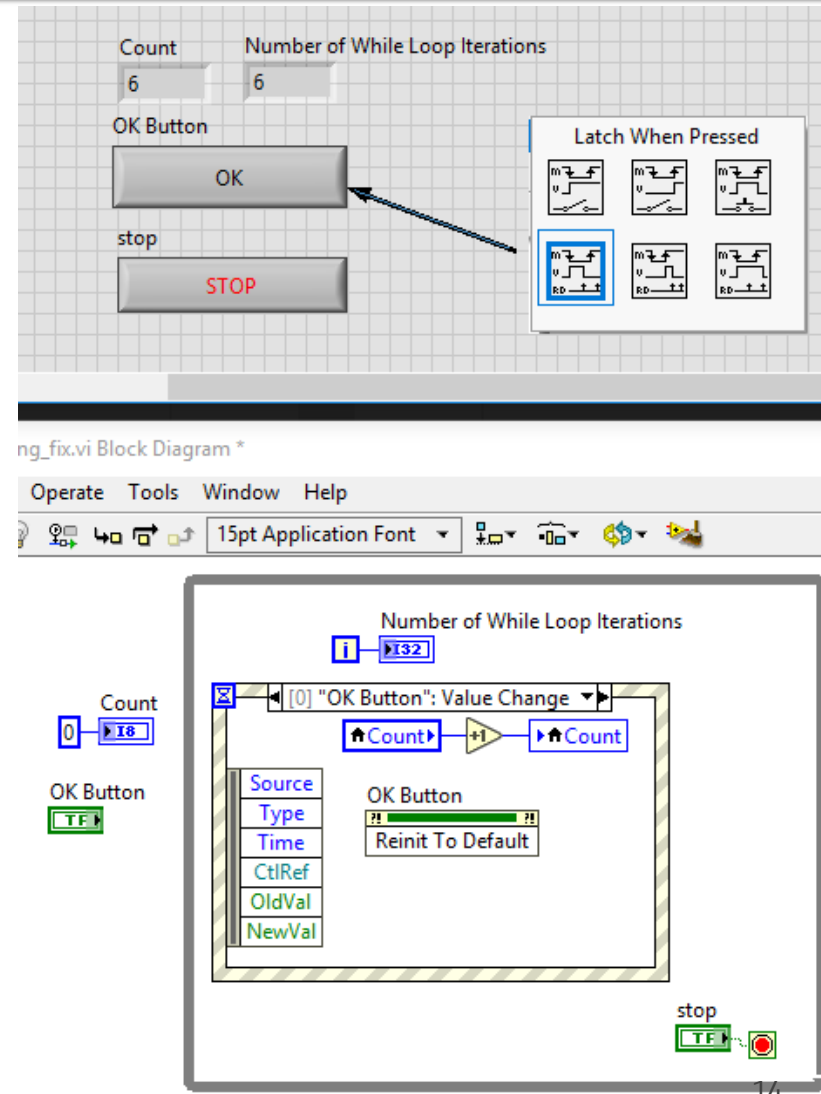


See "[Event Structure: Handling Boolean Push Button.vi](#)"

# Event Structures – Handling Boolean Switch Value Change

Another option that a student came up with involves implemented the invoke node for the Boolean button of Reinitializing to Default in the event case. You also need to change the mechanical action to Latch When Pressed.

This is a bit cleaner and the number of While Loop Iterations matches the number of times the OK Button is pressed.



# Event Structures Come at a Cost to the Programmer

Another frustrating thing is that if you are using **2 event structures in 1 loop structure**, then after one event structure implements it will **lock the front panel** by default until both event structures have fully implemented.

Locking the front panel won't allow the user to click on anything and it feels like the computer is frozen.

This doesn't work well for state machines that have inputs in multiple states. We'll discuss state machines in another lecture.

You can uncheck "Lock Front Panel" to prevent this.

# Event Structures Come at a Cost to the Programmer

Again, another frustrating thing with Event Structures is if you have 2 event structures in one loop structure where both look at the same event for one control, **then both event structures will implement** with one user interface event. 😞

For example, let's say we have two event structures in one loop that handle the value change of some Boolean button. If the user changes the value of the button, then both event structures will handle the event independently.



# Good Guidelines for Event Structures

1. Only use one event structure in a single loop structure.
2. However, if two event structures are needed in one loop, then don't have the same events in each (ex: same Boolean button) and uncheck "Lock Front Panel" for every event in both structures.

# Control Events (from LV help)

## Control Events

**Requires:** Base Development System

View the [class hierarchy](#).

Event	Description
Drag Ended	Generated after a drag and drop operation completes. <a href="#">Details</a>
Drag Enter	Generated when there is a drag operation pending and the cursor enters the bounds of a control. <a href="#">Details</a>
Drag Leave	Generated when there is a drag operation pending and the mouse leaves a control that can accept a drag, or if a user cancels a drag and drop operation when hovering over a control. <a href="#">Details</a>
Drag Over	Generated when there is a drag and drop operation pending, as the mouse moves over a control. <a href="#">Details</a>
Drag Source Update	Generated when the mouse moves or a key state changes during a drag and drop operation. <a href="#">Details</a>
Drop	Generated when the user lifts the mouse when hovering over a control when there is a drag and drop operation pending. <a href="#">Details</a>
Key Down	Generated on a control that has keyboard focus. If a key press matches a keyboard shortcut in the VI menu, such as Ctrl-C or Ctrl-V, LabVIEW does not generate a Key Down event, regardless of whether the menu item is enabled. <a href="#">Details</a>
Key Down?	Generated on a control that has keyboard focus. If a key press matches a keyboard shortcut in the VI menu, such as Ctrl-C or Ctrl-V, LabVIEW does not generate a Key Down event, regardless of whether the menu item is enabled. <a href="#">Details</a>
Key Repeat	Generated at regular intervals when the user presses and holds a key in a front panel control. <a href="#">Details</a>
Key Repeat?	Generated when the user presses and holds a key in a front panel control. <a href="#">Details</a>
Key Up	Generated when the user releases a key on the keyboard in a specific control on the front panel. <a href="#">Details</a>
Mouse Down	Generated when you click the mouse button on a specific control. <a href="#">Details</a>
Mouse Down?	Generated when you click the mouse button on a specific control. <a href="#">Details</a>
Mouse Enter	Generated when the cursor enters the bounds of the front panel object. <a href="#">Details</a>
Mouse Leave	Generated when the cursor leaves the bounds of the front panel object. <a href="#">Details</a>
Mouse Move	Generated when you move the mouse over a control. <a href="#">Details</a>
Mouse Up	Generated when you release the mouse button on a specific control. LabVIEW does not generate this event if a shortcut menu appears when you click the mouse button. <a href="#">Details</a>
Mouse Wheel	Generated when you scroll the mouse wheel over a control. <a href="#">Details</a>
Shortcut Menu Activation?	Generated when the user right-clicks a control to display the shortcut menu. <a href="#">Details</a>
Shortcut Menu Selection (App)	Generated when the user selects an <a href="#">application item</a> from the shortcut menu of a control. Use the <a href="#">Shortcut Menu Selection (User)</a> event to generate an event when the user selects a user-defined menu item. This event is posted after the built-in shortcut menu item is processed by LabVIEW. <a href="#">Details</a>
Shortcut Menu Selection (User)	Generated when the user selects a <a href="#">user-defined</a> item from the shortcut menu. Use the <a href="#">Shortcut Menu Selection (App)</a> event to generate an event when the user selects an application item from the shortcut menu. <a href="#">Details</a>
Shortcut Menu Selection? (App)	Generated when the user selects an <a href="#">application item</a> from the shortcut menu. This event is posted before the application item is processed by LabVIEW. <a href="#">Details</a>
Value Change	Generated when the user changes the value of a control. You must <a href="#">read the terminal of a latched Boolean control</a> in its Value Change event case. <a href="#">Details</a>

# Avoid Using 2 Event Structures in One Loop

## Avoid Placing Two Event Structures in One Loop

[»Table of Contents](#)

National Instruments recommends that you place only one Event structure in a loop. When an event occurs in this configuration, the Event structure handles the event, the loop iterates, and the Event structure waits for the next event to occur. If you place two Event structures in a single loop, the loop cannot iterate until both Event structures handle an event. If you have enabled front panel locking for the Event structures, the user interface of the VI can become unresponsive depending on how the user interacts with the front panel.

For example, if you place two Event structures in a single While Loop and configure the first Event structure to handle a Mouse Down event and configure the second Event structure to handle a Key Down event, the first Event structure receives a Mouse Down event when the user clicks the mouse button. The first Event structure executes the correct event case and finishes execution. Meanwhile, the second Event structure waits for a key press to occur. When the user presses a key, the second Event structure receives a Key Down event. After the second Event structure handles the event, the While Loop iterates. If the user continues to alternate interactions, generating a Mouse Down event, a Key Down event, a Mouse Down event, a Key Down event, and so on, the VI runs smoothly because the Event structures handle the events as they occur, and the While Loop continues to iterate.

However, if the user clicks the mouse button twice, causing two Mouse Down events to occur sequentially with no intervening Key Down event, the user interface hangs. The first time the user clicks the mouse button, the first Event structure receives a Mouse Down event, handles the event, and finishes execution. However, the second Event structure continues to wait for a Key Down event, and prevents the While Loop from iterating again. When the user clicks the mouse button a second time, LabVIEW generates a second Mouse Down event and locks the front panel until the first Event structure handles the event. At this point, the VI is in a deadlock state. The first Event structure cannot execute until the While Loop iterates, and the While Loop cannot iterate until the second Event structure receives and handles a Key Down event. Because the front panel is locked, no Key Down events can occur. The front panel remains locked and unresponsive until the user aborts the VI.

To avoid hanging the user interface with front panel locking, [configure all events](#) you want a VI to handle in a single Event structure or always make sure there is only one Event structure in a While Loop. Additionally, make sure there is always an [Event structure available](#) to handle events as they occur.

### LabVIEW 2011 Help

**Edition Date:** June 2011

**Part Number:** 371361H-01

[»View Product Info](#)

# CLAD Question

Which of the following will cause an event to be captured by the LabVIEW Event Structure?

- a. Changing a Front Panel control's Value via a mouse click on the control
- b. Changing a Front Panel control's Value property via a property node
- c. Changing a Front Panel control's Value via a control reference
- d. Changing a Front Panel control's Value via a local variable

# CLAD Question

Which of the following will cause an event to be captured by the LabVIEW Event Structure?

- a. Changing a Front Panel control's Value via a mouse click on the control
- b. Changing a Front Panel control's Value property via a property node
- c. Changing a Front Panel control's Value via a control reference
- d. Changing a Front Panel control's Value via a local variable

# Value Change Event (from LV help)

## Value Change Event

**Requires:** Base Development System

**Class:** [Control Events](#)

**Type:** [Notify](#)

Generated when the user changes the value of a control. You must [read the terminal of a latched Boolean control](#) in its Value Change event case.

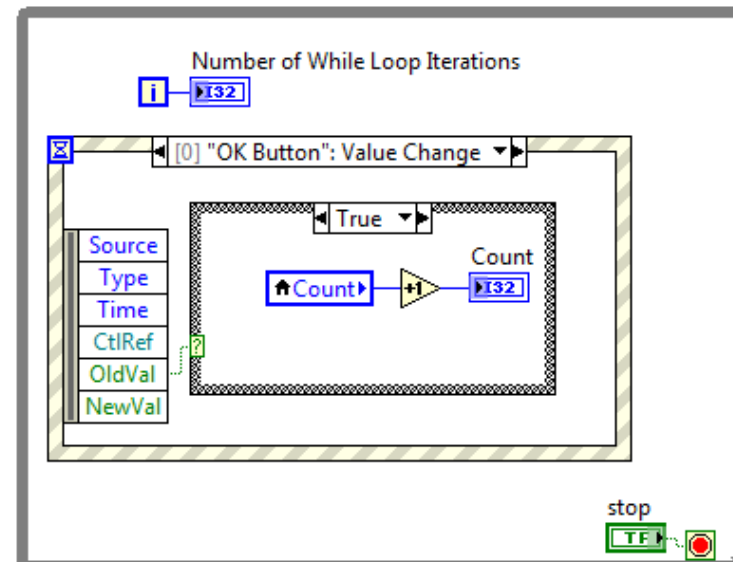
LabVIEW generates this event even if the user enters the same value as the current control value. When used with a [slide control](#), LabVIEW generates all intermediate values of the slide each time a user changes the value, including values that register before the user releases the mouse.



If you click a [combo box control](#) or [I/O control](#) but do not change the value, LabVIEW generates this event.

## Event Data Fields

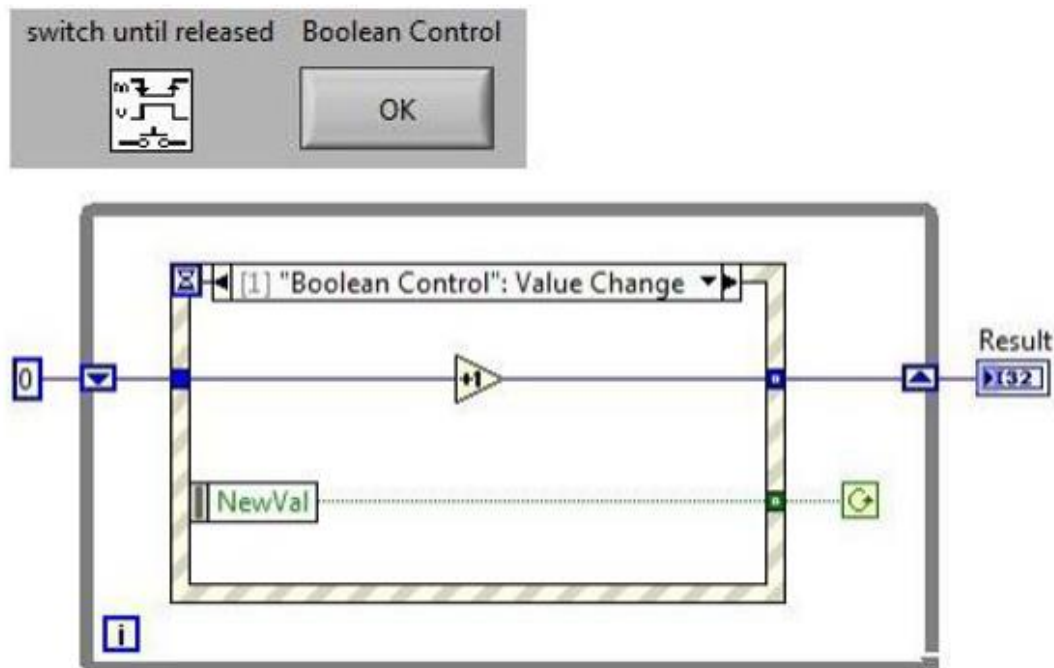
Name	Description
Source	<a href="#">Source</a> of the event. LabVIEW UI refers to any built-in user interface event. <div>0 LabVIEW UI</div>
Type	Type of event that occurred, such as Mouse Down, Value Change, Timeout, and so on.
Time	Value of the millisecond timer when the event occurred.
Time	Value of the millisecond timer when the event occurred.
OldVal	Value of the control before the data change.
NewVal	Value of the control after the data change.



# CLAD Question

**Q11:** While the VI executes, the user presses and then releases **Boolean Control** with Switch Until Released mechanical action. The starting value of **Boolean Control** is FALSE,

What value is displayed in the **Result** indicator after execution?



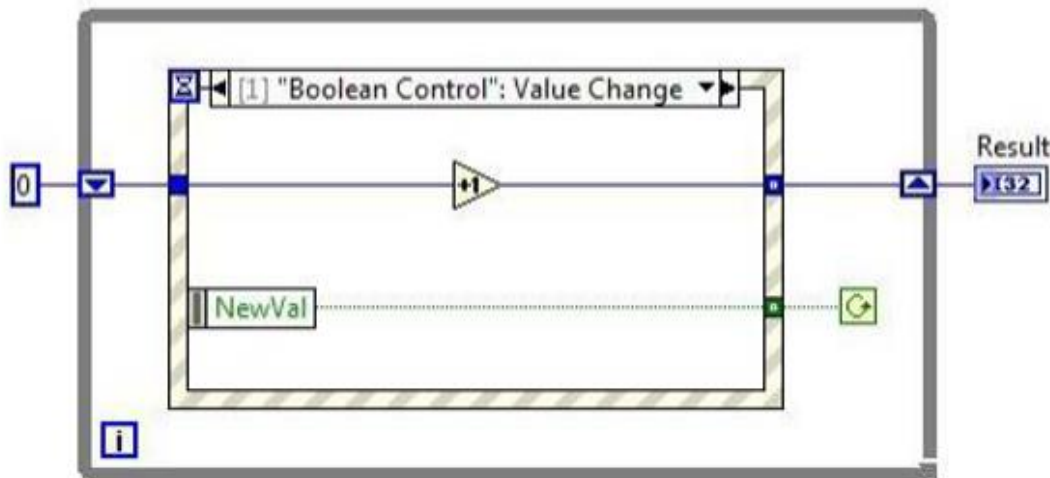
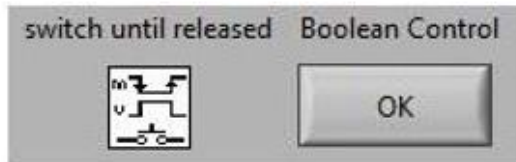
- A 0
- B 1
- C 2
- D 3



# CLAD Question

**Q11:** While the VI executes, the user presses and then releases **Boolean Control** with Switch Until Released mechanical action. The starting value of **Boolean Control** is FALSE,

What value is displayed in the **Result** indicator after execution?



The user pushes and releases the button which causes two changes in the button (F T and T F). This initially causes a value change to True event which triggers the event structure, which increments the 0 to 1. Since the conditional is set to Continue if True, the while loop continues. Now the program has time to register the release of the button which is a change from True to False. This also triggers the event structure and the new value of False will stop the While loop since it's set to Continue if True. Therefore, the event structure is triggered twice (increments twice) and result becomes 2.

- |   |   |
|---|---|
| A | 0 |
| B | 1 |
| C | 2 |
| D | 3 |

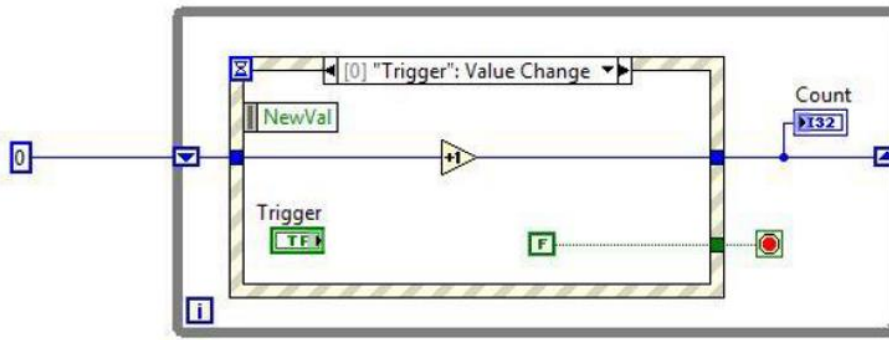


# CLAD Question

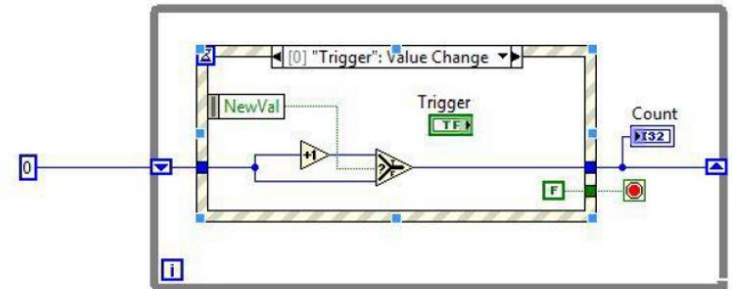
**Q2:** The **Trigger** control is configured with a switching mechanical action. The VI's requirement is to display a **Count** value that tracks the number of "value change" events, for changes to TRUE, that occur for the **Trigger**.

Which of the code snippets meets that requirement when the VI is run?

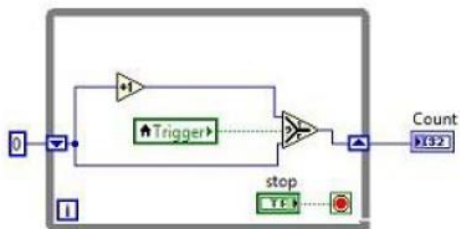
A



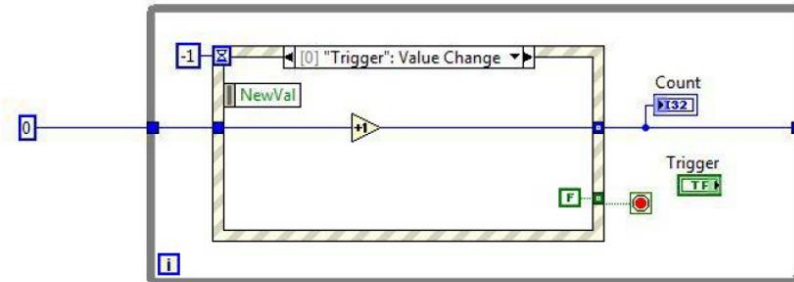
C



B



D

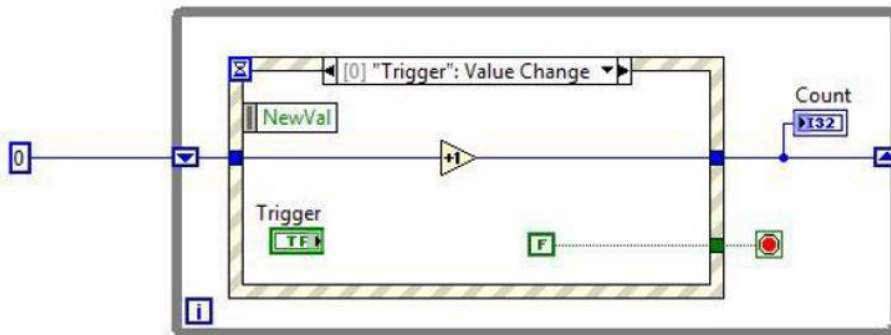


# CLAD Question

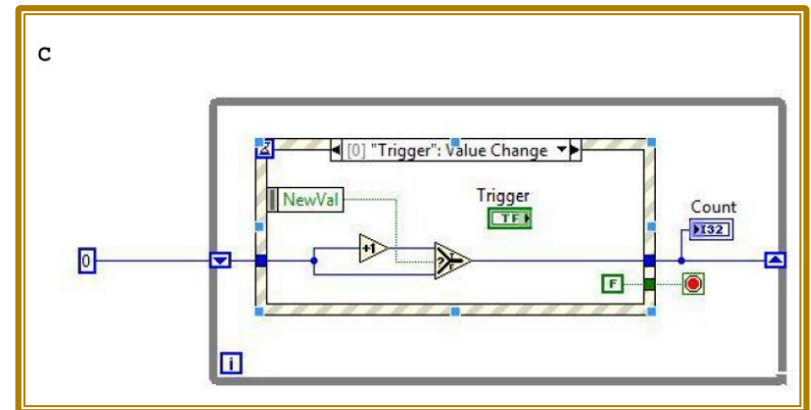
**Q2:** The **Trigger** control is configured with a switching mechanical action. The VI's requirement is to display a **Count** value that tracks the number of "value change" events, for changes to TRUE, that occur for the **Trigger**.

Which of the code snippets meets that requirement when the VI is run?

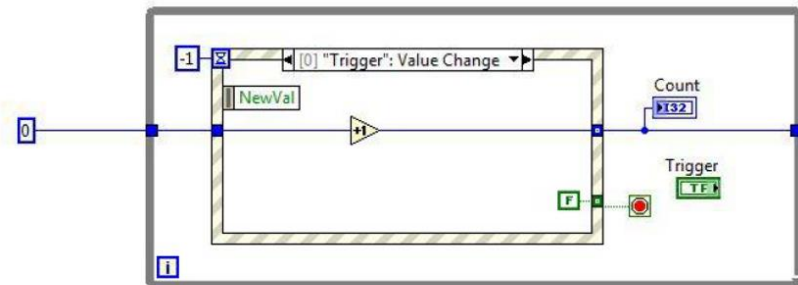
A



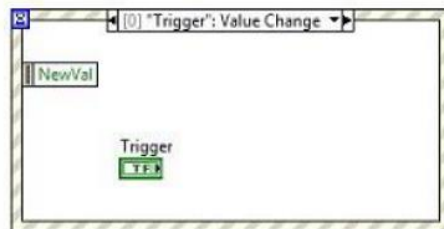
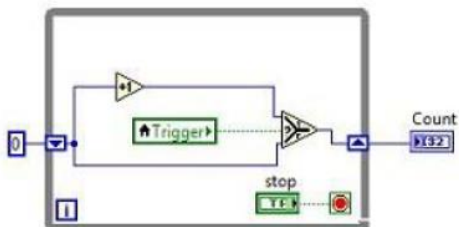
C



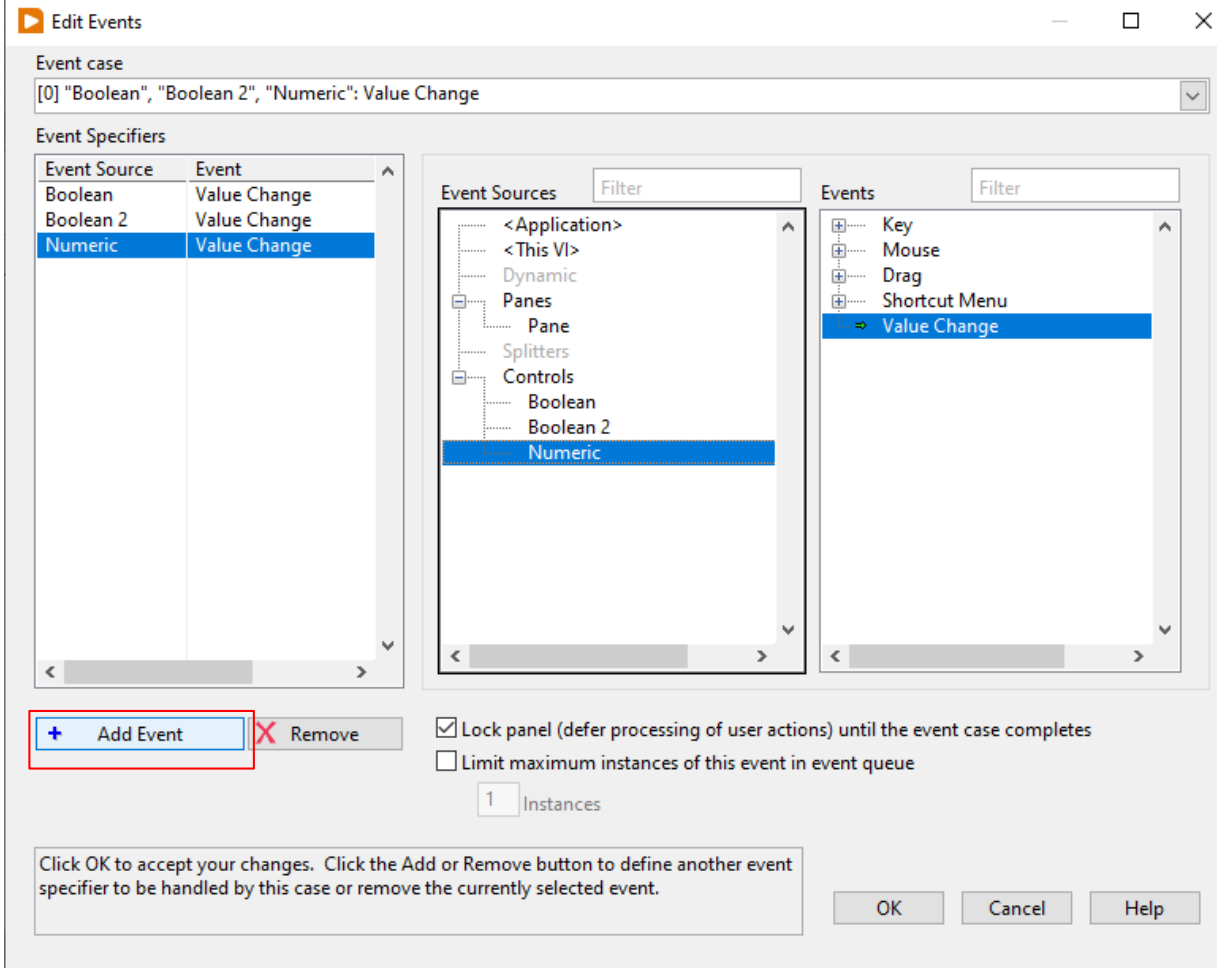
D



B



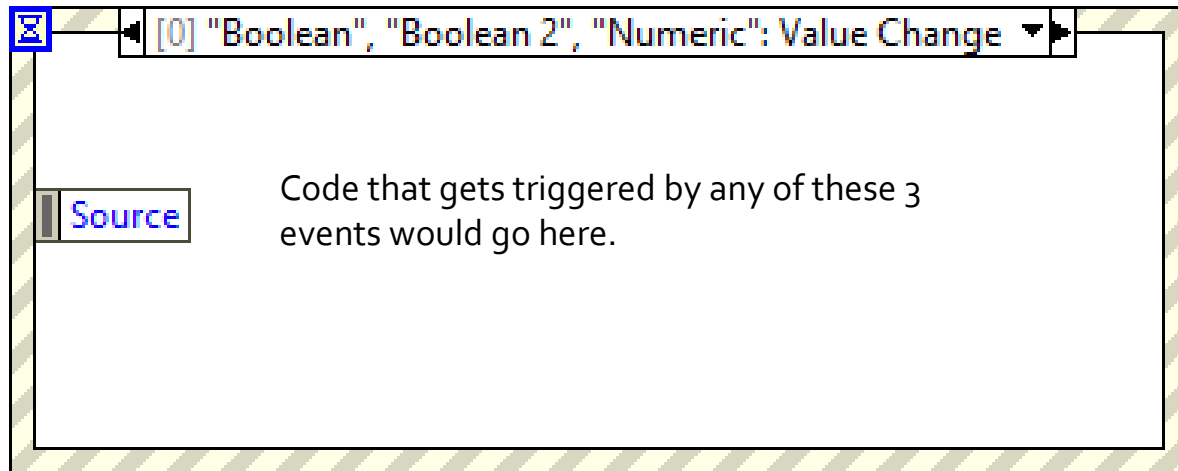
# Multiple Events Can Trigger the Same Event Structure Case



You can have multiple events trigger the same event structure case by clicking on "Add Event." This example has Boolean, Boolean2, and Numeric all trigger and the same code.

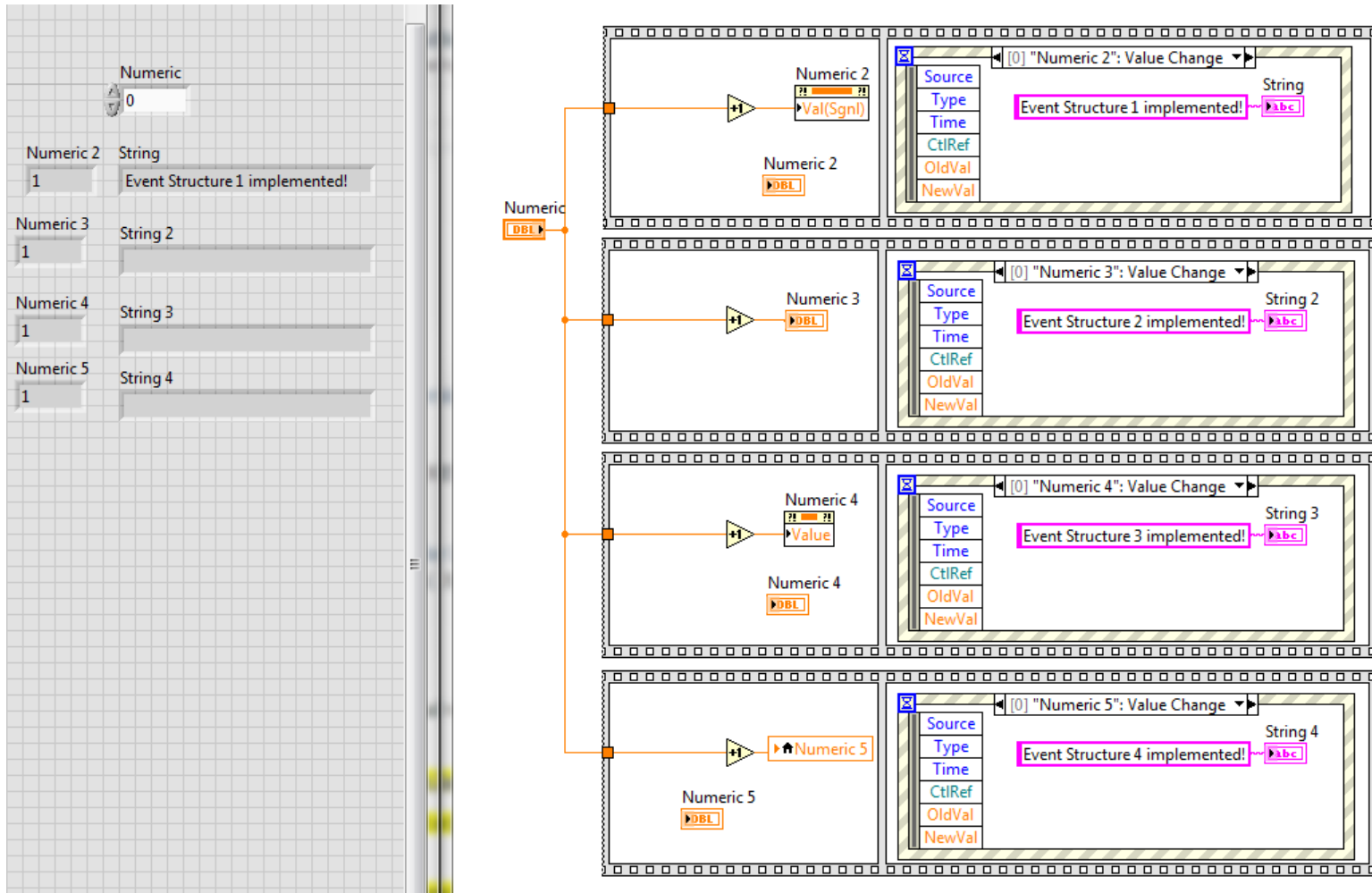
# Multiple Events Can Trigger the Same Event Structure Case

The resulting event structure looks like this.



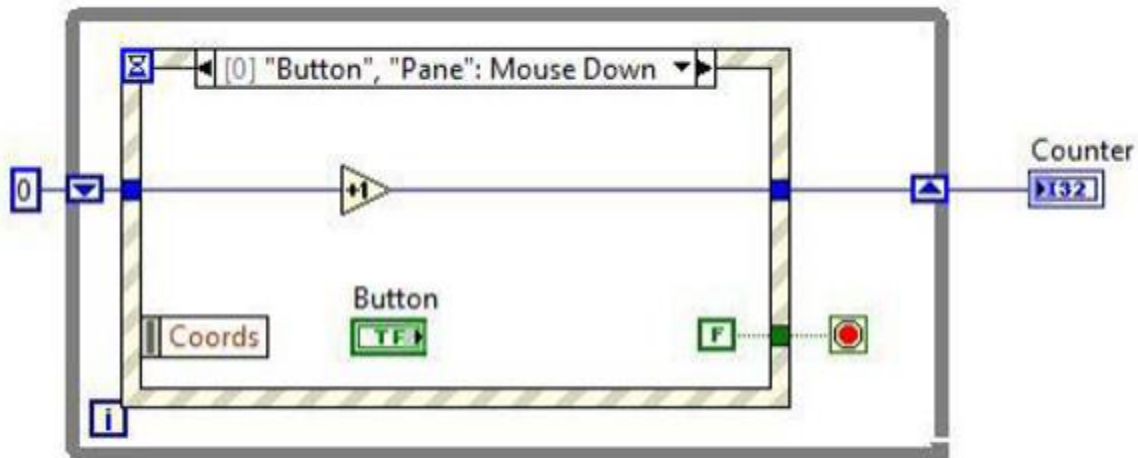
# Triggering Event Structures from Code

To trigger event structures from code, you have to use the property node called "Value (signaling)". Notice only the top structure Implemented and none of the other structures did.



# CLAD Question

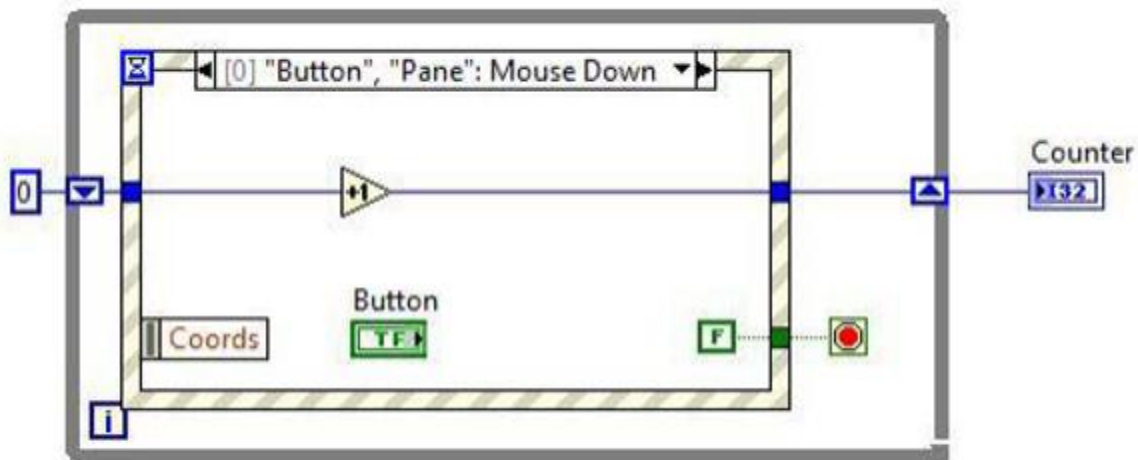
34. When the user clicks the **Button** control, how many times is the **Increment** function called?



- a. 0
- b. 1
- c. 2
- d. 3

# CLAD Question

34. When the user clicks the **Button** control, how many times is the **Increment** function called?



- a. 0
- b. 1
- c. 2
- d. 3

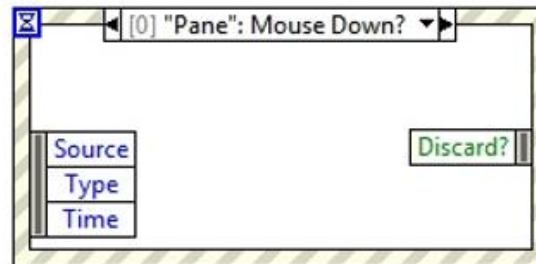
The event is triggered twice since the user pushes the mouse down on the button and on the panel.

# CLAD Question

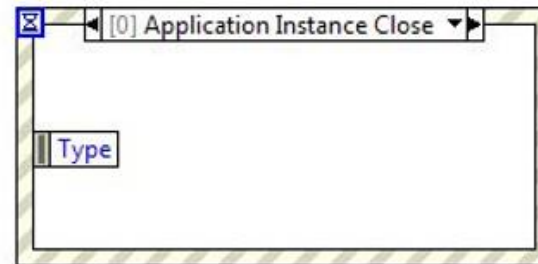
**Q1:** You must include the option to cancel when a user attempts to interactively close the front panel by selecting File>>Close.

Which Event case allows this functionality?

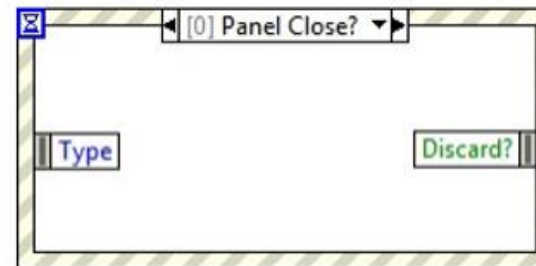
**A**



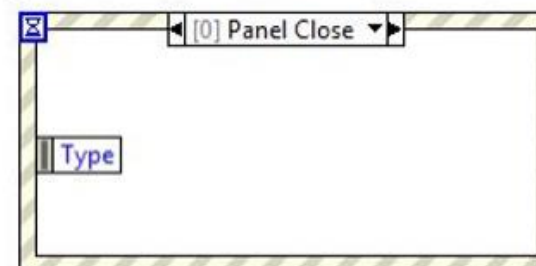
**B**



**C**



**D**



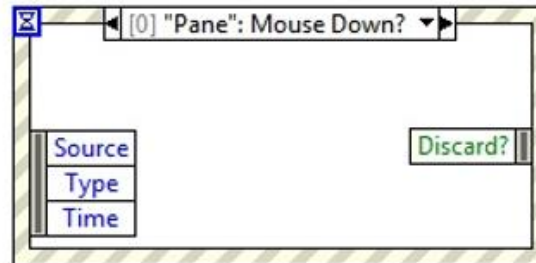


# CLAD Question

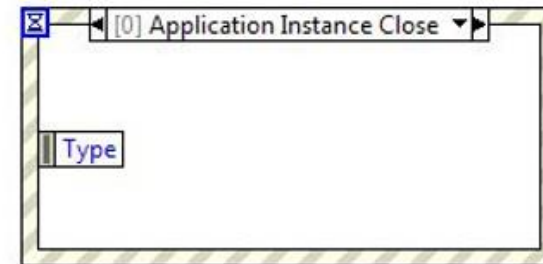
**Q1:** You must include the option to cancel when a user attempts to interactively close the front panel by selecting File>>Close.

Which Event case allows this functionality?

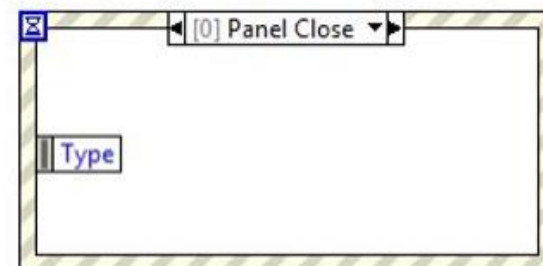
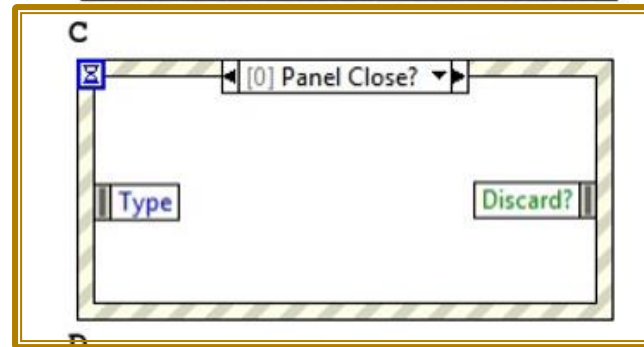
**A**



**B**



**C**



The Question Mark allows the program to make a change before the action happens. In this case, the action would be closing the program.