Tomoyuki Mano

Mini Course - LabVIEW – Day3

Events, queues, and more fun stuffs!

2022/2/15

So far we've learned…

- **Data types**
- **While and For structures**
  - Indexing
- **Case structures**
- **Arrays**
- **Clusters**
- **Sub VIs**

Today's content

- **Events and Event structures**
- **Local variables**
- **File IO**
- **Queues and Notifiers**
- **Consumer-Producer Pattern**
- **+4 exercises!**

Today we'll learn key concepts that are characteristic to LabVIEW programming, and these techniques are necessary to create useful real-world applications.

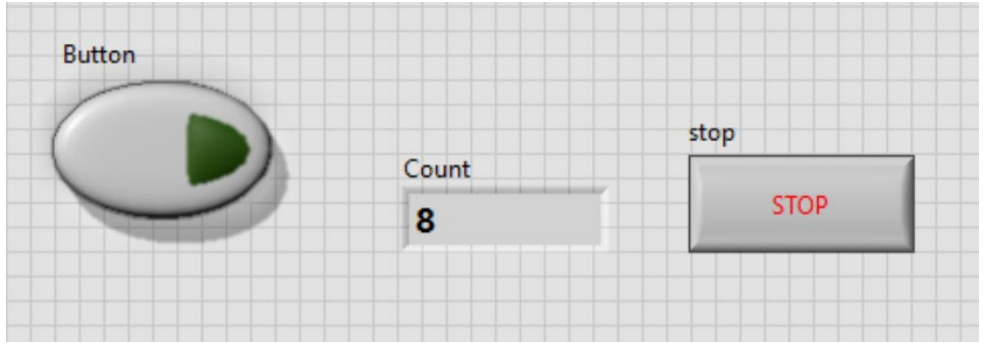
Review Exercise!

Counter App

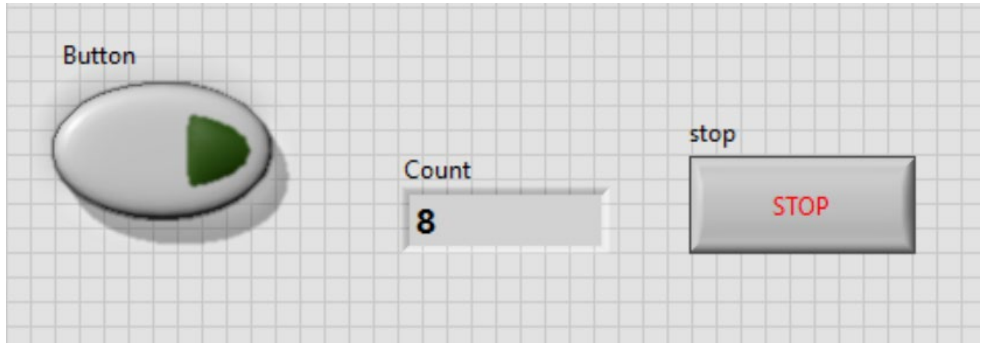
Make a simple counter program using LabVIEW. The program waits for user input (button click), and when the click happens the program increments the count by 1. The program will stop when a stop button is hit.

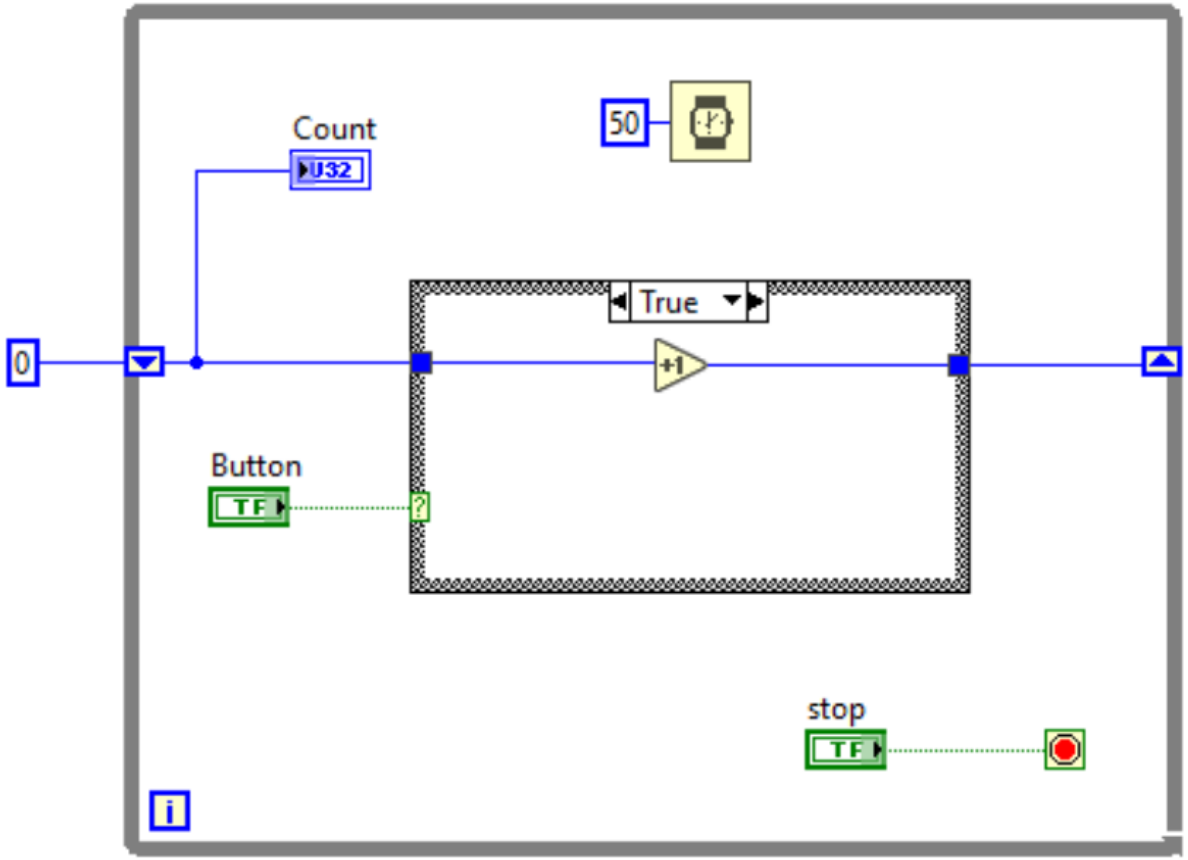
Use these techniques you learned so far:

- While loop
- Case structure
- Shift register



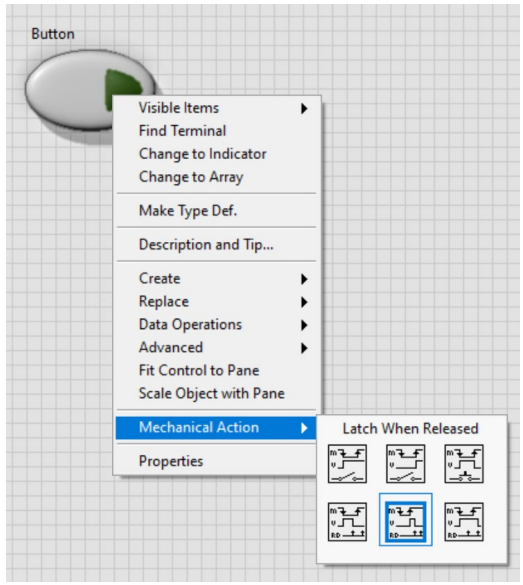
Review Exercise Answer

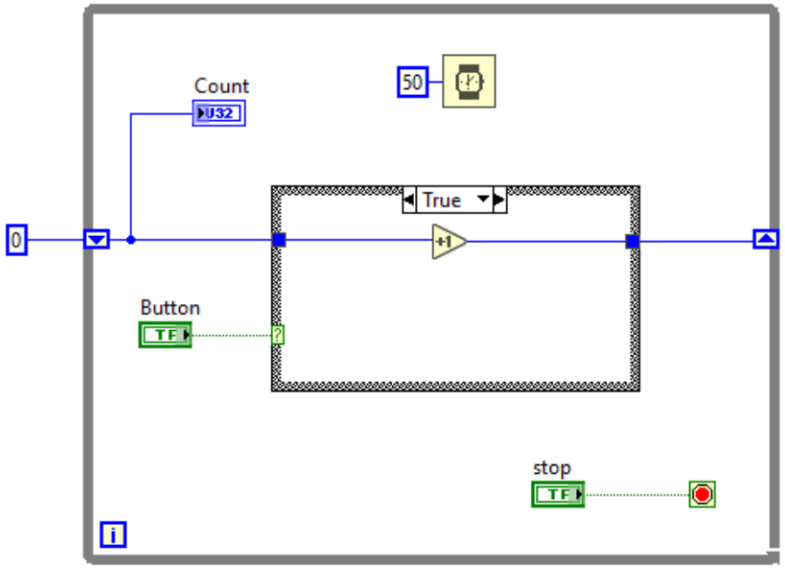




See “counter_polling.vi”

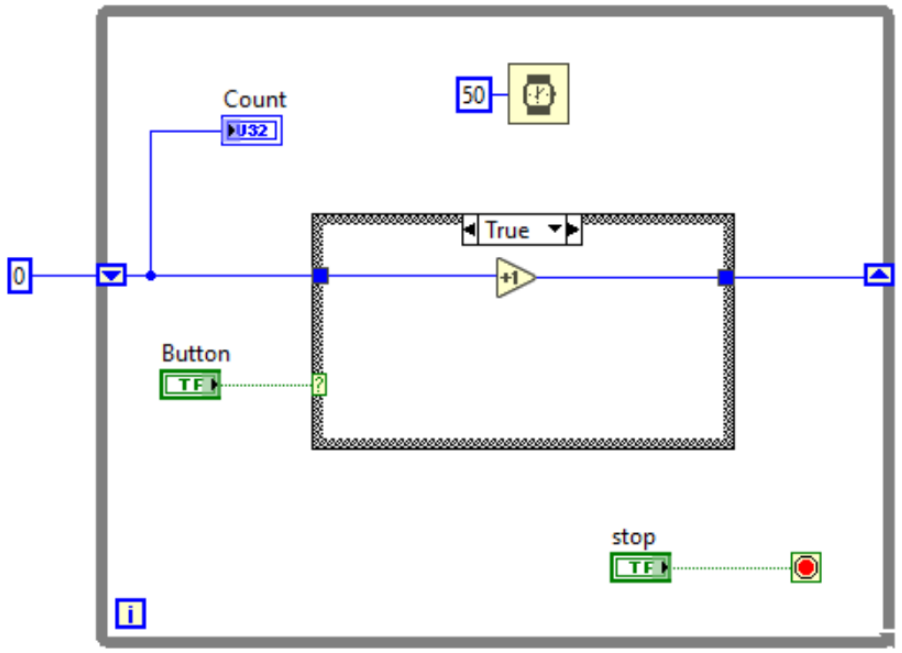
Tip: Latch and Switch mode for Boolean controllers





- You can change the mode of boolean controllers by right clicking and choose “Mechanical Action”.
- Switch mode: After clicking, the value toggles and keeps this value until next click.
- Latch mode: After clicking, the value toggles ***until the value is read in the block diagram*** and then comes back to the original value immediately.

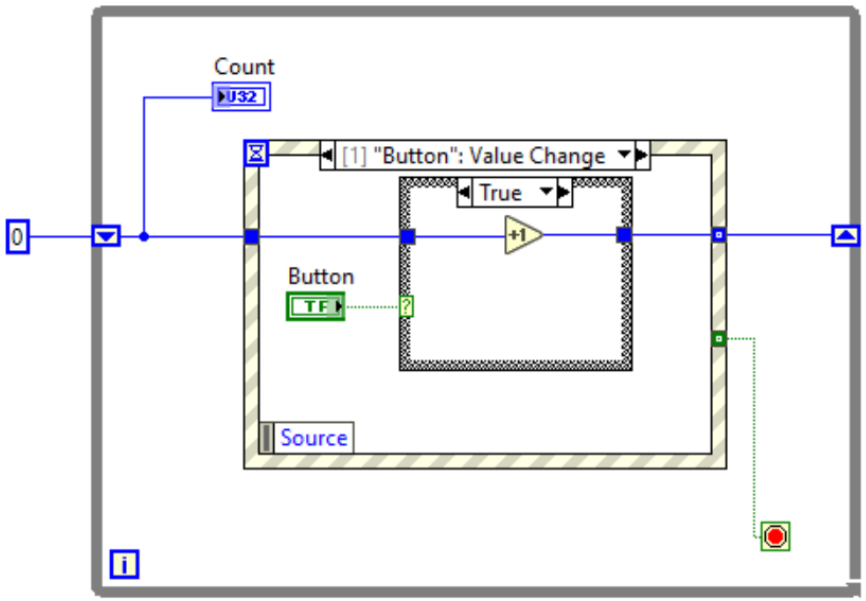
Events and Event Structure (1)



- Take a look at the counter app we just made
- We see that using while loop we check the value of “Button” every 50 msec.
- This is called ***polling***
- Polling is simple, but can be wasteful of CPU resources
- Also, monitoring many variables with this design becomes difficult

- We use ***events*** and ***event structure*** to solve this issue

Events and Event Structure (2)



- Using event structure, one can add cases which should be executed when a specified event happens (e.g. controller value change or keyboard inputs).
- Events are registered in a special queue in LabVIEW, which can very efficiently monitor the changes of the resources.
- It is almost always placed inside a While loop

- On the left, the counter app was rewritten using event structure (See “counter_event.vi”)

Events and Event Structure (3)

Why use events (event-driven programming)?

- User inputs (and certain kind of data stream) are unpredictable, so polling is not efficient.
- Polling frequency (e.g. 50msec) is arbitrary.
- Event structure is an efficient and scalable approach to monitor many actions within LabVIEW.

When you make an interactive program, please do use event structures (and producer-consumer pattern; later in this slide)!

Events and Event Structure (4)

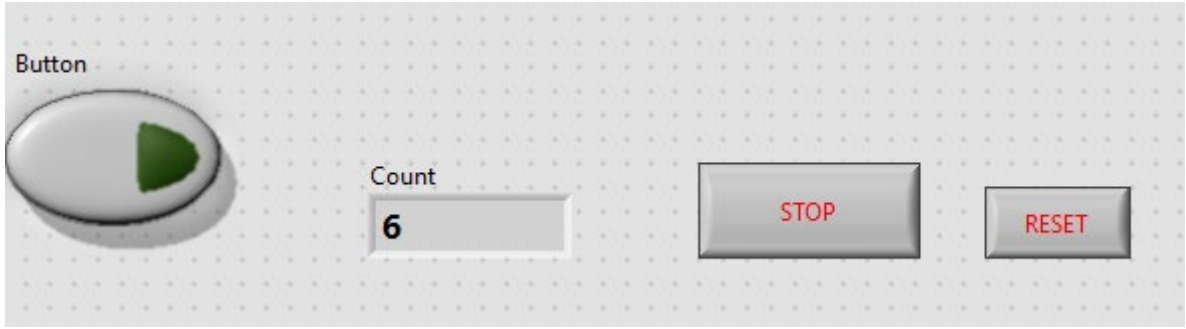
Event structure recommendations and caveats

- Normally, events only respond to user inputs. If you want to handle software-driven events, use dynamic events.
- Event cases are executed sequentially. Event handler should be made as compact as possible. If a complicated logic is needed, use producer-consumer pattern (see later slides).
- For more information, see https://zone.ni.com/reference/en-XX/help/371361R-01/lvhowto/caveatsrecmndtnsevnts/

Exercise!

Counter App v2

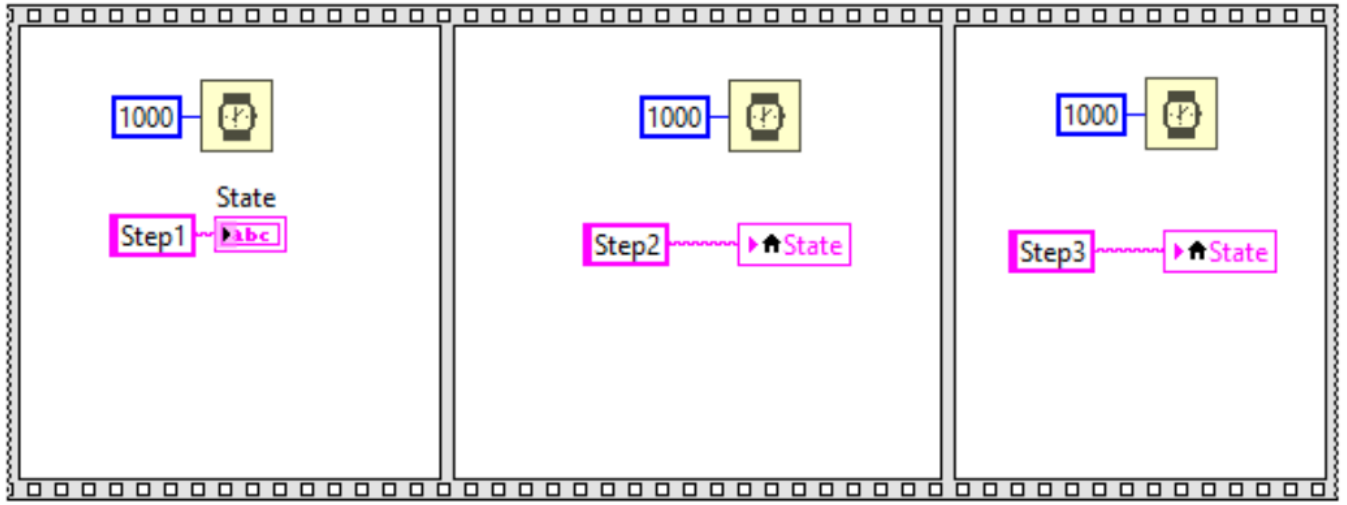
Rewrite your counter app using event structure. Also, add a “reset” button, which you use to reset the count back to 0.



Answer: See “counter_event_v2.vi”

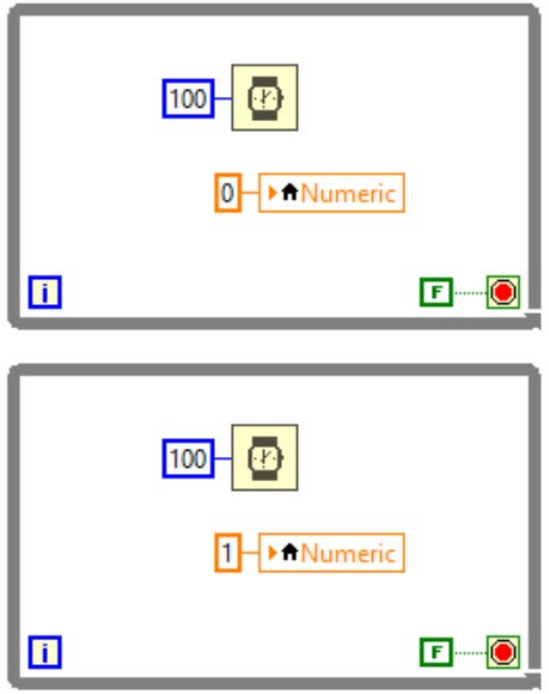
Local Variables

- A reference to a block diagram terminal (i.e. control or indicators)
- Local variables are readable and writable; you can write to it even if it is a control or read from it even if it is an indicator.
- You create a local variable by clicking the block diagram terminal and select “**Create” >> “Local Variable”**.
- Used to initialize the values, display the state, etc. See “local_variables.vi”



Carefully Use Local Variables

- Local variables are useful, and sometimes it is the only way to implement certain logic.
- However, local variables are often the cause of unexpected bugs (e.g. simultaneous read and writes; see demo). Also, code readability is low if too many local variables exit.
- **Try not to use local variables unless it is absolutely necessary.** Use wires or shift registers if possible.

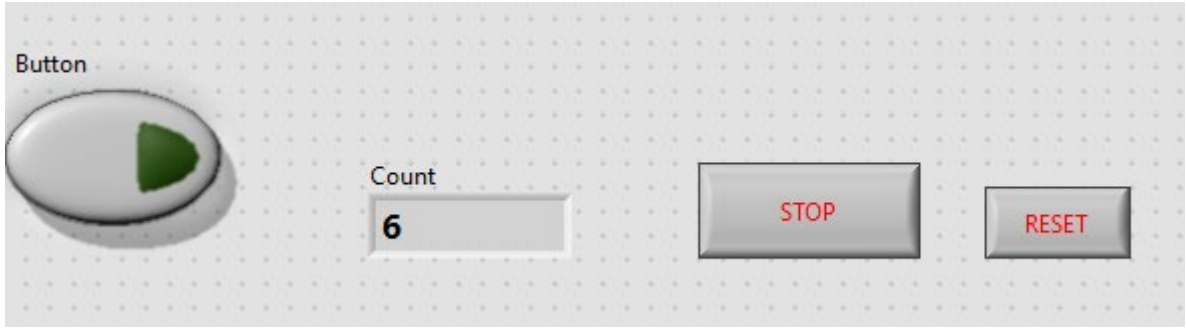


- Local variables can only be accessed within the same VI. If you need to share variables across VIs, use ***global variables***.

Exercise!

Counter App v3

In v2, the button flickering was not controlled. Now, using local variables, modify the program so that clicking causes the button to become green (True) for 100msec, and then reset back to False.



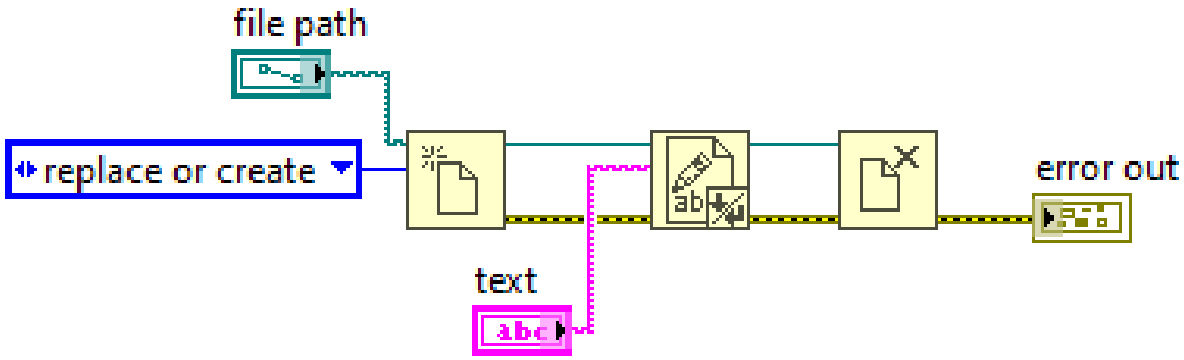
Answer: See “counter_event_v3.vi”

File IO

- LabVIEW has several “built-in” file-types which are easy to interact with
- Binary files of any format can also be handled, with varying difficulty depending on the format!
- We will focus on the Text File format today

Writing to a file

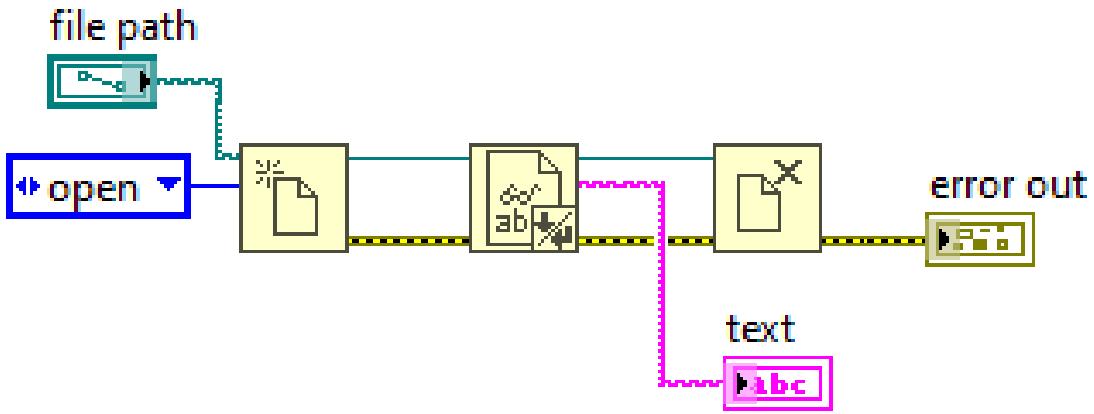
- The “Create/Open/Replace File” node creates a reference to a file
- Use “Write to Text File” to write strings into a file
- You need to close this when you’ve finished using it
- You can safely(ish) branch this wire, but only close it once!
- It’s probably better not to branch the wire unless you really need to



See “write_text.vi”

Reading from a file

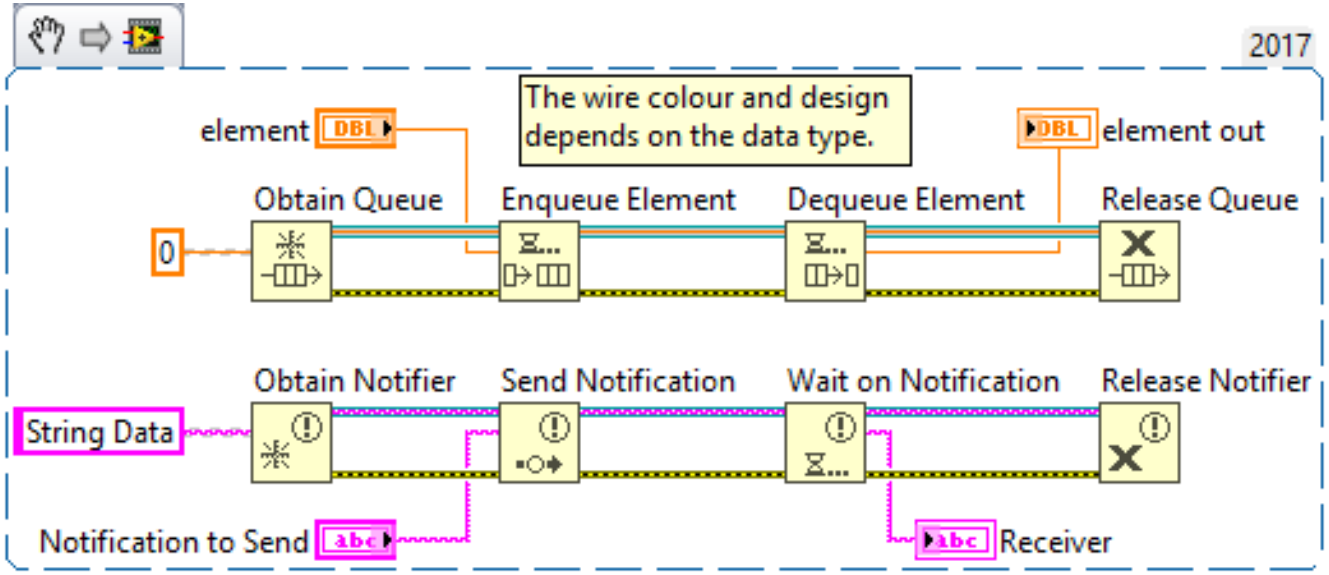
- The “Create/Open/Replace File” node creates a reference to a file
- Use “Read from Text File” node to read text.
- You need to close this when you’ve finished using it



See “read_text.vi”

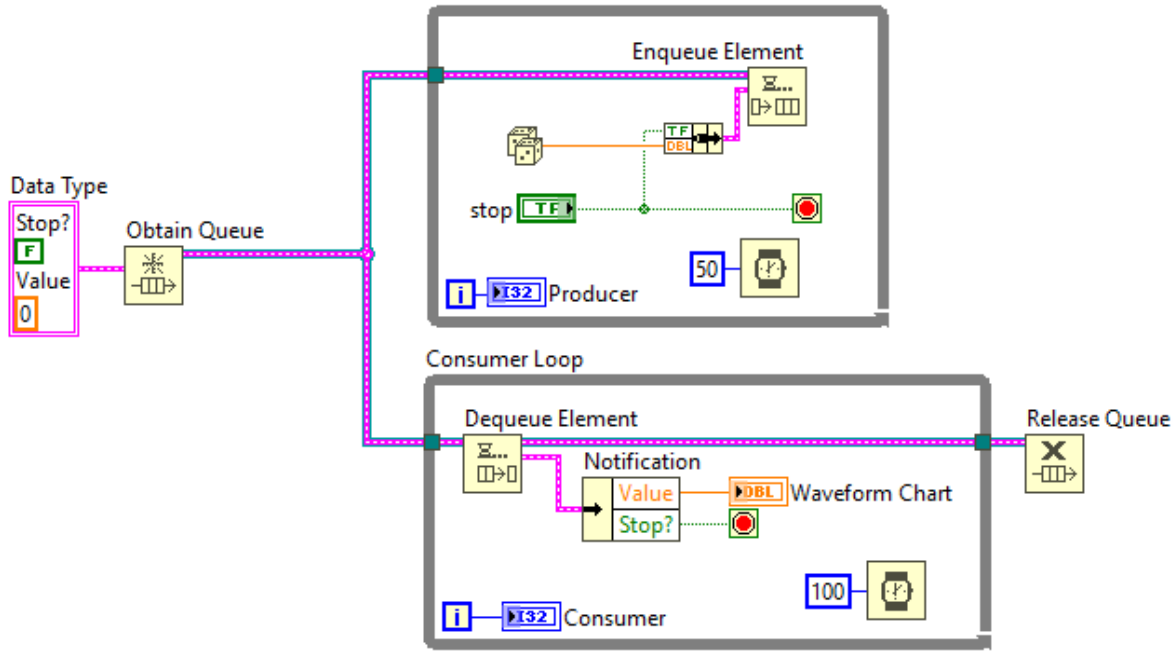
Queues and Notifiers

- Queues and Notifiers are two of the main inter-loop communication methods in LabVIEW - they are found in the Synchronization palette
- A Notifier sends the **latest value**, and can be shared between multiple receivers – you can use this for 1:1 or 1:N or N:M communication
- A Queue holds a buffered collection of values, but reading an element removes it from the queue – use this for 1:1 or N:1 communication



Producer-Consumer Pattern (1)

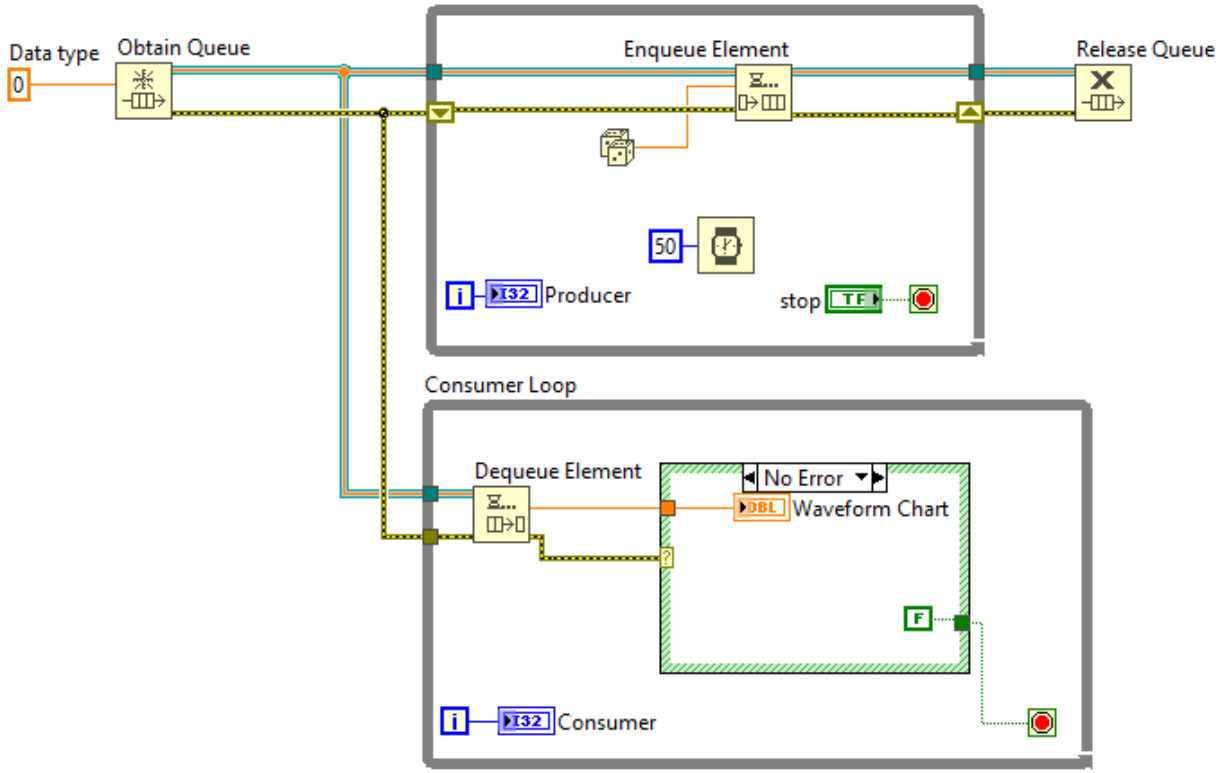
- Very common parallel processing design pattern.
- Producer loop produces data or commands and put it in a queue (Enqueue).
- Consumer loop obtains data out of queue (Dequeue) and process the data or commands.
- Queues maintains a first in/first out (FIFO) order.



See “queues1.vi”

Producer-Consumer Pattern (2)

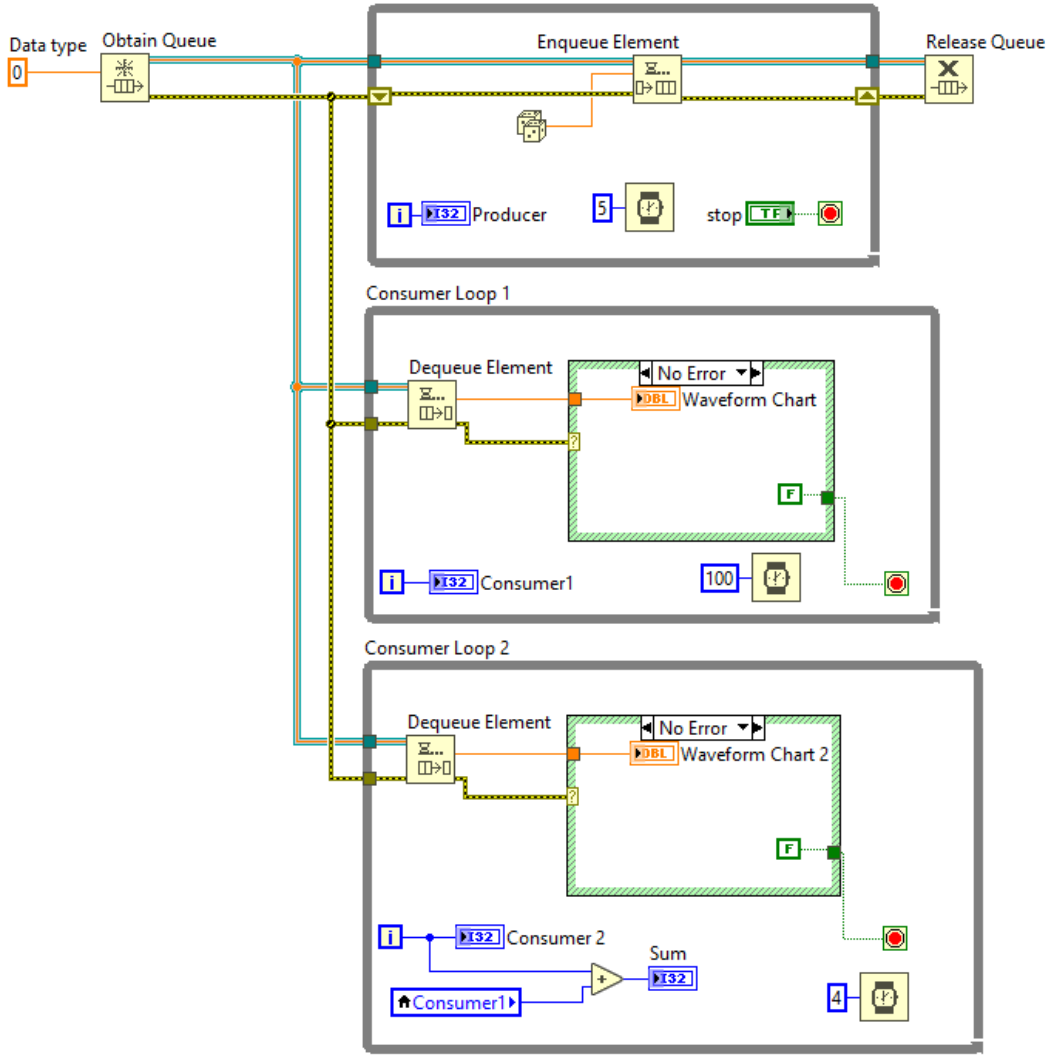
- In the first version, we passed Boolean value over queue.
- This design is a bit tedious, so let's replace it with a more convenient design.
- Notice that termination of the producer loop will cause the consumer loop to stop. Otherwise, you will be in a deadlock state!



See "queues2.vi"

Producer-Consumer Pattern (3)

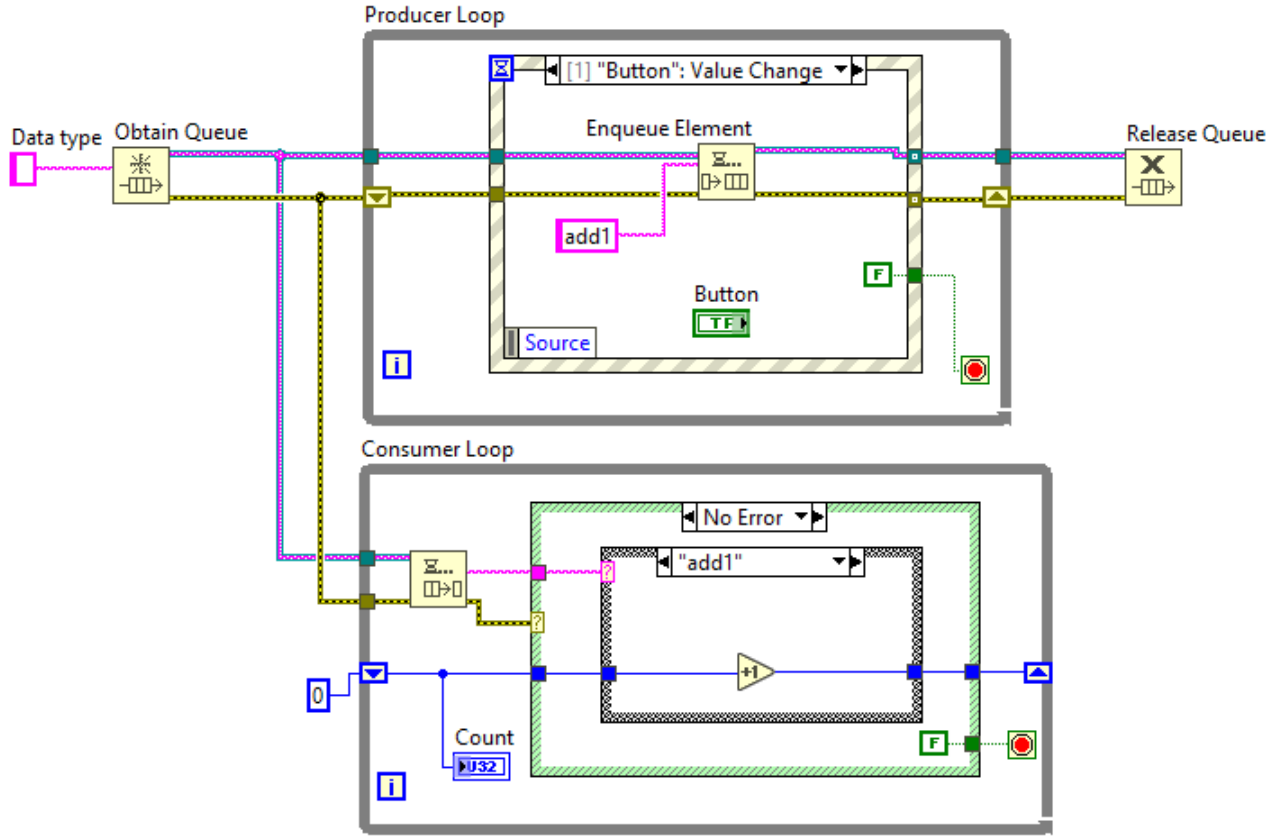
- You can have multiple consumers to process data in parallel.
- Make sure that you terminate all consumer loops when the producer loop is stopped.



See “queues3.vi”

Producer-Consumer Pattern (4)

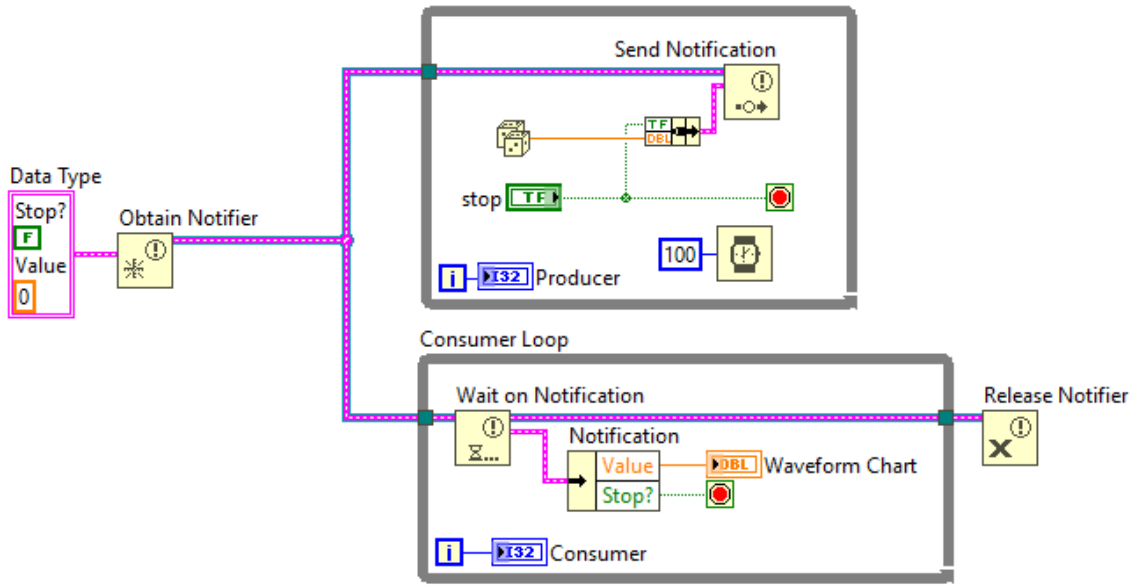
- Producer-consumer pattern is often used with Event structure.
- Producer loop handles user interactions (e.g. push commands into queue).
- Consumer pattern dequeues command and execute the process.



See “queues4.vi”

Notifiers

- Notifiers works essentially in the same way as queues.
- Notifier has no buffers and can only store the latest value.
- This is useful when you are only interested in the latest data point, such as displaying data stream in real time to the user.

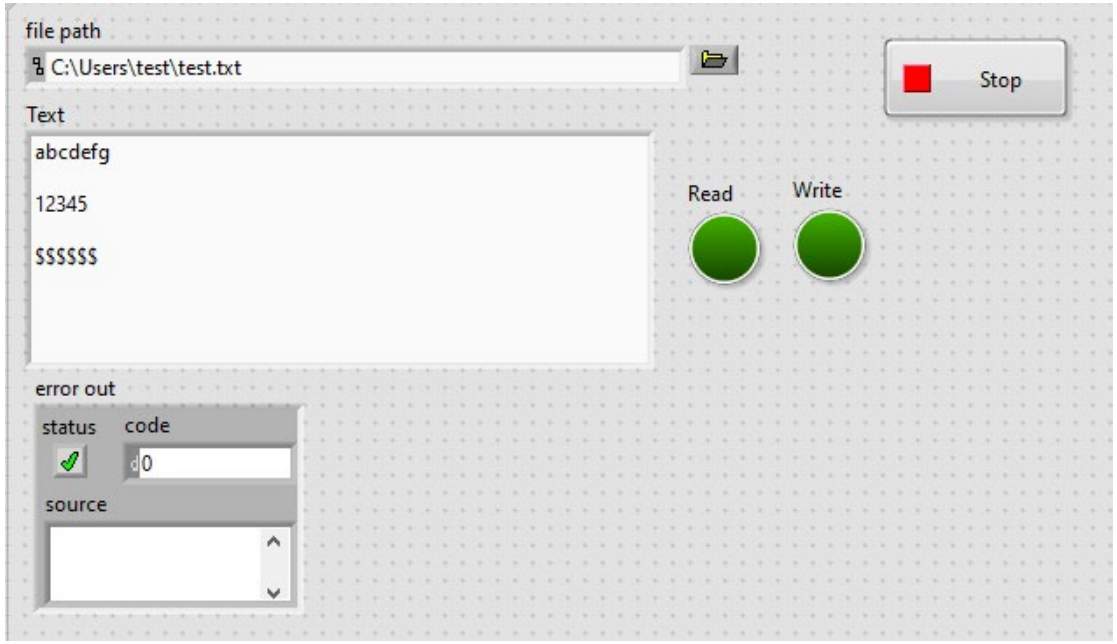


See “notifiers.vi”

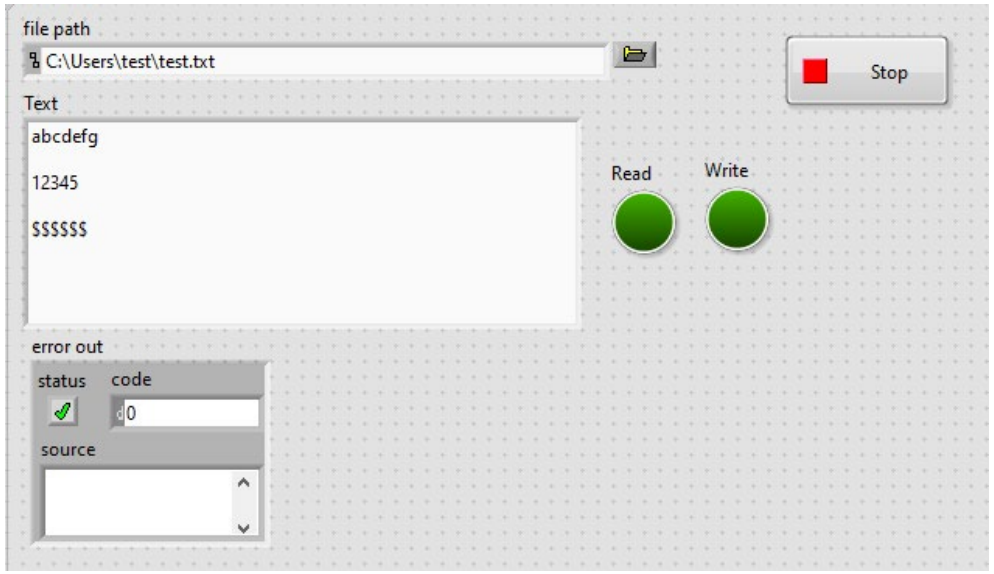
Exercise!

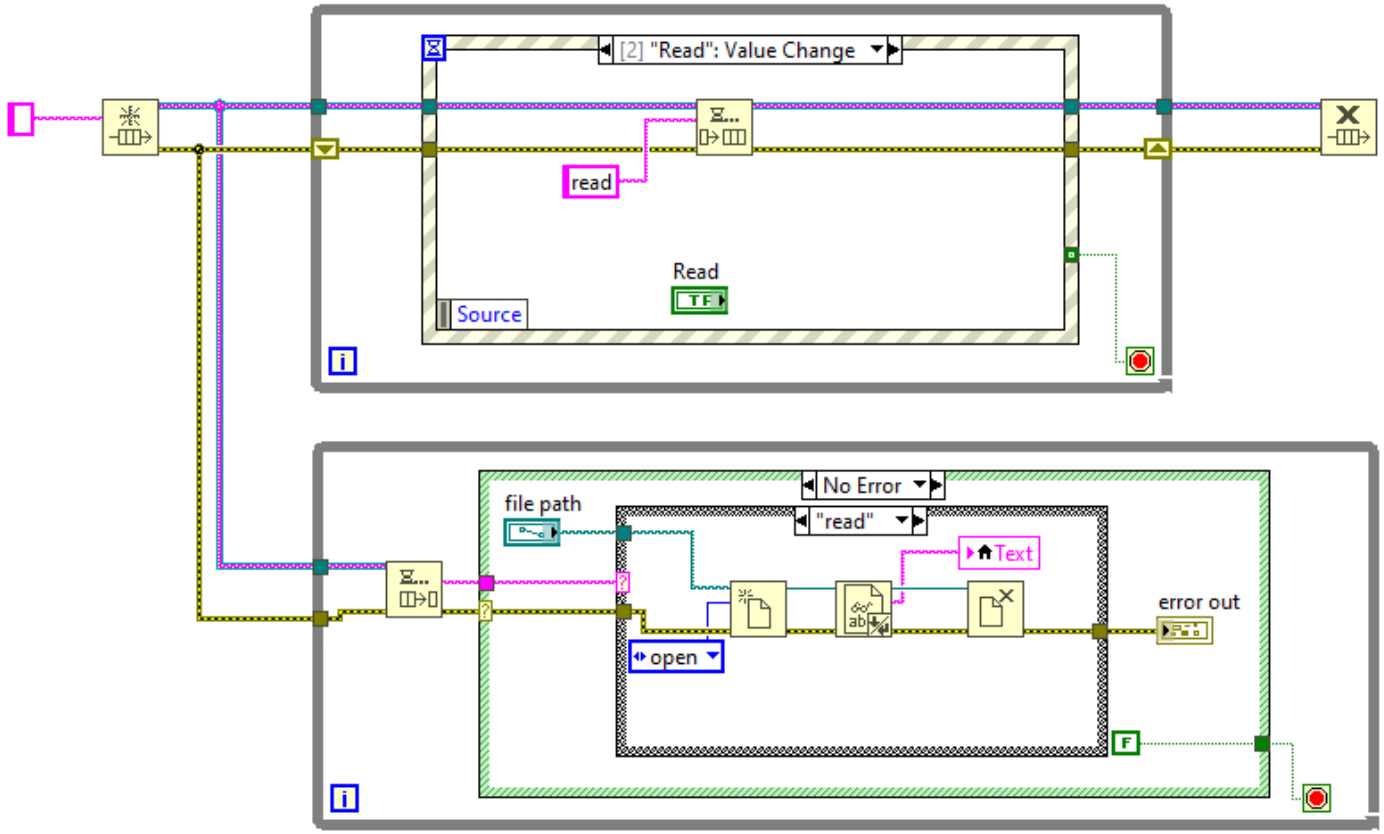
Text Editor App

When you hit “Read file” button, it reads the text file specified by the user. When you hit “Write file” button, the text in the text box will be written to the file. Implement this program using producer-consumer pattern and Event structure. You may need to use local variables.



Exercise Answer





See “text_editor.vi”

Thank you!