Create a Print Queue Console

Goal

Use the Producer/Consumer (Event) template to create a print queue console that submits print jobs to a printer and monitors the print queue.

Scenario

You must create a print console VI that controls print job submissions and monitors the processing status of the print jobs. Each time the user clicks on the Queue Event button, a print job will be sent to the printer. You will monitor the number of print jobs in the queue using a vertical bar. LEDs will indicate the print processing status.

The print process for each job includes two steps-an initialization stage with a fixed duration of 300 ms and a print stage with a varying duration based on the size of the job.

Design

The controls and type definition enums necessary for this project are already created. Your task is to use the Producer/Consumer design pattern template that ships with LabVIEW to create the print console VI. Table 1 lists the controls used in the front panel of the print console VI.

Table 1. Front Panel Controls and Descriptions

Control	Control Description
Queue Event Button	Each time the user clicks on this button, a print job is sent to the printer.
Queued Print Jobs bar	This bar displays the number of print jobs that are currently in the print queue. Each time a print job is sent to the printer, a new job is added to the queue. When the printer starts a new job, the print job is removed from the queue.
Waiting LED	This LED is turned on only when there are no print jobs in the queue.
Initializing LED	This LED is turned on for the duration of the setup stage for each print job.
Printing LED	This LED is turned on for the duration of the print stage for each print job.
Stop Button	This button stops the VI.

Table 2 lists the Event cases you will implement in the producer loop.

Table 2. Producer Loop Event Cases

Event Case	Case Description
Queue Event: Value Change	In this case, you use the Enqueue Element function to queue a string constant with the value of `print job`.
Stop: Value Change	In this case, you stop the While Loop.
Timeout	In this case, you use the Get Queue Status function to get the number of elements in the queue. You pass the value to the Queued Print Jobs bar.

Table 3 lists the states you will implement in the consumer loop state machine.

Table 3. Consumer Loop States

State	State Description
Wait	You perform the following tasks in this state. • Use the Dequeue Element function to wait for a queued print job. • Set the Dequeue Element function to timeout for 100 ms. • Set the next state to be the Wait state, if the Dequeue Element function times out. Set the next state to be Initialize Setup, if the Dequeue Element function does not time out. • Set the Waiting LED to True.
Initialize Setup	You perform the following tasks in this state. • Set the Waiting LED to False. • Set the Initializing LED to True. • Transition to the Setup state.
Setup	You perform the following tasks in this state. • Simulate print setup with a 300 ms wait. • Set the Initializing LED to False. • Transition to the Initialize Print state

Table 3. Consumer Loop States (Continued)

State	State Description
Initialize Print	You perform the following tasks in this state. • Set the Printing LED to True. • Transition to the Print state.
Print	You perform the following tasks in this state. • Simulate the print job using a variable wait time. • Set the Printing LED to False. • Transition to the Wait state.

Implementation

Note To reinforce the development skills you have learned so far, you are given limited instructions to create the application.

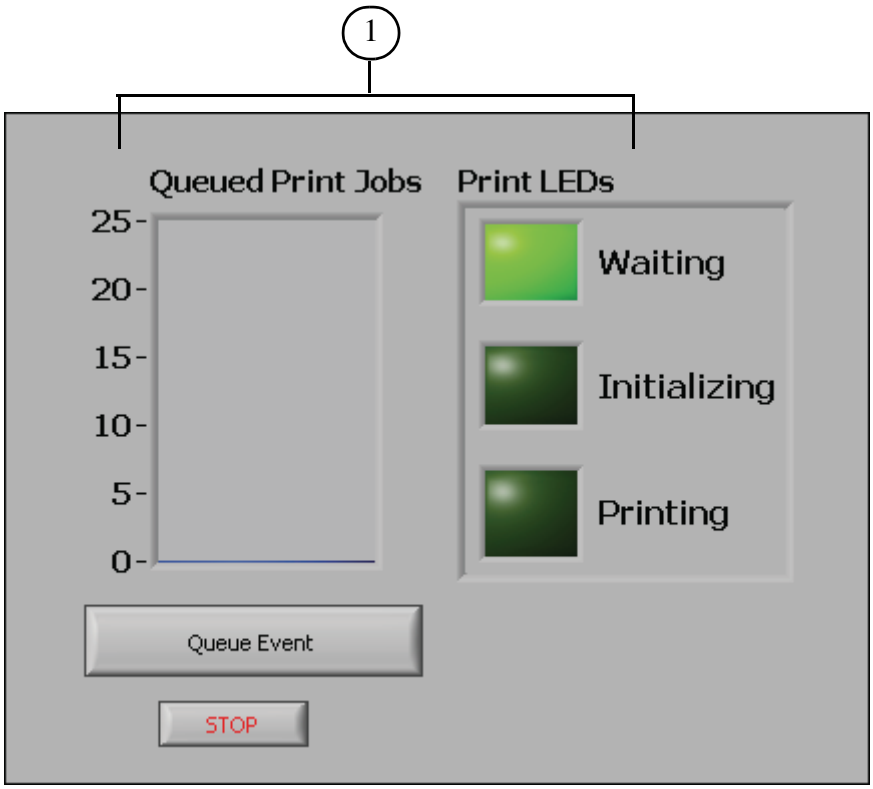
The files that you need to complete this exercise are here: `<NI eLearning>\LV Core 2\Event_Design Patterns\Exercise`.

Create a New VI From a Template

1. Open `Print Console.lvproj` in the `<Exercise>` directory.
2. Create a new VI using the Producer/Consumer Design Pattern (Events) design pattern.
3. Select **File»VI Properties»Documentation**. In the VI description section, replace the template documentation with a concise description of your application.
4. Create a meaningful icon for the VI.
5. Save the VI as `Print Console.vi` in the `<Exercise>` directory. Confirm that the new VI is added to the `Print Console.lvproj` project.
6. Save the `Print Console.lvproj` project.

Create the Front Panel

Create the front panel as shown in Figure 1 using the `Queued Print Jobs.ctl` and the `Print LEDs.ctl` available in the `Print Console.lvproj` project.



1 Already created in Print Console.lvproj

Figure 1. Print Console Front Panel

Producer Loop

Complete the block diagram by first modifying the producer loop and then modifying the consumer loop.

The producer loop responds to front panel events. Each time the user clicks the Queue Event button, a print job is added to the queue. The producer loop is also responsible for updating the Queued Print Jobs bar with the number of elements in the queue.

Figure 2 and Figure 3 are examples of the completed producer loop.

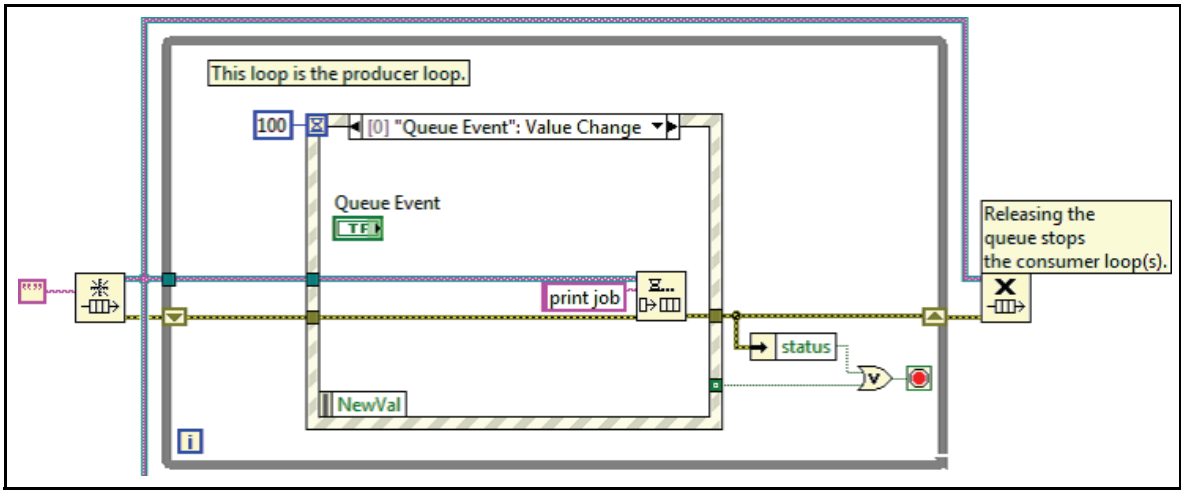


Figure 2. Producer Loop—Queue Event Case

1. In the Queue Event case, replace the `element` string with `print job` in the Enqueue Element function.

2. Add a Timeout case to the Event structure. The Timeout case updates the Queued Print Jobs bar by reading the number of elements in the queue.

   ❑ Select **<Application>** in the Event Sources section and select **Timeout** in the Events section.

Note If the Event Data Node label is invalid (black text), right-click the label and select a valid one.

Note Because the Timeout case is an application event, the Event Selector label at the top of the Timeout case differs from other events associated objects. The label name only specifies the event and does not include the associated object.

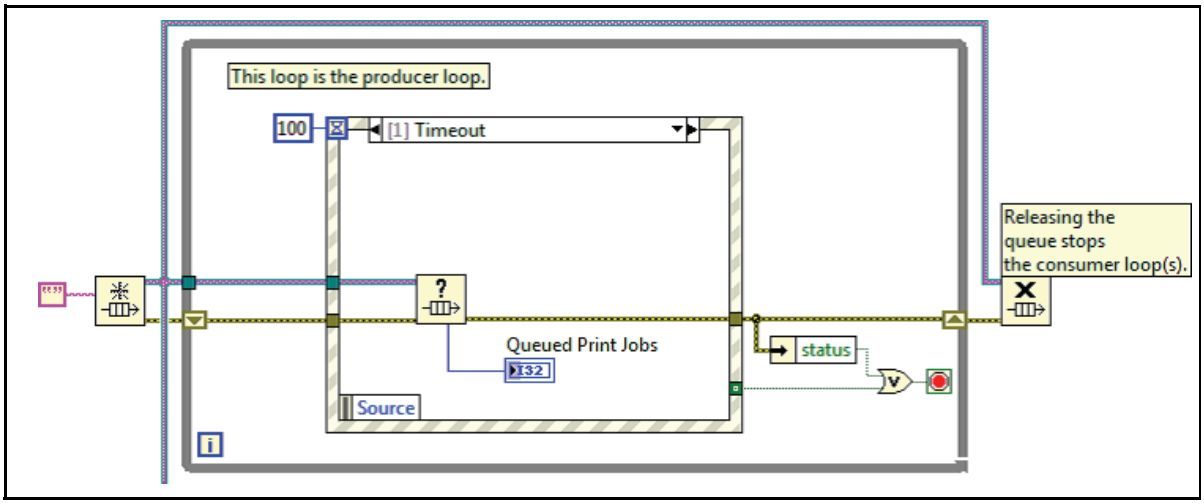


Figure 3. Producer Loop—Timeout Event Case

- ❑ Add a Get Queue Status function to read the number of elements in the queue.
- ❑ Set the Event structure to timeout every 100 ms by wiring a constant value of `100` to the Timeout terminal.

Note You do not need to make any changes to the Stop case as it already exists in the template.

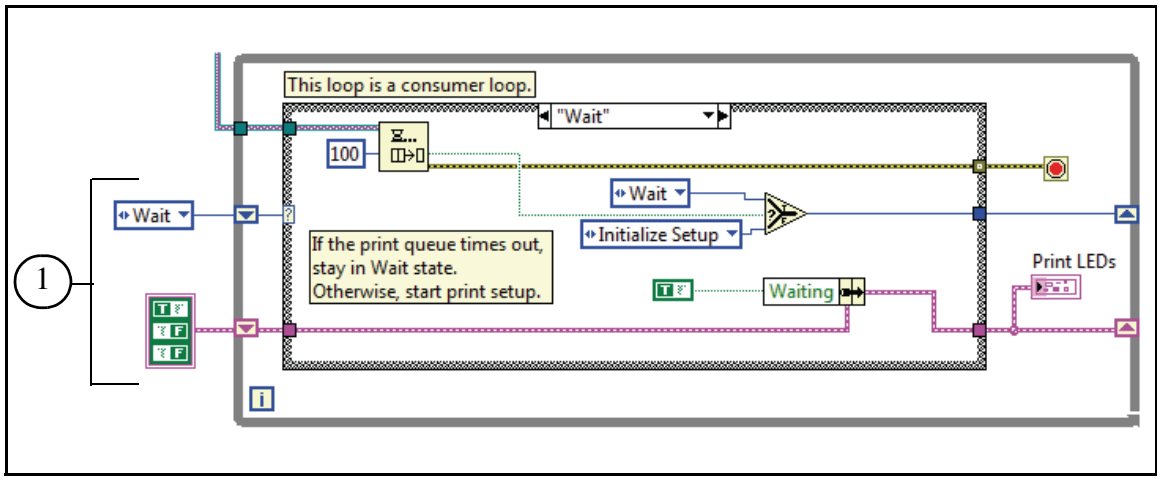
Consumer Loop

The consumer loop simulates the print process and updates the LEDs. Implement the print process using a state machine. The Wait state receives print jobs. When a print job is received, the state machine initiates a series of steps to simulate the print process. The Print LEDs update after each state. Therefore, the Initializing LED turns on after the Initialize Setup state and turns off after the Setup state. Similarly, the Printing LED turns on after the Initialize Print state and turns off after the Print state.

Figures 4 through 8 are examples of the completed consumer loop.

1. Create a state machine in the consumer loop by starting with the Wait state. The Wait state receives print jobs using the Dequeue Element function. If the Dequeue Element function receives a queue element within the specified 100 ms timeout the state machine transitions to the

Initialize Setup state. If the Dequeue Element function times out, the state machine stays in the Wait state.



1 Already created in Print Console.lvproj

Figure 4. Consumer Loop—Wait Case

- ❑ The print processing states type definition enum is created for you. Select and drag the `Print Processing States.ctl` from the Project Explorer window onto the block diagram.
- ❑ Wire the `Print Processing States.ctl` to the Case selector to associate the Case structure with the enum values.

- ❑ Move the Dequeue Element function into the Case structure.
- ❑ Use the timed out? output of the Dequeue Element function to determine if the next state is the Wait state or the Initialize Setup state.

- ❑ Add a Bundle by Name function to set the Boolean state of the Waiting LED.
- ❑ Drag the `Print LEDs.ctl` from the Project Explorer window onto the block diagram to create a Print LEDs cluster constant on the block diagram.

2. Refer to Figure 5 to create the Initialize Setup case.

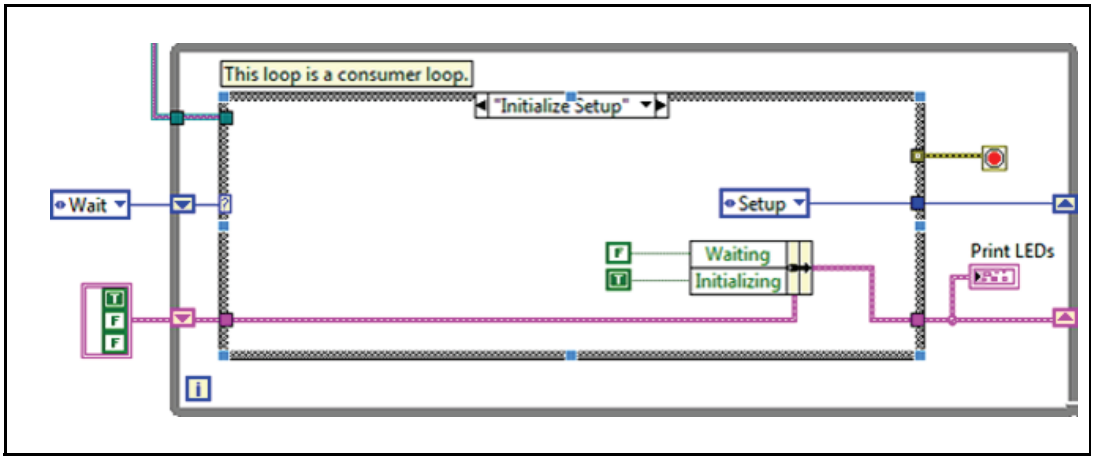


Figure 5. Consumer Loop—Initialize Setup Case

❑ To create cases associated with each of the enum values, right-click the edge of the Case structure and select **Add Case for Every Value**.

Tip Alternatively, you can also create new cases by duplicating existing cases. To duplicate a case, right-click the edge of the Case structure and select Duplicate Case. Be careful when duplicating cases that include front panel terminals as this will result in duplicated front panel objects.

❑ Right-click on the error tunnel and select **Use Default if Unwired**.

3. Refer to Figure 6 to create the Setup state.

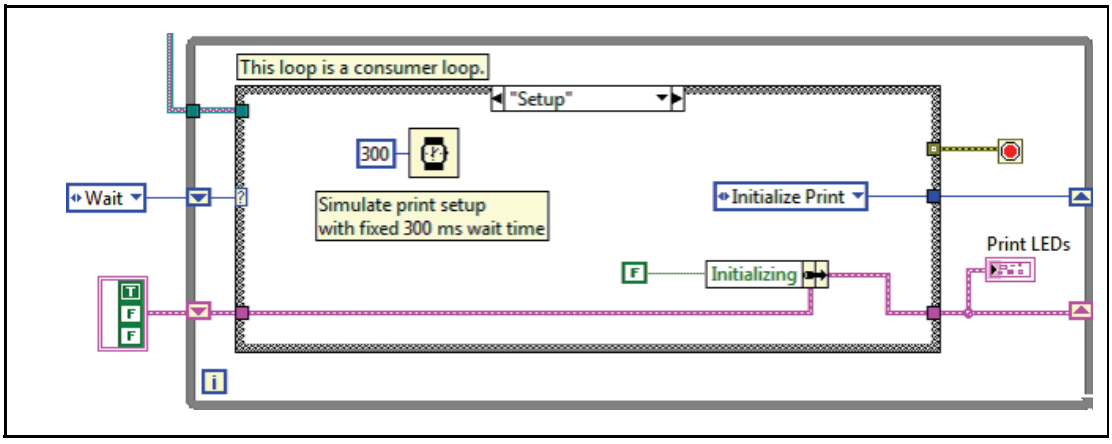


Figure 6. Consumer Loop—Setup Case

❑ Simulate the initialization duration with a 300 ms wait.

4. Refer to Figure 7 to create the Initialize Print state.

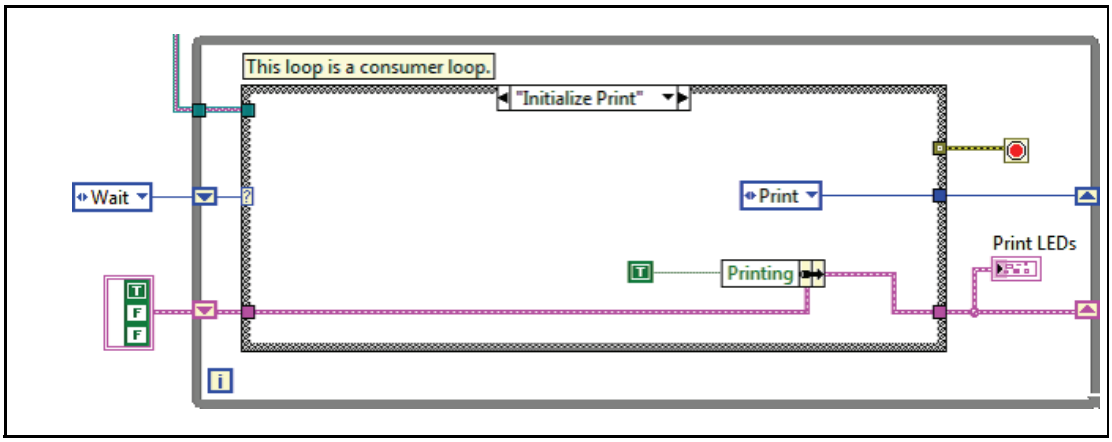


Figure 7. Consumer Loop—Initialize Print Case

5. Refer to Figure 8 to create the Print state.

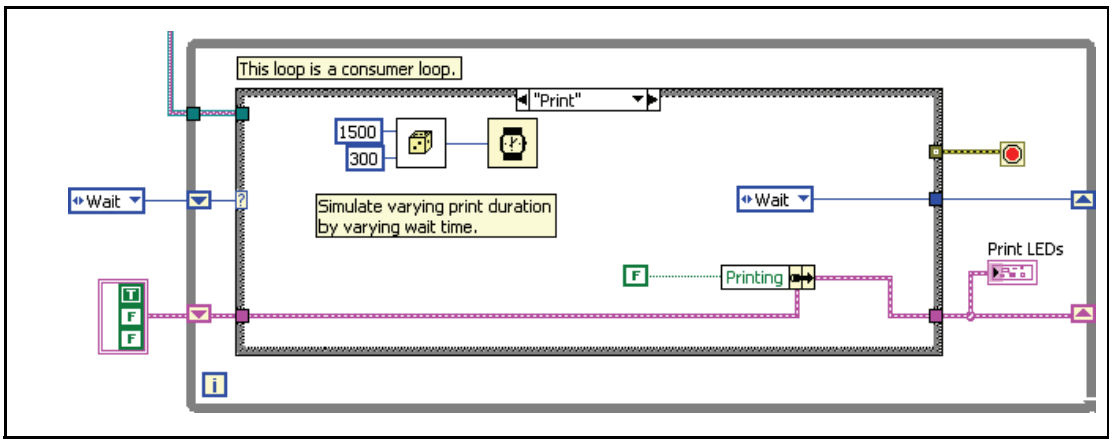


Figure 8. Consumer Loop—Print Case

- ❑ To simulate a varying print duration, generate a random wait time between 300 ms and 1500 ms. Use the Random Number in Range VI found in your `Print Console.lvproj` to set a random wait duration for a print job.

6. Save the VI and save the `Print Console.lvproj` project.

Test

1. Run the VI to confirm that it behaves correctly. Each time you click the Queue Event button, a new print job is submitted to the print queue. If

you submit several print jobs in succession, you will see a backlog of print jobs in the Queued Print Jobs bar.

- ❑ Notice that after you stop queueing the print jobs, the print process continues to execute.

- ❑ Notice that the consumer loop stops only when there is an error. This occurs when the producer loop stops and the Release Queue function destroys the queue reference. Without a valid queue reference, the Dequeue Element function in the consumer loop returns error code **1122** with an explanation of **Refnum became invalid while node waited for it** or an error code **1** with an explanation of **An input parameter is invalid** (if there are elements left in the queue when you press stop).

  Stopping a consumer loop on error is relatively easy to implement but is not recommended when creating professional applications that are scalable and maintainable. A better approach is to shutdown gracefully and report all errors to the user.

2. Close the VIs and project. Save any changes.

End of Exercise

Notes