Exercise 2-4: Channeled Message Handler

Goal

- Explore an example using the CMH design pattern.

Hardware Setup

(Hardware) In the exercises where we work with Analog Input/Output channels, we use PCI-6221/USB-6212 multifunction I/O device paired with the BNC-2120 shielded connector block. Analog Input 2 should be connected to the Sine/Triangle BNC connector. Analog Input 3 should be connected to the TTL Square Wave BNC connector. The Sine/Triangle waveform switch should be set to Sine.

Scenario

You have a VI that uses the CMH design pattern to process messages, like the Producer/Consumer (Events) LabVIEW project in the previous exercise.

Guided Instructions

1. Open `C:\Exercises\ LabVIEW Core 2\Channeled Message Handler\Channeled Message Handler.lvproj`.
2. From the **Project Explorer** window, open the Main VI.
3. Explore the Message Queue VIs included in this project.
   - In the **Project Explorer** window, expand the **Support VIs** folder, notice the Message Queue VIs under the **Message Queue LabVIEW** library.
   - These three VIs are used in the Main VI block diagram.
     - Enqueue Message
     - Dequeue Message
     - Abort Message Queue
   - Switch to the block diagram of the Main VI.
   - Open the **Context Help** window.
   - Notice that the Enqueue Message and Dequeue Message VIs have **message** (string data type) and **data** (variant data type) terminals.
   - If you look at the block diagrams of the Message Queue VIs, you will see that these VIs use the channel wire writer and reader endpoints (in this case — messenger channel).
4. Examine the block diagram similarities of the CMH design pattern compared to the Producer/Consumer (Events) design pattern from the previous exercise.
   - The Event Handling Loop in the CMH design pattern is similar to the Producer Loop in the Producer/Consumer (Events) design pattern.
   - The Message Handling Loop in the CMH design pattern is similar to the Consumer Loop in the Producer/Consumer (Events) design pattern.
5. Explore how the CMH behaves if an error occurs in the Event Handling Loop.
   - Imagine an error occurs in the Event Handling Loop. The error will be passed into the Error Handler - Event Handling Loop VI.
   - Double-click the **Error Handler - Event Handling Loop VI** to open its block diagram.
   - Examine the block diagram of this VI to explore how it handles the error. This VI passes the error into the **Check Loop Error VI**.
   - Notice that the **Exit on Error?** input of the **Check Loop Error VI** isn't wired, so we are not passing any value to it.

- Double-click the **Check Loop Error VI** to open its block diagram.
- Notice that the **Exit on Error** control will read default False value when this VI executes. This means that the Check Loop Error VI will enqueue an Error message along with the error message data on the message channel.

Note: If the error is listed in the **Ignore Errors** array, the Check Loop Error VI will ignore the error and do nothing.

- Go back to the Main VI block diagram. Because the error handling VIs enqueued an Error message, the Dequeue **Message VI** in the Message Handling Loop now dequeues the Error message and the error message data. This causes the inner Case structure of the Message Handling Loop to execute the Error case.
- The Error case displays an error dialog using the Simple Error Handler VI.

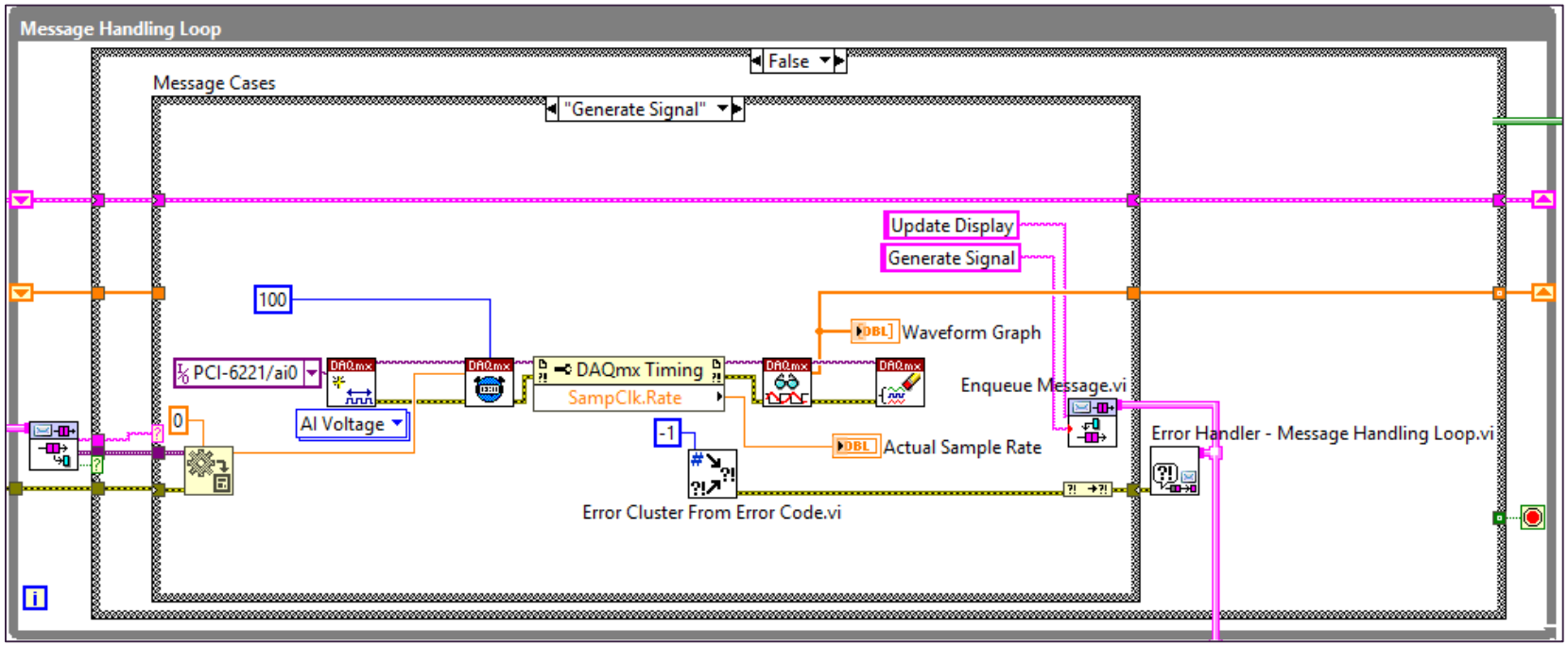
6. Explore how the CMH handles shutting down both loops if an error occurs in the Message Handling Loop.
   - Imagine an error occurs in the Message Handling Loop. The error will be passed into the Error Handler - Message Handling Loop VI.
   - Examine the block diagram of this VI to explore how it handles the error. This VI passes the error into the **Check Loop Error VI**.
   - Notice that now a **true** boolean constant is wired to the **Exit on Error** terminal of the Check Loop Error VI.
   - Open the block diagram of the **Check Loop Error VI**.
   - Based on the value of the **Exit on Error** control (which is TRUE), the corresponding Case structure executes the **Abort Message Queue VI** instead of queuing an Error message.
   - Now open and examine the block diagram of the Abort Message Queue VI. Abort Message Queue VI will enqueue an Exit message along with the error message data on the message channel. It also wires the TRUE boolean constant to the **abort** terminal of the Messenger channel writer endpoint.
   - Go back to the Main VI block diagram. Because the Error Handler - Message Handling Loop VI enqueued an Exit message, the error message data, and the **abort** terminal's value, the Dequeue Message VI dequeues the same data: Exit message, the error message data, and the **true** value from the **channel aborted** terminal.
   - **Channel aborted** output of the Dequeue Message VI passes the true value to the conditional terminal of the outer Case structure.
   - The True case performs the following actions.
     - Displays an error dialog using the Simple Error Handler VI.
     - Writes Exit Application message to the **Current Consumer State** indicator.
     - Uses the Write VI to tell the Event structure in the Event Handling Loop to execute the <**channel value**>**: User Event** event case next.
     - Sends a True value to the Condition terminal, which stops the Message Handling Loop.

Note: To learn more about this technique, refer to the following Events Functions topics in the *LabVIEW Help: Create User Event , Register For Events , Generate User Event* , and Destroy User Event.

   - Select the <**channel value**>**: User Event event** case in the Event Handling Loop. Notice that this case sends a True value to the conditional terminal, which stops the loop.

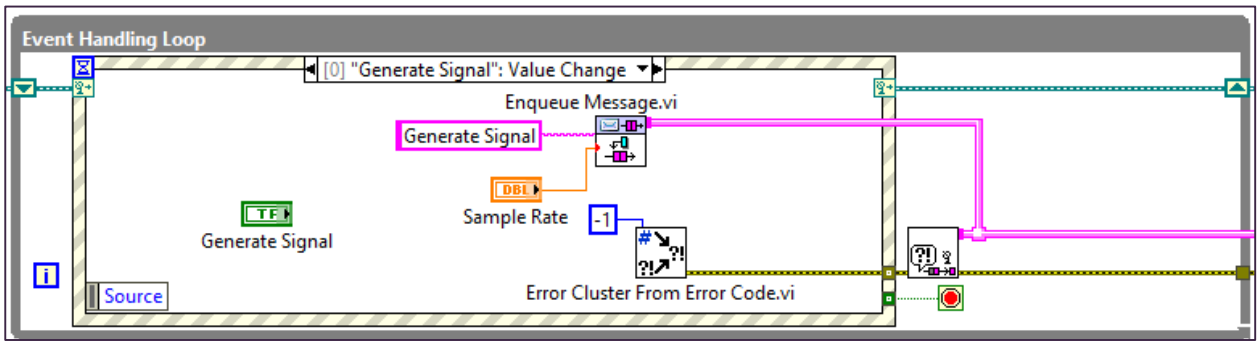
7. Test how the CMH handles an error in the Message Handling Loop.

- Select the **Generate Signal** case in the Message Handling Loop. Delete the error wire between the **Variant To Data** function and the **Merge Errors** function.
- Add an **Error Cluster From Error Code VI** to the block diagram as shown in the following figure.
- Right-click the **error code input** and select **Create» Constant**.
- Set the constant to –1.
- Wire the **error out** output of the Error Cluster From Error Code VI to the **Merge Errors** function.



- Run the VI.
- Click **Generate**. Notice that the VI handles the error by exiting.
- To follow the flow of data, run the VI again.
- Turn on the execution highlighting.
- Click **Generate**, and follow the flow of data on the block diagram.
- When finished, change the Generate Signal case back to its original code

8. Test how the CMH handles an error in the Event Handling Loop.
   - Select the **“Generate Signal”: Value Change** event case in the Event Handling Loop.
   - Add an **Error Cluster From Error Code VI** on the block diagram, as shown in the following figure.
   - Right-click the **error code** input and select **Create» Constant**.
   - Set the constant to –1.
   - Wire the **error out** output of the Error Cluster From Error Code VI to the **error in** input of the Error Handler — Event Handling Loop VI.



   - Run the VI.
   - Click **Generate**. Notice that the VI handles the error by displaying an error dialog.
   - To follow the flow of data, run the VI again.
   - Turn on the execution highlighting and click **Generate** again. Follow the flow of data on the block diagram.
   - When finished, change the **“Generate Signal”: Value Change** event case back to its original code.
9. Close the VI and the project.

Explore a Channeled Message Handler Template

Follow the instructions below to use that template in LabVIEW 2019.

1. Navigate to C:\Exercises\LabVIEW Core 2\Channeled Message Handler Template directory.
2. Copy the Channeled Message Handler folder into this directory: `C:\Program Files (x86)\National Instruments\LabVIEW 2019\ ProjectTemplates\Source\Core.`
3. Install the Channeled Message Handler.xml file into this directory: `C:\Program Files (x86)\National Instruments\LabVIEW 2019 \ProjectTemplates\MetaData.`
4. Go to the **Getting Started** window.
5. Select **File» Create Project**.
6. In the **Templates** page, select **Channeled Message Handler** and click **Finish**.
7. Close the project without saving when finished.

On the Job

Do any of your applications require using the CMH design pattern because the Event-Driven State Machine design pattern does not meet the application requirements?

_____

_____

_____

If so, continue to study the Channeled Message Handler design pattern.

End of Exercise 2-4