

## Exercise 3-1      Create a Queued Message Handler

### Goal

Create a LabVIEW project from the Queued Message Handler project template.

### Scenario

In the sprint planning meeting for this iteration, the product owner chose to implement two user stories that both apply to the overall system architecture:

- As the boiler operator, I want the application to be flexible enough that I can replace the boiler in my system or request that the application include additional boiler features, so that these updates occur with a relatively quick turnaround and minimal down-time..
- As a boiler operator, I want the user interface to remain responsive while boiler operations are occurring, so that I can always shut down the boiler in the event of an emergency.

The product owner asks you to lead the development effort for this sprint. After discussing each user story with the team, you identify the following development tasks to implement the two user stories:

- Select the top-level architecture for the application, ensuring that the user needs for scalability, readability, maintainability, parallel processing, and user interface responsiveness will be satisfied.
- Implement the basic framework for the application.
- Create a build specification to update at the end of each development sprint.

After you design and implement the code for these tasks, the next steps of the scrum process are the following:

1. Review the code with the scrum team and implement code review feedback.



**Note** For this course assume you have had successful code reviews with no major feedback.

2. Test the code to ensure that you successfully satisfy the user stories and tasks for the sprint.
3. Build the build specification and create the executable.
4. Test the executable to ensure that you can successfully deploy the code.
5. Ensure that the user stories have been implemented to the customer's satisfaction.

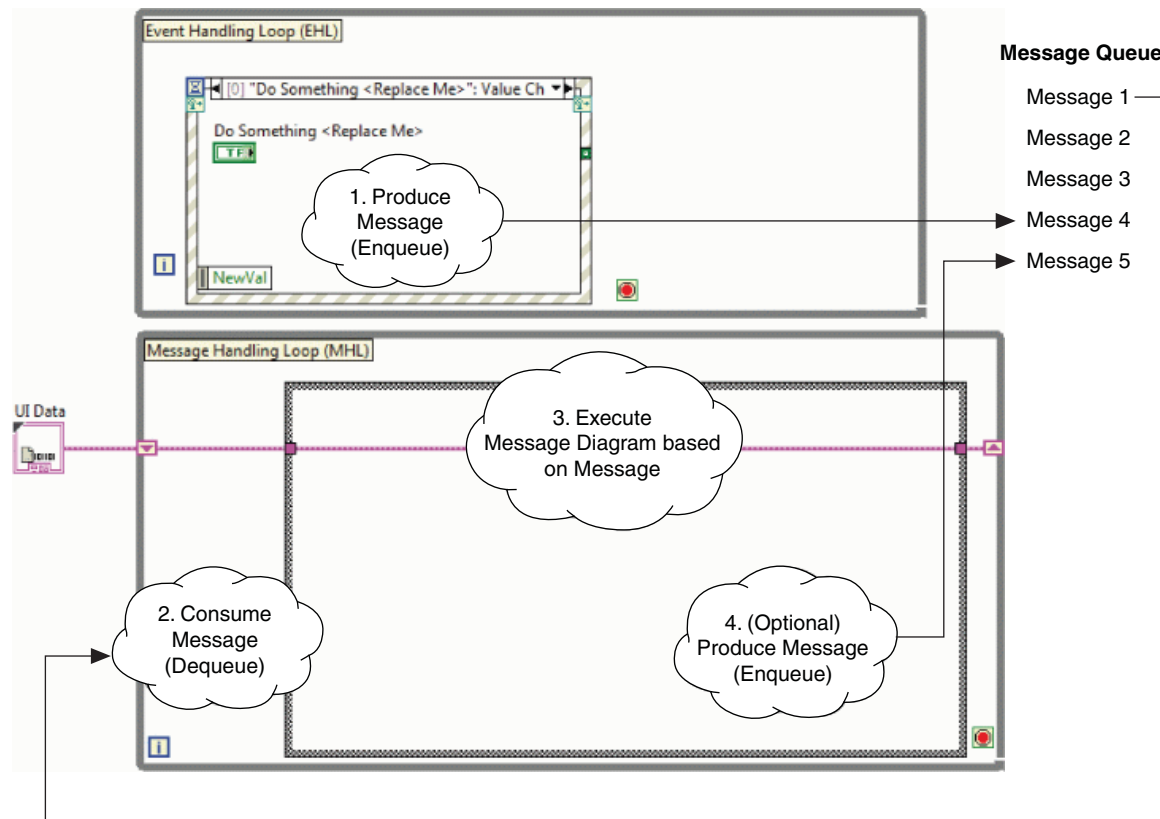


**Note** For this course, assume that the customer is satisfied with the implementation of each user story.

## Design

For the first task of this sprint, the team meets to brainstorm architecture ideas. One team member suggests using a state machine since there is a sequential flow to the user actions. Someone else points out the need for parallelism and suggests the producer/consumer pattern. The product owner recognizes that the application may need to generate messages from multiple processes so he suggests using the queued message handler template as a starting point. An overview of this template is shown in Figure 3-1.

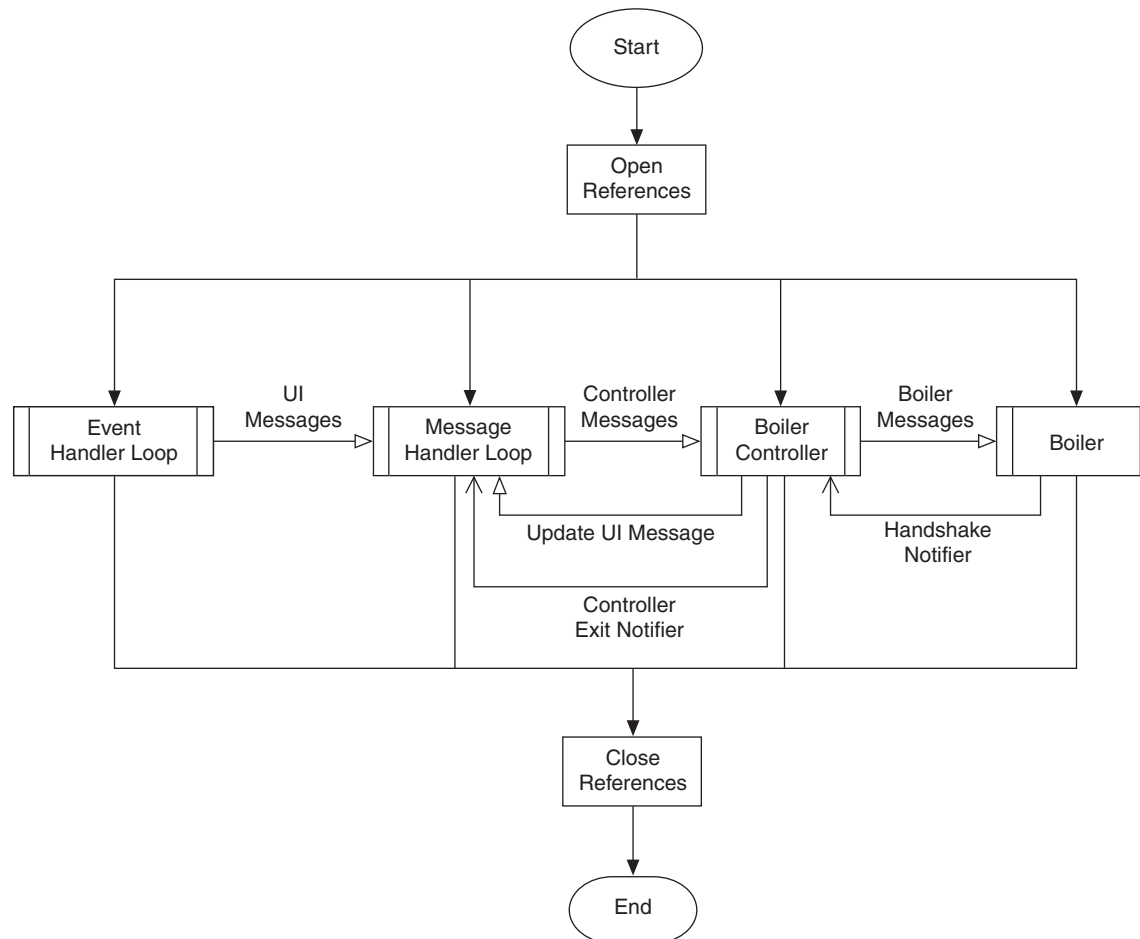
**Figure 3-1.** Queued Message Handler Overview



The Queued Message Handler (QMH) template facilitates multiple sections of code running in parallel and sending data between them. Each section of code represents a task, such as acquiring data, and is designed similarly to a state machine. Because of this design, you can divide each task into states.

In addition to the Event Handling Loop (EHL) and the Message Handling Loop (MHL) you will need a third parallel process to handle user actions. This third process also handles communication with the boiler, as shown in Figure 3-2.

**Figure 3-2. Communication Between Loops**



## Implementation

### Create a LabVIEW Project for the Application Using the Queued Message Handler Template

1. Open `Boiler Controller.lvproj` in the `<Exercises>\LabVIEW Core 3\Course Project` directory.



**Note** The `<Exercises\LabVIEW Core 3>` folder contains a project for use in this course. For future projects, LabVIEW can generate the starter files for a queued message handler application. Select **File»Create Project** in LabVIEW to display the Create Project dialog box and select Queued Message Handler in the **Templates** category.

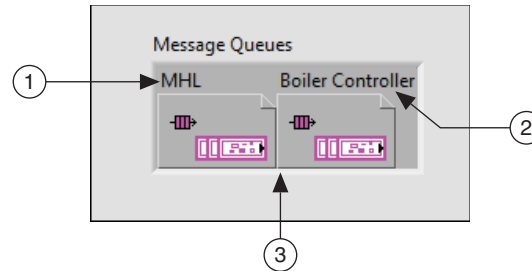
2. In the **Project Explorer** window, open and explore **Main.vi**.
  - ☐ Run the VI and click the **Something**, **Something Else**, and **Stop** buttons to see how the **Display** indicator updates.
  - ☐ Review the block diagram comments.
  - ☐ Click the **Highlight Execution** button and run the VI.
  - ☐ Click the **Something**, **Something Else**, and **Stop** buttons and notice how the execution trace changes.
  - ☐ Save and close `Main.vi`.

### Implement Basic Application Framework

1. In the **Project Explorer** window, open **Support VIs»Message Queue.lvlib»Create All Message Queues.vi**.
2. Right-click the **Message Queues Out** type def control select **Open Type Def**.

3. Modify `All Message Queues.ct1` as shown in Figure 3-3.

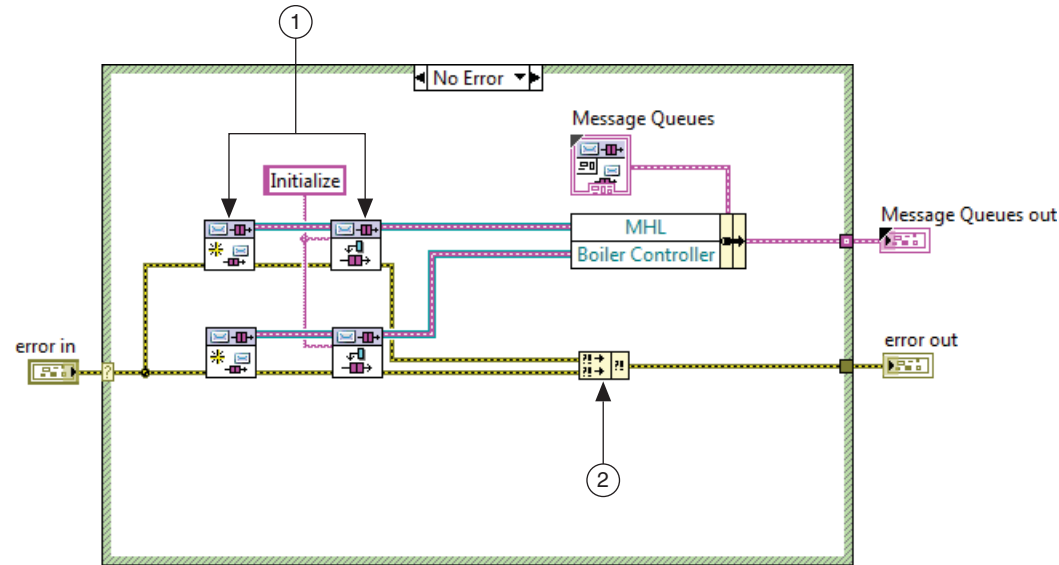
**Figure 3-3. Message Queues Control**



- 1 Rename the **UI** queue refnum to **MHL**.
  - 2 Copy and paste the **MHL** queue refnum to create a copy. Rename the new queue refnum **Boiler Controller**.
  - 3 Right-click the **Message Queues** cluster and select **Autosizing»Arrange Horizontally** from the shortcut menu.
4. Save and close `All Message Queues.ct1`.

5. Modify the Create All Message Queues VI block diagram as shown in Figure 3-4.

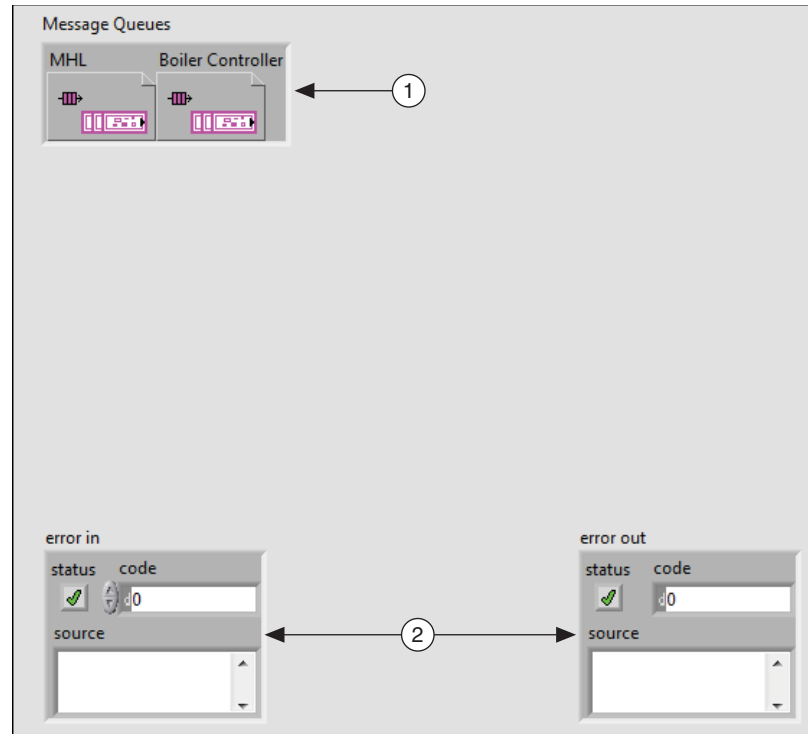
**Figure 3-4.** Create All Message Queues Block Diagram



- 1 **Obtain Message Queue and Enqueue Message**—Duplicate the Obtain Message Queue VI and Enqueue Message VI to send a Initialize message to the MHL and Boiler Controller queue refnums.
  - 2 **Merge Errors**—Merges the errors from both queues.
6. Save and close Create All Message Queues.vi.
  7. Create the Boiler Controller VI.
    - ☐ In the **Project Explorer** window, create a virtual folder under **My Computer** named Boiler Controller.
    - ☐ In the virtual folder, create a new VI and save the VI to <Exercises>\LabVIEW Core 3\Course Project\Controller\Boiler Controller.vi.

- Create the Boiler Controller VI front panel as shown in Figure 3-5.

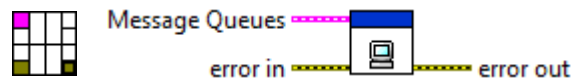
**Figure 3-5.** Boiler Controller VI Front Panel



- 1 **Message Queues type def**—Drag **All Message Queues.cti** from the **Project Explorer** window.
- 2 **Error In and Error Out**—Place an Error In and Error Out cluster on the front panel.

- Modify the icon and connector pane as shown in Figure 3-6.

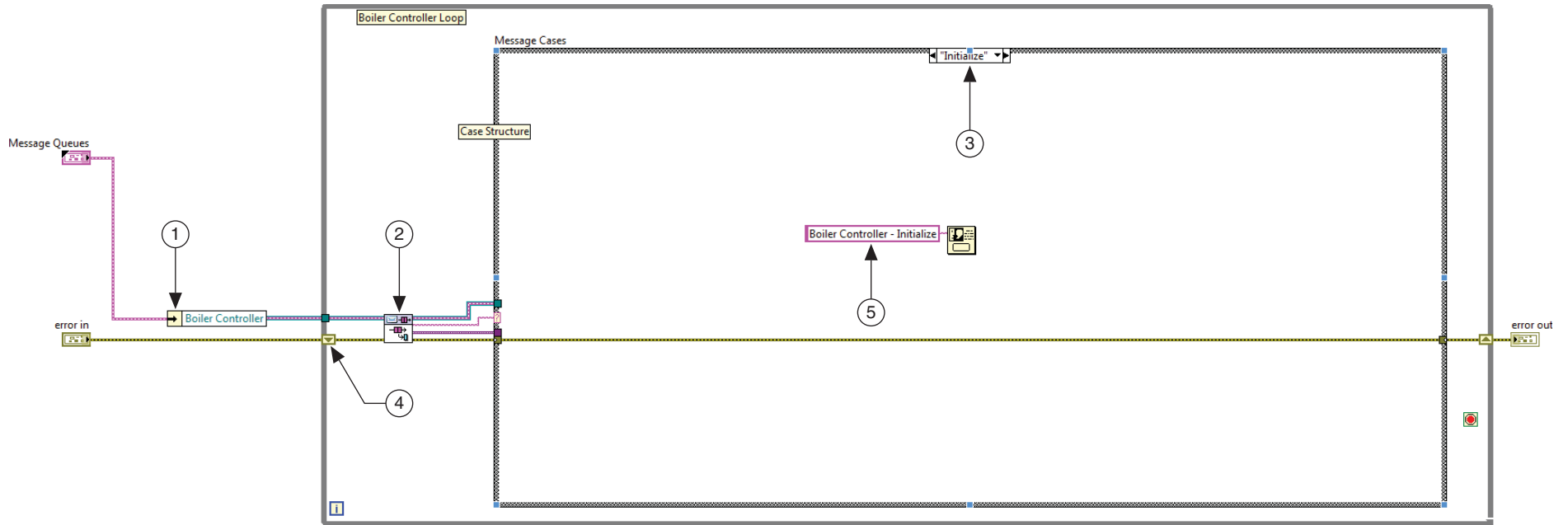
**Figure 3-6.** Boiler Controller Icon and Connector Pane



- 1 Filter glyphs by keyword, `computer`, to find an appropriate glyph for the icon.

- Modify the Boiler Controller block diagram as shown in Figure 3-7.

**Figure 3-7.** Boiler Controller VI Block Diagram Initialize Case

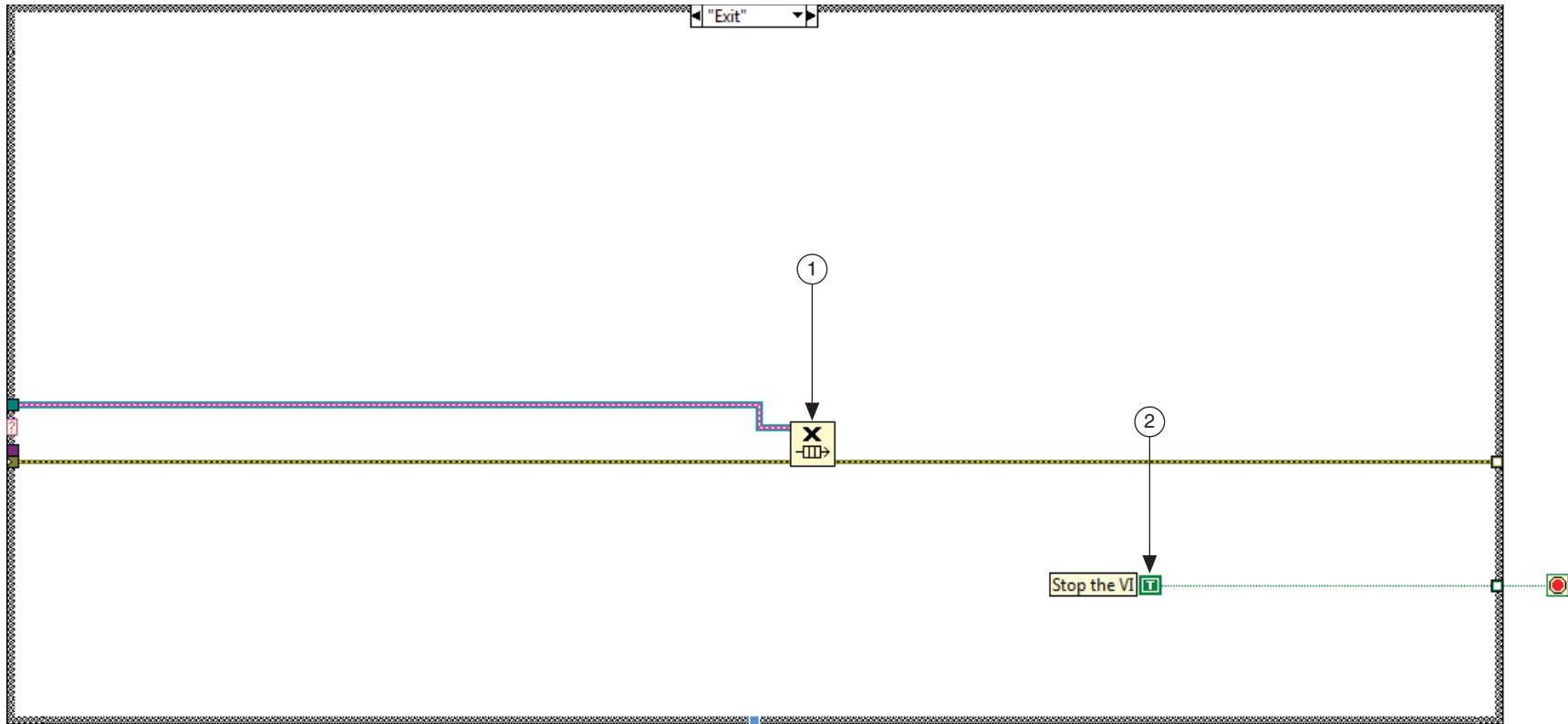


- 1 **Unbundle By Name**—Makes the Boiler Controller queue refnum available.
- 2 **Dequeue Message**—Wire the **message** output of the Dequeue Message VI to the case selector of the Case structure.
- 3 **Case structure**—Rename the True case to `Initialize`.
- 4 Right-click the tunnel and select **Replace with Shift Register** from the shortcut menu.
- 5 **One Button Dialog**—Program the message to display `Boiler Controller - Initialize`.



- Right-click the edge of the Case Structure and select **Add Case After** to create the Exit case as shown in Figure 3-8.

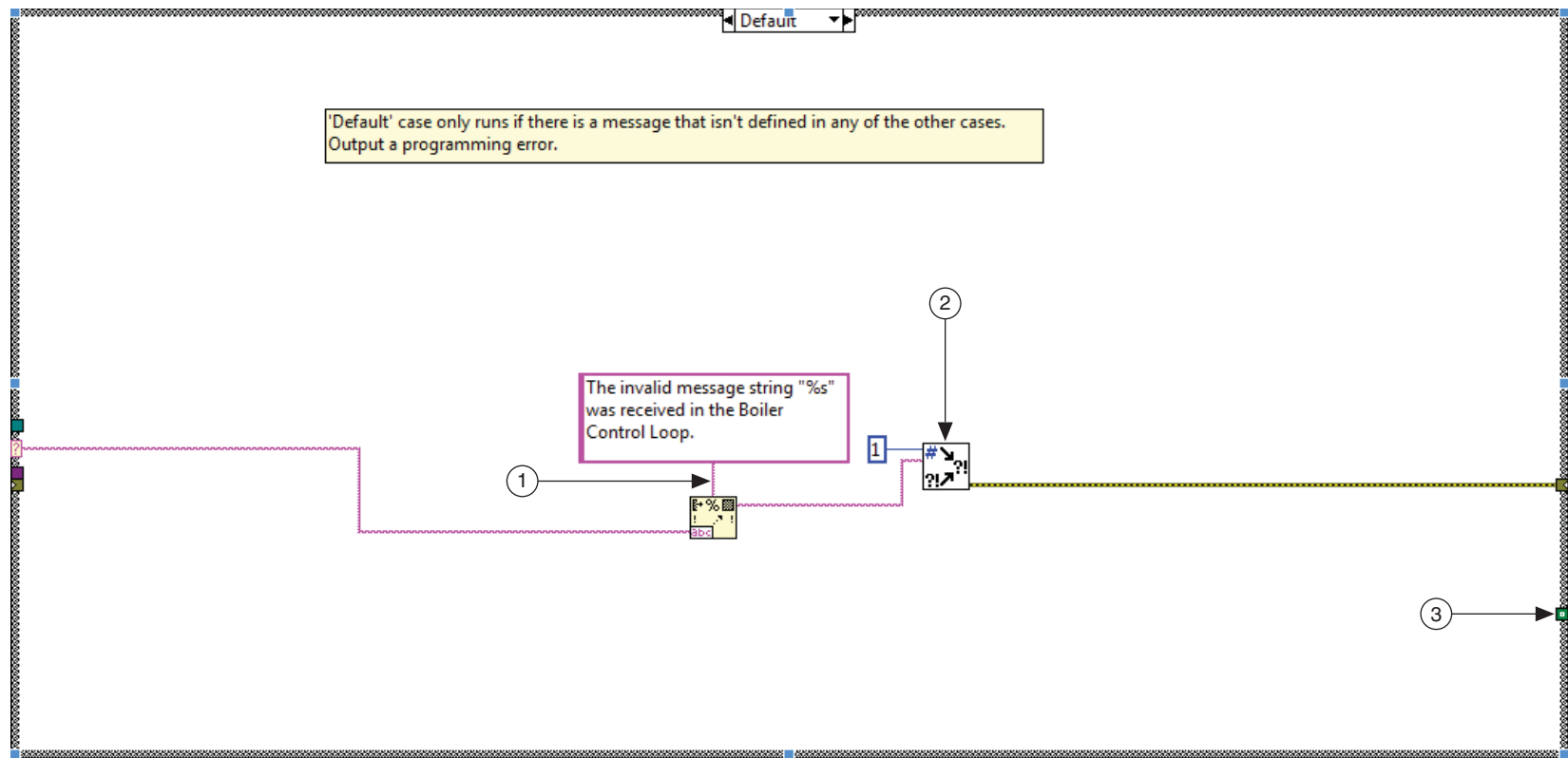
**Figure 3-8.** Boiler Controller Exit Case



- 1 **Release Queue**—The Release Queue function releases references to the queue.
- 2 **True constant**—Wire a True constant to the conditional terminal of the While Loop.

- Modify the Default case as shown in Figure 3-9.

**Figure 3-9. Boiler Controller Default Case**

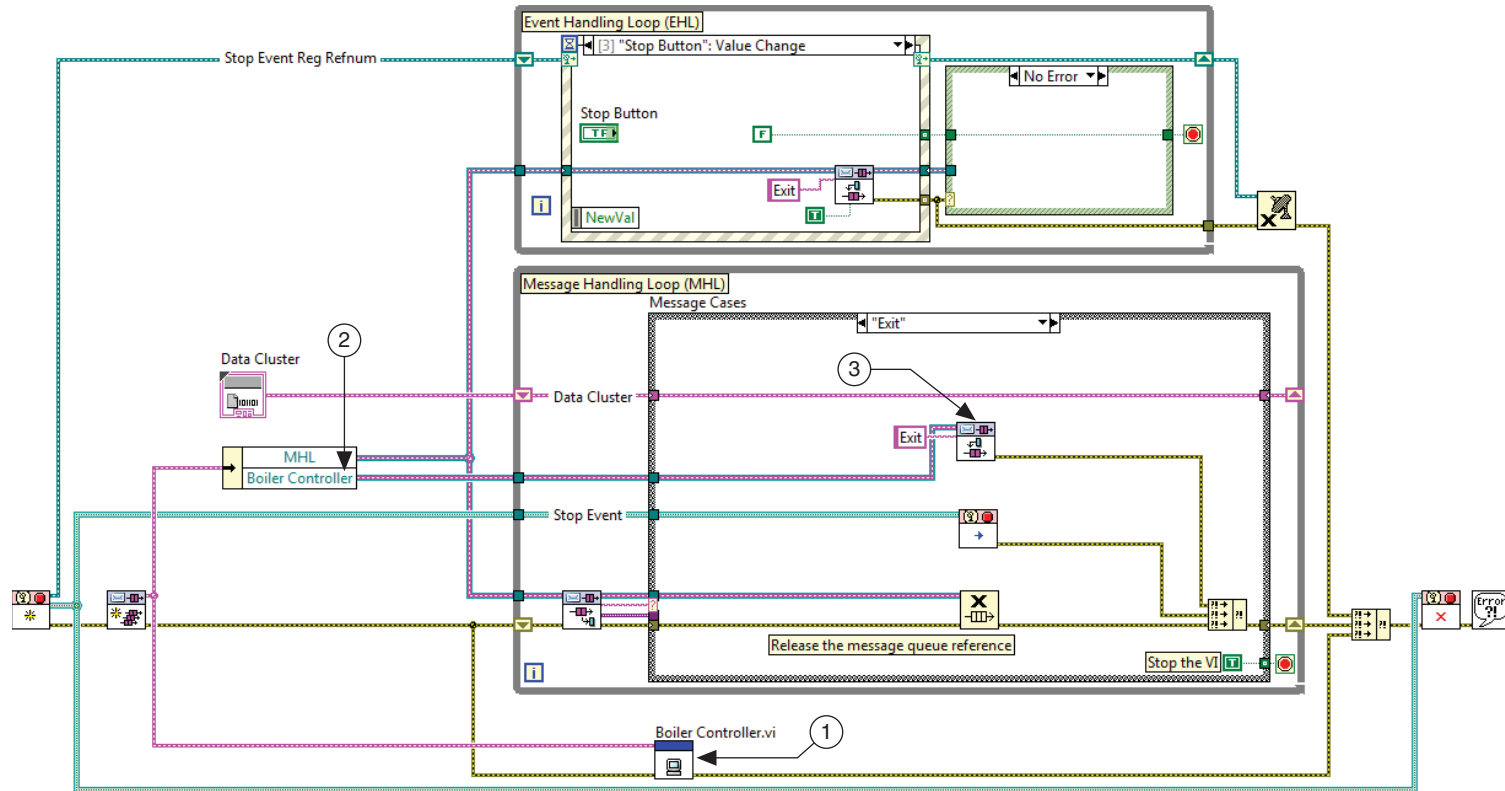


- 1 **Format Into String**—Wire a message into the **format string** input of the Format Into String function as shown.
- 2 **Error Cluster From Error Code**—Converts a code into an error cluster. Wire a constant 1 to the **error code** input.
- 3 Right-click the tunnel and select **Use Default If Unwired** from the shortcut menu to use the default value for the tunnel data type for all unwired tunnels.

- Save and close the Boiler Controller VI.

- Call the Boiler Controller VI from the Main VI as shown in Figure 3-10.

**Figure 3-10.** Main VI Calling the Boiler Controller VI

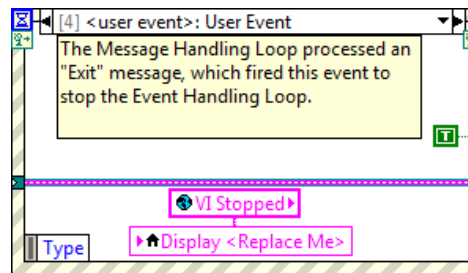



- Boiler Controller**—Drag **Boiler Controller.vi** from the **Project Explorer** window.
- Expand the **Unbundle By Name** function to display the **Boiler Controller** terminal. The Boiler Controller sends an **Exit** message to the queue during the **Exit** case of the MHL.
- Enqueue Message**—Wire an **Exit** string to the **Message** input to send an **Exit** message to the Boiler Controller Loop.

- Save the Main VI.
- Run the Main VI and verify that the **Boiler Controller - Initialize** dialog box launches.

11. Click the **Do Something** and **Do Something Else** buttons and verify that the **Display** status string updates appropriately.
12. Click the **Stop** button and verify that all loops exit successfully.
13. Use a global variable to update the display indicating that the VI has been stopped.
  - ☐ Move a copy of `Boiler System Globals.vi` from the `<Exercises>\LabVIEW Core 3\External\Support VIs to <Exercises>\LabVIEW Core 3\Course Project\support`.
  - ☐ Add `Boiler System Globals.vi` to the **Support VIs** virtual folder in the Boiler Controller project.
  - ☐ Update the User Event case of the Event Handling Loop to use the **VI Stopped** global variable as shown in Figure 3-11.

**Figure 3-11.** Event Handling Loop User Event Case Using Global Variable



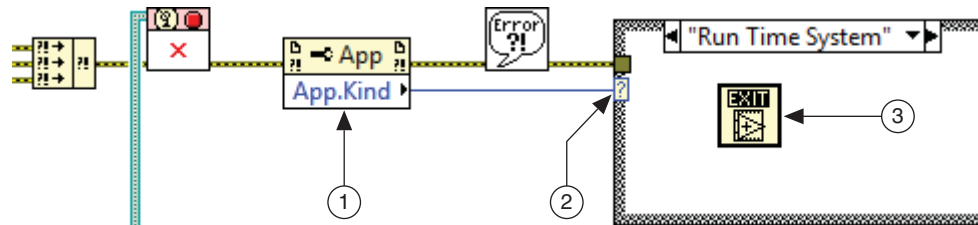
 **Note** By using global variables from which you are only reading in this project, you reduce the number of strings used on the block diagram. This is an advantage if you need to update a string that is used in multiple places because you need to update it in only one place.

14. Save Main VI.

## Creating a Build Specification and Executable

1. Modify the Main VI as shown in Figure 3-12 to close the application when you run the executable.

**Figure 3-12.** Main VI Shutdown Code



- 1 **Property Node**—Click the **Property** terminal and select **Application»Kind** from the menu.
  - 2 **Case Structure**—Wire the **App.Kind** terminal to the case selector of the Case structure. Right-click the Case structure and select **Add Case for Every Value** from the shortcut menu.
  - 3 **Quit LabVIEW**—Place a Quit LabVIEW function in the Run Time System case so that the application closes when you run it as an executable.
2. Save `Main.vi`.
  3. Update the Main Application build specification.
    - a. Under the **Build Specifications** folder in the **Project Explorer** window, right-click **Main Application** and select **Properties** from the shortcut menu.
    - b. Enter `Boiler Controller.exe` in the **Target filename** text box.
    - c. Change the **Destination Directory** to the following: `<Exercises>\LabVIEW Core 3\Course Project\builds\Boiler Controller\Main Application`.
    - d. Click the **OK** button to save the updated build specification.

## Test the Application

1. Build the executable.
  - a. Right-click the Main Application build specification and select **Build** from the shortcut menu. LabVIEW creates a new build incorporating the changes you made.
  - b. Click the **Explore** button when LabVIEW finishes the build.
2. Double-click `Boiler Controller.exe` to run the application.
3. Verify that the **Boiler Controller - Initialize** dialog box launches.
4. Click the **Do Something** and **Do Something Else** buttons and verify that the **Display** status string updates appropriately.
5. Click the **Stop** button and verify that the window closes.

## End of Exercise 3-1