LabVIEW™ Core 3 Exercises

Course Software Version 2014
November 2014 Edition
Part Number 325511D-01

Worldwide Technical Support and Product Information

`ni.com`

Worldwide Offices

Visit `ni.com/niglobal` to access the branch office websites, which provide up-to-date contact information, support phone numbers, email addresses, and current events.

National Instruments Corporate Headquarters

11500 North Mopac Expressway   Austin, Texas 78759-3504   USA   Tel: 512 683 0100

To comment on National Instruments documentation, refer to the National Instruments website at `ni.com/info` and enter the Info Code `feedback`.

Contents

Student Guide

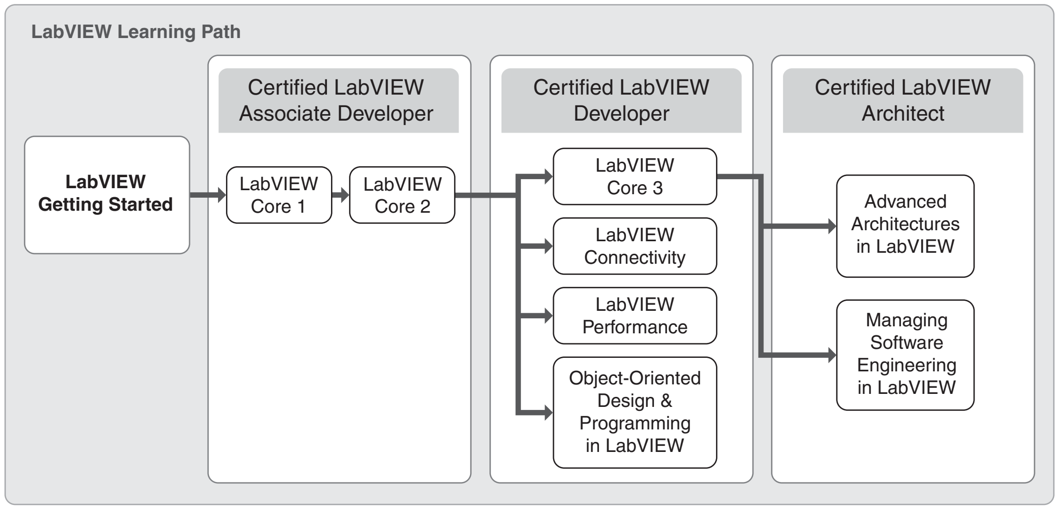
Thank you for purchasing the *LabVIEW Core 3* course kit. This course manual and the accompanying software are used in the three-day, hands-on *LabVIEW Core 3* course.

You can apply the full purchase of this course kit toward the corresponding course registration fee if you register within 90 days of purchasing the kit. Visit `ni.com/training` for online course schedules, syllabi, training centers, and class registration.

Note For course and exercise manual updates and corrections, refer to `ni.com/info` and enter the Info Code `core3`.

A. NI Certification

The *LabVIEW Core 3* course is part of a series of courses designed to build your proficiency with LabVIEW and help you prepare for NI LabVIEW certification exams. The following illustration shows the courses that are part of the LabVIEW training series. Refer to `ni.com/training` for more information about NI Certification.


LabVIEW Learning Path
Certified LabVIEW Associate Developer
Certified LabVIEW Developer
Certified LabVIEW Architect
LabVIEW Getting Started
LabVIEW Core 1
LabVIEW Core 2
LabVIEW Core 3
LabVIEW Connectivity
LabVIEW Performance
Object-Oriented Design & Programming in LabVIEW
Advanced Architectures in LabVIEW
Managing Software Engineering in LabVIEW

B. Course Description

LabVIEW Core 3 introduces you to structured practices to design, implement, document, and test LabVIEW applications. This course focuses on developing hierarchical applications that are scalable, readable, and maintainable. The processes and techniques covered in this course help reduce development time and improve application stability. By incorporating these design practices early in your development, you avoid unnecessary application redesign, increase VI reuse, and minimize maintenance costs.

This course assumes that you have taken the *LabVIEW Core 1* and *LabVIEW Core 2* courses or have equivalent experience.

This course kit is designed to be completed in sequence. The course and exercise manuals are divided into lessons, described as follows.

In the course manual, each lesson consists of the following:

- An introduction that describes the purpose of the lesson and what you will learn
- A discussion of the topics in the lesson
- A summary quiz that tests and reinforces important concepts and skills taught in the lesson

In the exercise manual, each lesson consists of the following:

- A set of exercises to reinforce the topics in the lesson
- Some lessons include optional and challenge exercise sections or additional exercises to complete if time permits

Note The exercises in this course are cumulative and lead toward developing a final application at the end of the course. If you skip an exercise, use the solution VI for that exercise, available in the `<Solutions>\LabVIEW Core 3` directory, in later exercises.

C. What You Need to Get Started

Before you use this course manual, make sure you have the following items:

- ☐ Windows XP or later installed on your computer
- ☐ LabVIEW Professional Development System 2012 or later
- ☐ *LabVIEW Core 3* course CD, containing the following folders:

Directory	Description
`Exercises`	Folder containing VIs and other files used in the course
`Solutions`	Folder containing completed course exercises

D. Installing the Course Software

Complete the following steps to install the course software.

1. Insert the course CD in your computer.
2. Follow the prompts to install the course material.

The installer places the `Exercises` and `Solutions` folders at the top level of the root directory. Exercise files are located in the `<Exercises>\LabVIEW Core 3` directory.

Tip Folder names in angle brackets, such as `<Exercises>`, refer to folders in the root directory of your computer.

E. Course Goal

Given a requirements document for a LabVIEW development project, you will follow a software development process to design, implement, document and test the key application features in a manner that satisfies requirements for readability, scalability, and maintainability.

1

Developing Successful Applications

Topics

Exercise 1-1 Review a Requirements Document

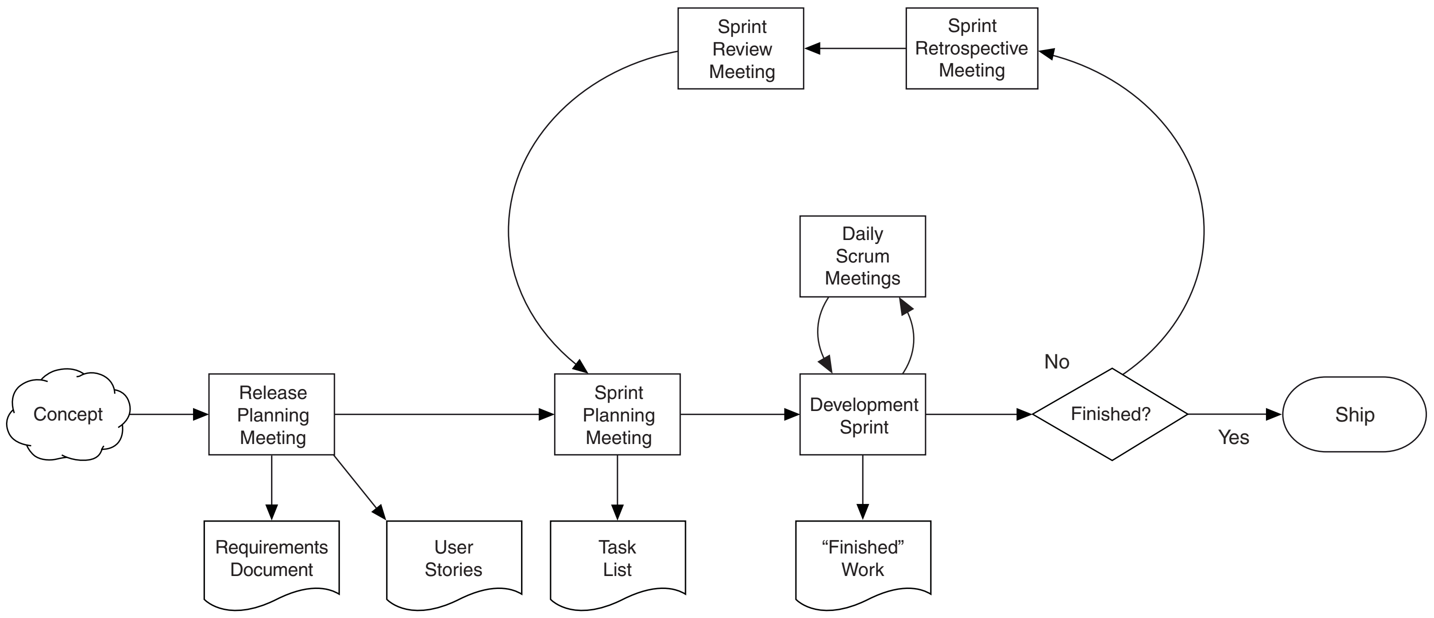
Goal

Review a requirements document and locate examples of different types of requirements.

Scenario

You are a member of a scrum team developing an application to control the startup process of a boiler. After meeting with the customer to discuss their needs and researching common boiler operations and procedures, the team developed a requirements document. Review this document for completeness and accuracy, as it is the foundation for the rest of the scrum development process, as shown in Figure 1-1.

Figure 1-1. Diagram of the Scrum Development Process



Implementation

1. Review the `Boiler Controller Requirements Document.docx` located in the `<Exercises>\LabVIEW Core 3\Release Planning` directory or in *Appendix A, Boiler Controller Requirements* of the *LabVIEW Core 3 Course Manual*, for the boiler startup controller to understand the software you create in this course.

Note Many organizations use their own techniques to create a requirements document. If your organization is not using a format for a requirements document, you can use this requirements document as a basis for other requirements documents.

2. As a group, work with the instructor and your classmates to identify at least one of the following requirements.

   ☐ Non-functional requirement

   ☐ Functional requirement

End of Exercise 1-1

Exercise 1-2 Review User Stories

Goal

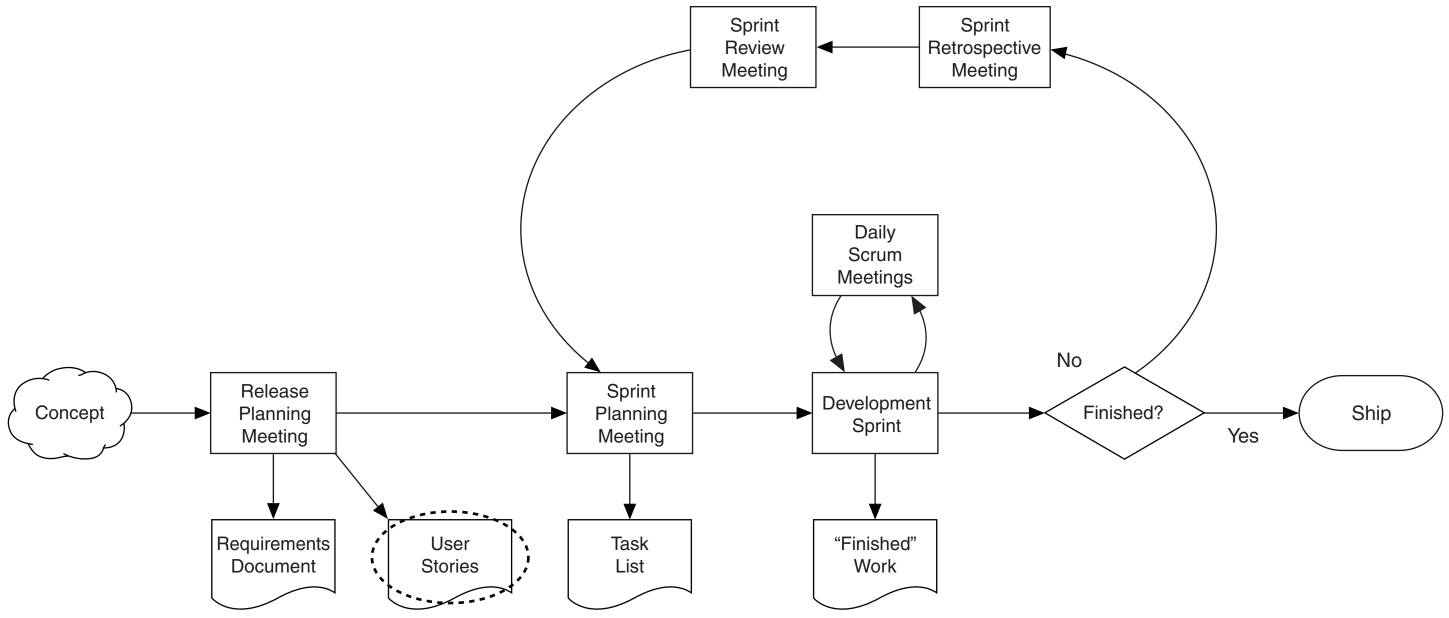
Review a list of user stories.

Scenario

After the scrum team completed the requirements document review, the product owner met with a customer representative to create a list of user stories. Each user story informally describes an aspect of the application from the perspective of the end-users and what they need the application to do as part of their job.

The product owner returns to the scrum team with a list of user stories. This list of user stories serves as the product backlog for the development of this project. Each development sprint focuses on implementing one or more user stories.

Figure 1-2. Diagram of the Scrum Development Process



Implementation

1. As a class, review and discuss the list of user stories for the boiler startup controller found in the `Boiler Controller User Stories.docx` document in the `<Exercises>\LabVIEW Core 3\Release Planning` directory, or in Appendix B*, User Stories* of the *LabVIEW Core 3 Course Manual*, to understand of the needs of the end-user. You will base development sprints on these user stories.

End of Exercise 1-2

2 Organizing the Project

Topics

Exercise 2-1 Create a Project Library

Goal

Create a project library and set access scope for files that will be distributed together.

Scenario

A member of your development team, with the help of a boiler expert, created a software model for the boiler that your application will control. To simplify distribution of these files, and to control how this code is used, you must create a project library and set the access scope for each of the files your teammate developed.

Design

Create a LabVIEW project library (`.lvlib`) for the files your teammate developed and set the access scope for each file.

Table 2-1. Boiler Project Library Files

File	Access Scope
`Boiler Configuration.ctl`	Public
`Boiler Data.ctl`	Private
`Boiler References.ctl`	Private
`Boiler UI Data.ctl`	Private
`Boiler.vi`	Public
`Change Flame Level - Next.vi`	Private
`Change Flame Level.vi`	Private

Note Do not open these files. They require code that you have not developed yet.

Implementation

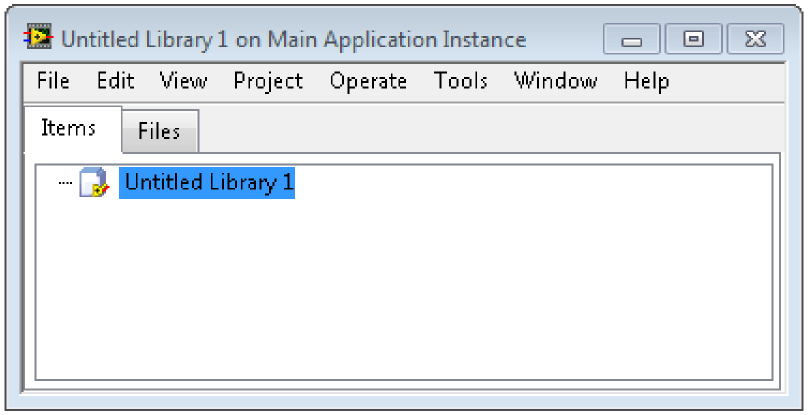
1. Open LabVIEW.

2. Select **File»New** to display the **New** dialog box.

3. In the New window, select **Other Files»Library**.

   A stand-alone project library window for the new project library file appears, as shown in Figure 2-1.

Figure 2-1. New Project Library Window


Untitled Library 1 on Main Application Instance
File Edit View Project Operate Tools Window Help
Items Files
Untitled Library 1

4. Right-click the project library icon and select **Add»File** from the shortcut menu.

5. Navigate to the `<Exercises>\LabVIEW Core 3\External\Boiler` directory from the **Select a File to Insert** dialog box.

6. Select all the files in the `Boiler` directory and click the **Add File** button.

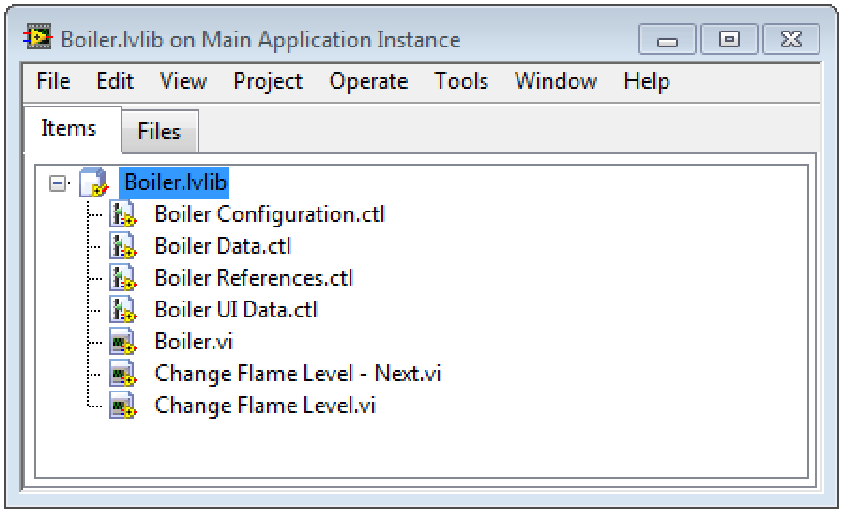
Note Click the **Ignore All** button in the dialog box that appears. LabVIEW is looking for controls that you create in future exercises.

7. Right-click the project library icon and select **Save»Save As** from the context menu.

8. Name the project library `Boiler.lvlib` and save it in the `<Exercises>\LabVIEW Core 3\External\Boiler` directory.

9. Click **Save All** when prompted to save the individual files. The project library window is shown in Figure 2-2.

Figure 2-2. Boiler.lvlib Project Window


Boiler.lvlib on Main Application Instance
File Edit View Project Operate Tools Window Help
Items
Files

Boiler.lvlib
- Boiler Configuration.ctl
- Boiler Data.ctl
- Boiler References.ctl
- Boiler UI Data.ctl
- Boiler.vi
- Change Flame Level - Next.vi
- Change Flame Level.vi

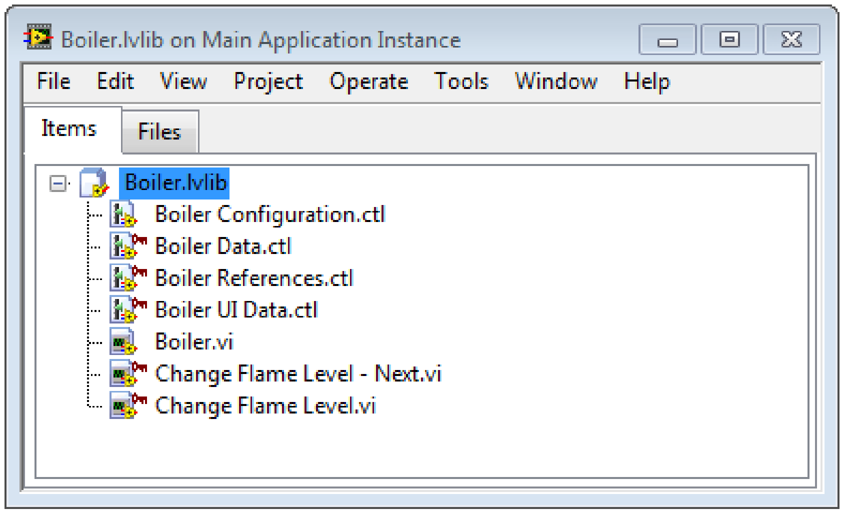
10. Set the access scope for the files in the library.

- ☐ Right-click the project library icon and select **Properties** from the shortcut menu to display the **Project Library Properties** dialog box.
- ☐ Click **Item Settings** to display the **Item Settings** page.
- ☐ Using the information in Table 2-1, click an item in the **Contents** tree to select the item and then click the proper radio button in the **Access Scope** area to assign the proper scope.
- ☐ Click the **OK** button to incorporate the changes into the project library and close the dialog box.

11. Save changes to the project library.

    Notice that the items you marked private have an icon with a key, as shown in Figure 2-3.

Figure 2-3. Boiler.lvlib Project Window with Private Access Items


Boiler.lvlib on Main Application Instance
File Edit View Project Operate Tools Window Help
Items
Files

Boiler.lvlib
Boiler Configuration.ctl
Boiler Data.ctl
Boiler References.ctl
Boiler UI Data.ctl
Boiler.vi
Change Flame Level - Next.vi
Change Flame Level.vi

12. Close the project library.

End of Exercise 2-1

Exercise 2-2 Resolve Project Conflicts

Goal

Use Project Explorer tools to resolve conflicts and manage files in a LabVIEW project.

Scenario

Conflicts can arise within a LabVIEW project if top-level VIs call incorrect versions of nested code. Applications that are saved in multiple locations as a result of archiving, backup, or division of work, can lead to the use of incorrect code and broken applications.

In this exercise, you examine a LabVIEW project that contains conflicts and use the tools in the Project Explorer to resolve the conflicts and manage the project.

Design

The project in this exercise contains the following conflicts:

- Two VIs within the project have the same name, `Generate Signal.vi`.
- A VI in the project calls a subVI outside the project, `Log to File.vi`, that has the same name as a VI within the project.

Implementation

Part I: Resolving Conflicts

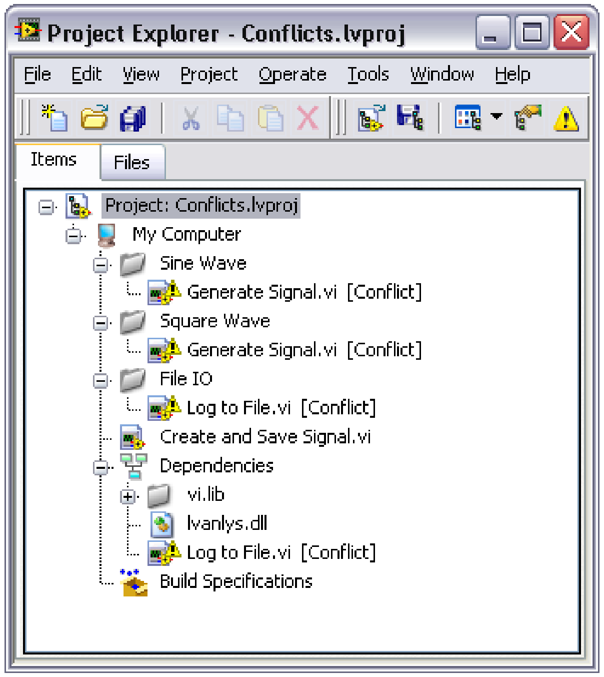
1. Explore a LabVIEW project containing conflicts.

   ☐ Open `Conflicts.lvproj` in the `<Exercises>\LabVIEW Core 3\Project Explorer Tools` directory.

   ☐ Expand **Sine Wave**, **Square Wave**, **File IO**, and **Dependencies** in the project tree, as shown in Figure 2-4.

   Notice that LabVIEW has determined that various VIs have conflicts. A conflict is a potential cross-link that occurs when LabVIEW tries to load a VI that has the same qualified name as an item already in the project. When there is a conflict, it is unclear which VI a calling VI should reference.

Figure 2-4. LabVIEW Project with Conflicts


Project Explorer - Conflicts.lvproj
File Edit View Project Operate Tools Window Help
Items
Files

```
Project: Conflicts.lvproj
  My Computer
    Sine Wave
      Generate Signal.vi [Conflict]
    Square Wave
      Generate Signal.vi [Conflict]
    File IO
      Log to File.vi [Conflict]
    Create and Save Signal.vi
    Dependencies
      vi.lib
      lvanlys.dll
      Log to File.vi [Conflict]
    Build Specifications
```

☐ Double-click **Generate Signal.vi** in the **Sine Wave** virtual folder.

☐ Run the VI and observe that this VI generates a sine wave.

☐ Close the VI. You do not have to save any changes.

☐ Double-click **Generate Signal.vi** in the **Square Wave** virtual folder.

☐ Run the VI and observe that this VI generates a square wave.

☐ Close the VI.

2. View the file paths of the items in the project tree.

   ☐ In the **Project Explorer** window, select **Project»Show Item Paths**.

   Tip The best way to determine if cross-linking exists is to view the full path to the item. Viewing file paths is often the first step in resolving cross-linking conflicts. You can rename or move files as needed, but first you must determine which file is the correct file. Displaying the full path of an item helps you determine the correct file.

   ☐ Notice that the Generate Signal VI is actually two different VIs with the same name.

   Conflicts might occur if you try to include VIs of the same name because only one VI of a given name can be in memory at a time. If you have a VI with a specific name in memory and you attempt to load another VI that references a subVI of the same name, the VI links to the VI in memory.

3. Determine which VIs in the project call the conflicting VIs.

   ☐ Right-click **Generate Signal.vi** in the **Sine Wave** virtual folder and select **Find»Callers** from the shortcut menu.
   In the project tree, LabVIEW highlights the Create and Save Signal VI because it calls the Generate Signal VI as a subVI.

   ☐ Right-click **Generate Signal.vi** in the **Square Wave** virtual folder and select **Find»Callers** from the shortcut menu.
   Even though this Generate Signal VI has no callers in the project, you note that the Generate Signal VI in the `Square Wave` directory performs a different function than the Generate Signal VI in the `Sine Wave` directory. Therefore, renaming one or both files is an appropriate way to resolve the conflict.

4. Manually rename the conflicting files to remove the conflict.

   ☐ In the **Project Explorer** window, right-click **Generate Signal.vi** in the **Sine Wave** folder, and select **Rename**.

   ☐ Rename the VI `Sine Wave - Generate Signal.vi` and click **OK**.

   ☐ Click **Save** when LabVIEW prompts you to save the changes made to the calling VI, `Create and Save Signal.vi`. LabVIEW updates the Create and Save Signal VI to preserve the links to the subVI.

   ☐ In the **Square Wave** folder, right-click **Generate Signal.vi** and select **Rename**.

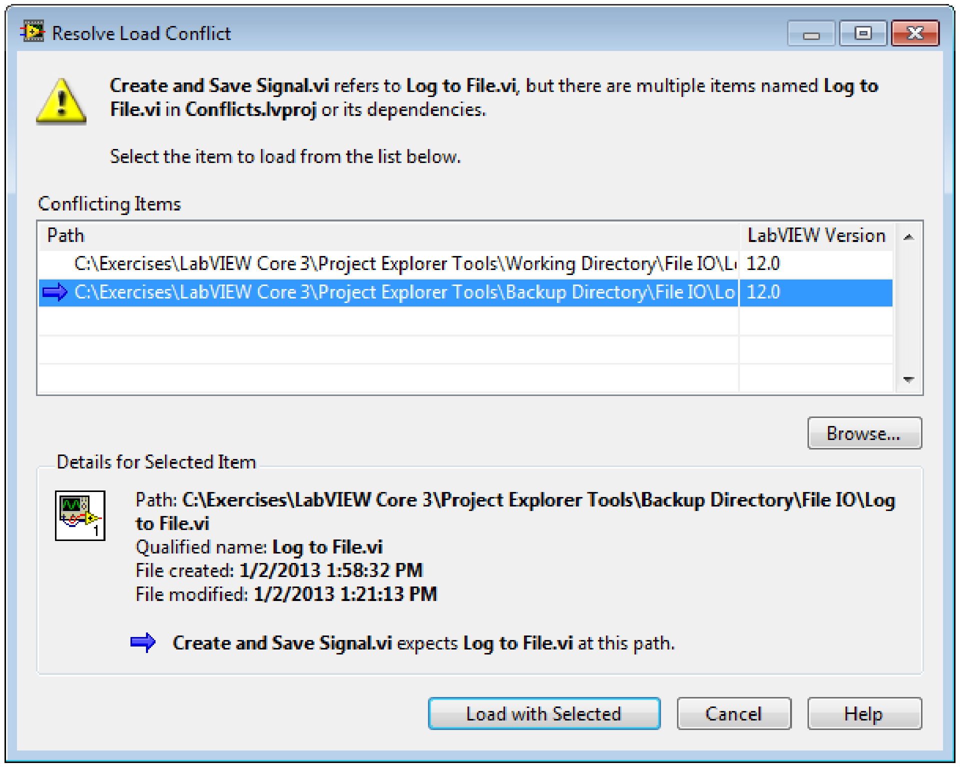
   ☐ Rename the VI `Square Wave - Generate Signal.vi` and click **OK**.

   Tip You also can rename files from the **Files** page view of the project.

5. Resolve a conflict using the **Resolve Project Conflicts** dialog box.

- ☐ Notice that there is copy of `Log to File.vi` in the **File IO** virtual folder and a copy of `Log to File.vi` in **Dependencies**, which indicates that a file within the project calls the second copy. The file paths of these two VIs are different, as shown in the **Paths** section of the **Project Explorer** window.
- ☐ Right-click each copy of `Log to File.vi` and select **Find»Callers** to determine which file, if any, within the project calls each copy.
- ☐ Double-click **Create and Save Signal.vi** in the LabVIEW Project. The **Resolve Load Conflict** dialog box appears as shown in Figure 2-5 and prompts you to select the subVI you want the caller to reference.

Figure 2-5. Resolve Load Conflict Dialog Box



Resolve Load Conflict

Create and Save Signal.vi refers to **Log to File.vi**, but there are multiple items named **Log to File.vi** in **Conflicts.lvproj** or its dependencies.

Select the item to load from the list below.

Conflicting Items

Path	LabVIEW Version
C:\Exercises\LabVIEW Core 3\Project Explorer Tools\Working Directory\File IO\L	12.0
C:\Exercises\LabVIEW Core 3\Project Explorer Tools\Backup Directory\File IO\Lo	12.0

Details for Selected Item

Path: **C:\Exercises\LabVIEW Core 3\Project Explorer Tools\Backup Directory\File IO\Log to File.vi**
Qualified name: **Log to File.vi**
File created: **1/2/2013 1:58:32 PM**
File modified: **1/2/2013 1:21:13 PM**

Create and Save Signal.vi expects **Log to File.vi** at this path.

- ☐ In the **Resolve Load Conflict** dialog box, select the Log to File VI in `Working Directory` and click the **Load with Selected** button.
- ☐ Click the **Show Details** button in the **Load Summary Warning** dialog box, which informs you that the Log to File subVI was loaded from a different location.
- ☐ Examine the information display in the **Load and Save Warning List** dialog box and click the **Close** button.
- ☐ Review the items in the **Project Explorer** window. All conflicts in the project should now be resolved.

6. Fix the errors in the Create and Save Signal VI.
   - ☐ The Create and Save Signal VI has unlinked subVIs after resolving the conflicts.
   - ☐ To relink the subVIs, open the block diagram, right-click each subVI, and select **Relink to SubVI** from the shortcut menu.
   - ☐ Save and close the Create and Save Signal VI.
7. Save the project.

Part II: Exploring Other File Management Tools

1. Use auto-populating folders in the project.
   - ☐ Right-click the **Sine Wave** virtual folder in the project tree and select **Convert to Auto-populating Folder** from the shortcut menu.
   - ☐ Navigate to the `<Exercises>\LabVIEW Core 3\Project Explorer Tools\Working Directory\Sine Wave` directory and click **Select Folder**. Notice in the project tree that the Sine Wave folder icon changes to indicate that the folder is now set to auto-populate.

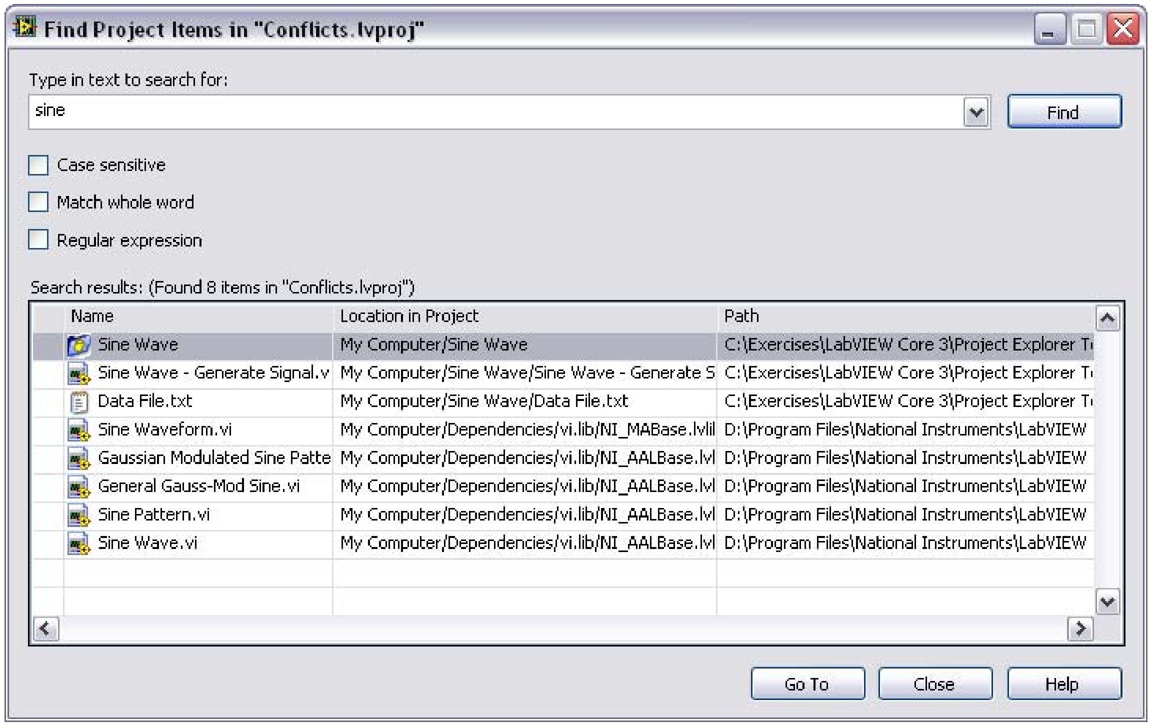
Note In auto-populate mode, the contents of the project folder reflect the hierarchy of the specified folder on disk, as well as any changes that are made outside the development environment.

   - ☐ Click the Windows **Start** button and select **All Programs»Accessories»Notepad** to launch Notepad.
   - ☐ In Notepad, select **File»Save** and save the file as `Data File.txt` in the `<Exercises>\LabVIEW Core 3\Project Explorer Tools\Working Directory\Sine Wave` directory.
   - ☐ Close Notepad.
   - ☐ Notice that `Data File.txt` has been added to the **Sine Wave** auto-populating folder on the **Items** page of the **Project Explorer** window.

2. Search for project items.

- ☐ In the **Project Explorer** window, select **Edit»Find Project Items**.
- ☐ In the **Find Project Items** dialog box, enter `sine` in the textbox, as shown in Figure 2-6, and click **Find**.

Figure 2-6. Find Project Items Dialog Box



Type in text to search for: sine

☐ Case sensitive
☐ Match whole word
☐ Regular expression

Search results: (Found 8 items in "Conflicts.lvproj")

Name	Location in Project	Path
Sine Wave	My Computer/Sine Wave	C:\Exercises\LabVIEW Core 3\Project Explorer T
Sine Wave - Generate Signal.v	My Computer/Sine Wave/Sine Wave - Generate S	C:\Exercises\LabVIEW Core 3\Project Explorer T
Data File.txt	My Computer/Sine Wave/Data File.txt	C:\Exercises\LabVIEW Core 3\Project Explorer T
Sine Waveform.vi	My Computer/Dependencies/vi.lib/NI_MABase.lvlil	D:\Program Files\National Instruments\LabVIEW
Gaussian Modulated Sine Patte	My Computer/Dependencies/vi.lib/NI_AALBase.lvl	D:\Program Files\National Instruments\LabVIEW
General Gauss-Mod Sine.vi	My Computer/Dependencies/vi.lib/NI_AALBase.lvl	D:\Program Files\National Instruments\LabVIEW
Sine Pattern.vi	My Computer/Dependencies/vi.lib/NI_AALBase.lvl	D:\Program Files\National Instruments\LabVIEW
Sine Wave.vi	My Computer/Dependencies/vi.lib/NI_AALBase.lvl	D:\Program Files\National Instruments\LabVIEW

- ☐ Select **Sine Wave - Generate Signal.vi** and click the **Go To** button. **Sine Wave - Generate Signal.vi** is now highlighted in the **Project Explorer** window.

3. Save and close the project.

End of Exercise 2-2

3

Creating an Application Architecture

Topics

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.

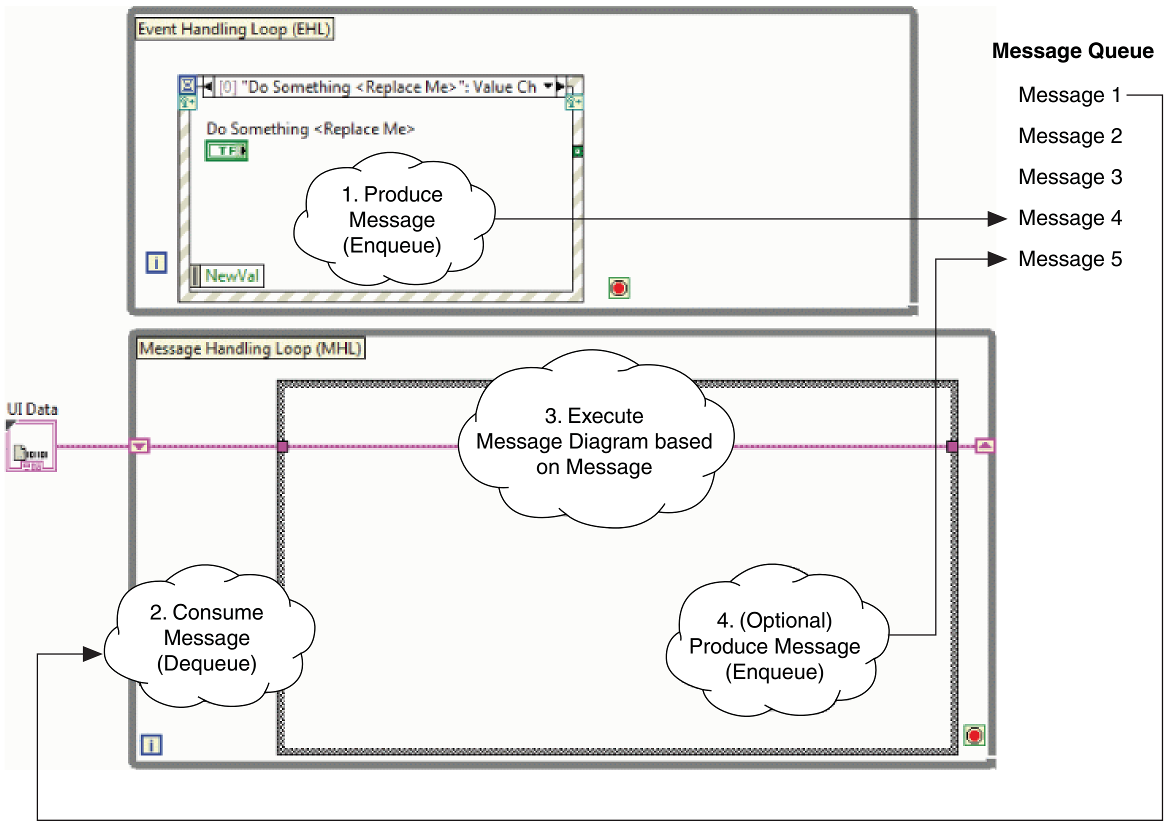
2. Test the code to ensure that you successfully satisfy the user stories and tasks for the sprint.
3. Build the build specification and create the executable.
4. Test the executable to ensure that you can successfully deploy the code.
5. Ensure that the user stories have been implemented to the customer's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

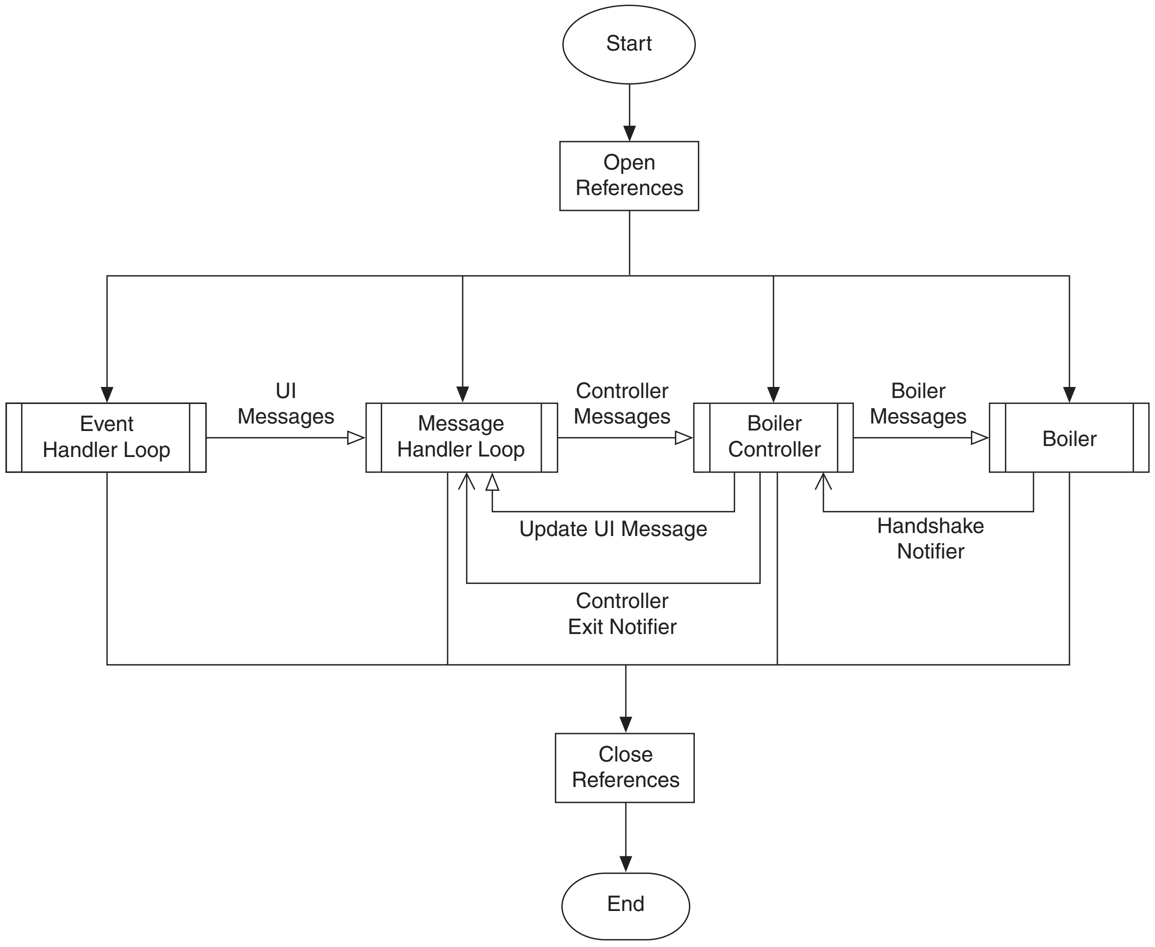
Figure 3-1. Queued Message Handler Overview


Event Handling Loop (EHL)
[0] "Do Something <Replace Me>": Value Ch
Do Something <Replace Me>
NewVal
1. Produce Message (Enqueue)
Message Queue
Message 1
Message 2
Message 3
Message 4
Message 5
Message Handling Loop (MHL)
UI Data
3. Execute Message Diagram based on Message
2. Consume Message (Dequeue)
4. (Optional) Produce Message (Enqueue)

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`.

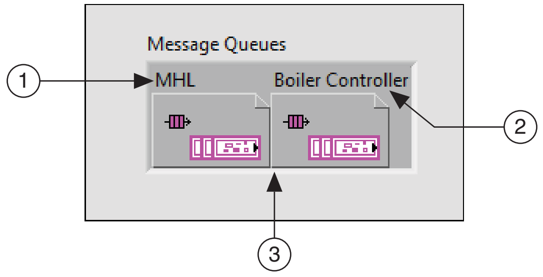
Implement Basic Application Framework

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.ctl` as shown in Figure 3-3.

Figure 3-3. Message Queues Control

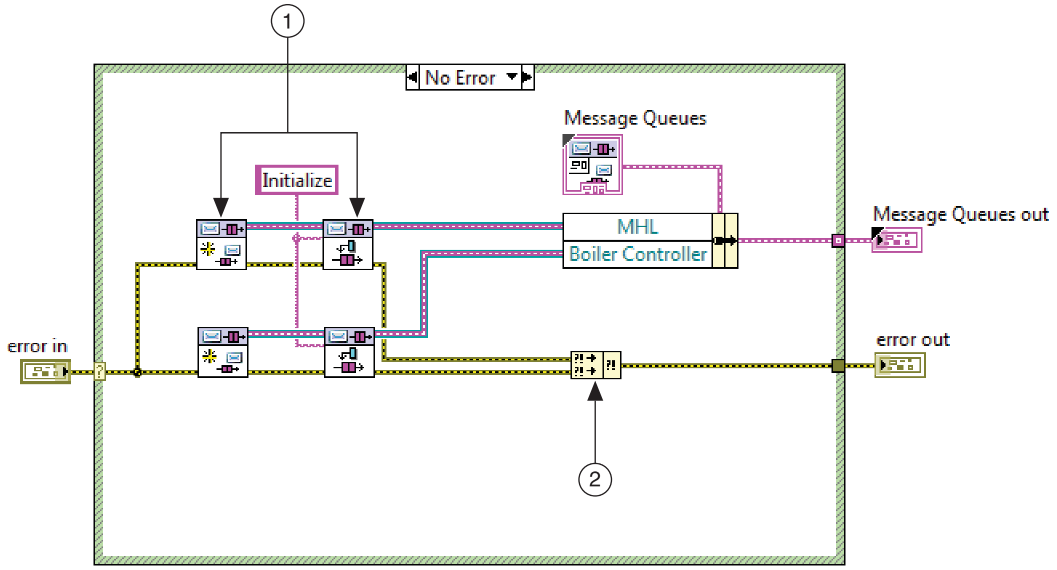


1	Rename the **UI** queue refnum to `MHL`.
2	Copy and paste the **MHL** queue refnum to create a copy. Rename the new queue refnum `Boiler Controller`.
3	Right-click the **Message Queues** cluster and select **Autosizing»Arrange Horizontally** from the shortcut menu.

4. Save and close `All Message Queues.ctl`.

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.

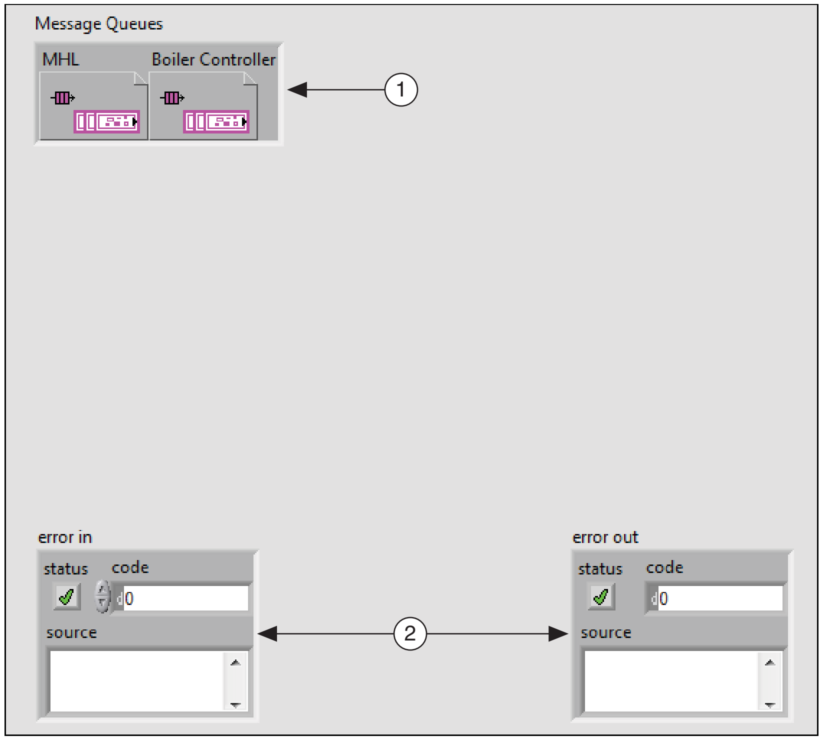
6. Save and close `Create All Message Queues.vi`.

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.

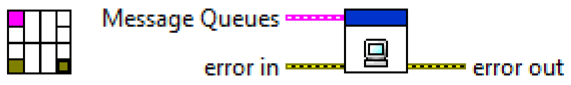
Figure 3-5. Boiler Controller VI Front Panel



1	**Message Queues type def**—Drag **All Message Queues.ctl** 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.

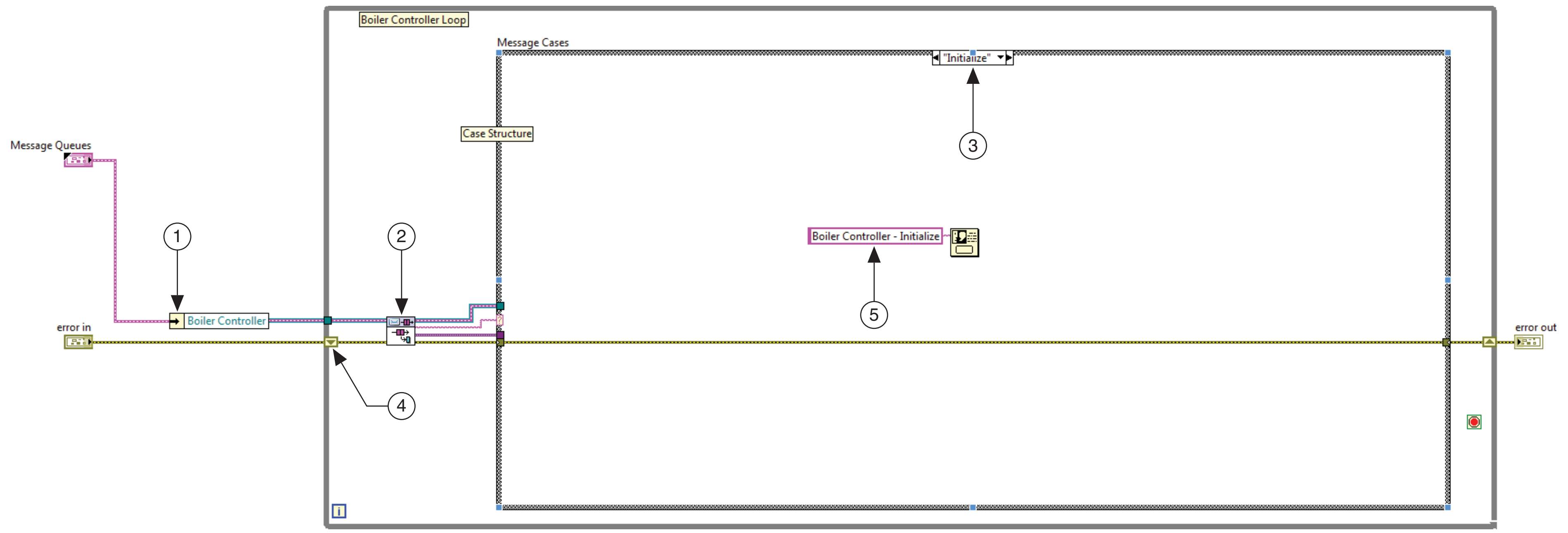
Figure 3-6. Boiler Controller Icon and Connector Pane



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.

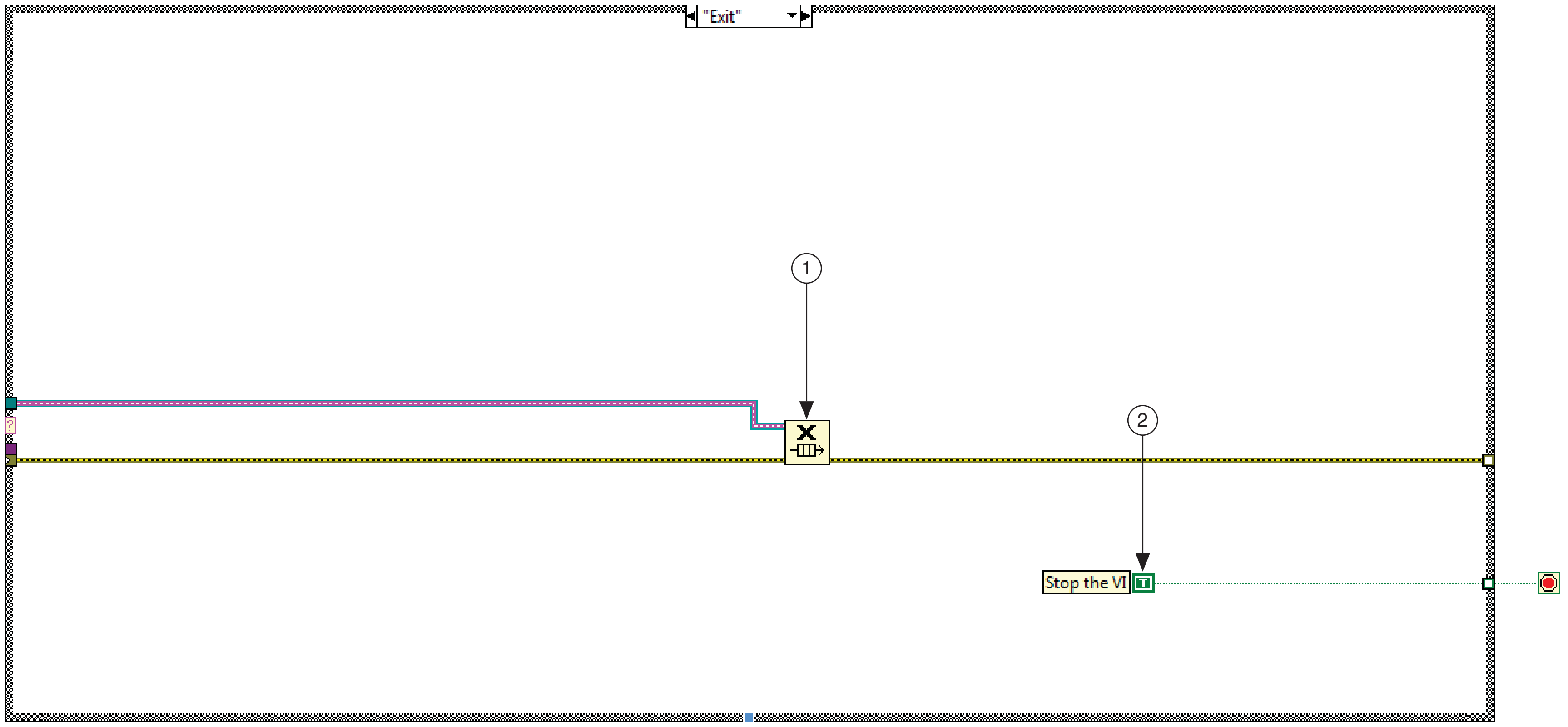
3 **Case structure**—Rename the True case to `Initialize`.

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

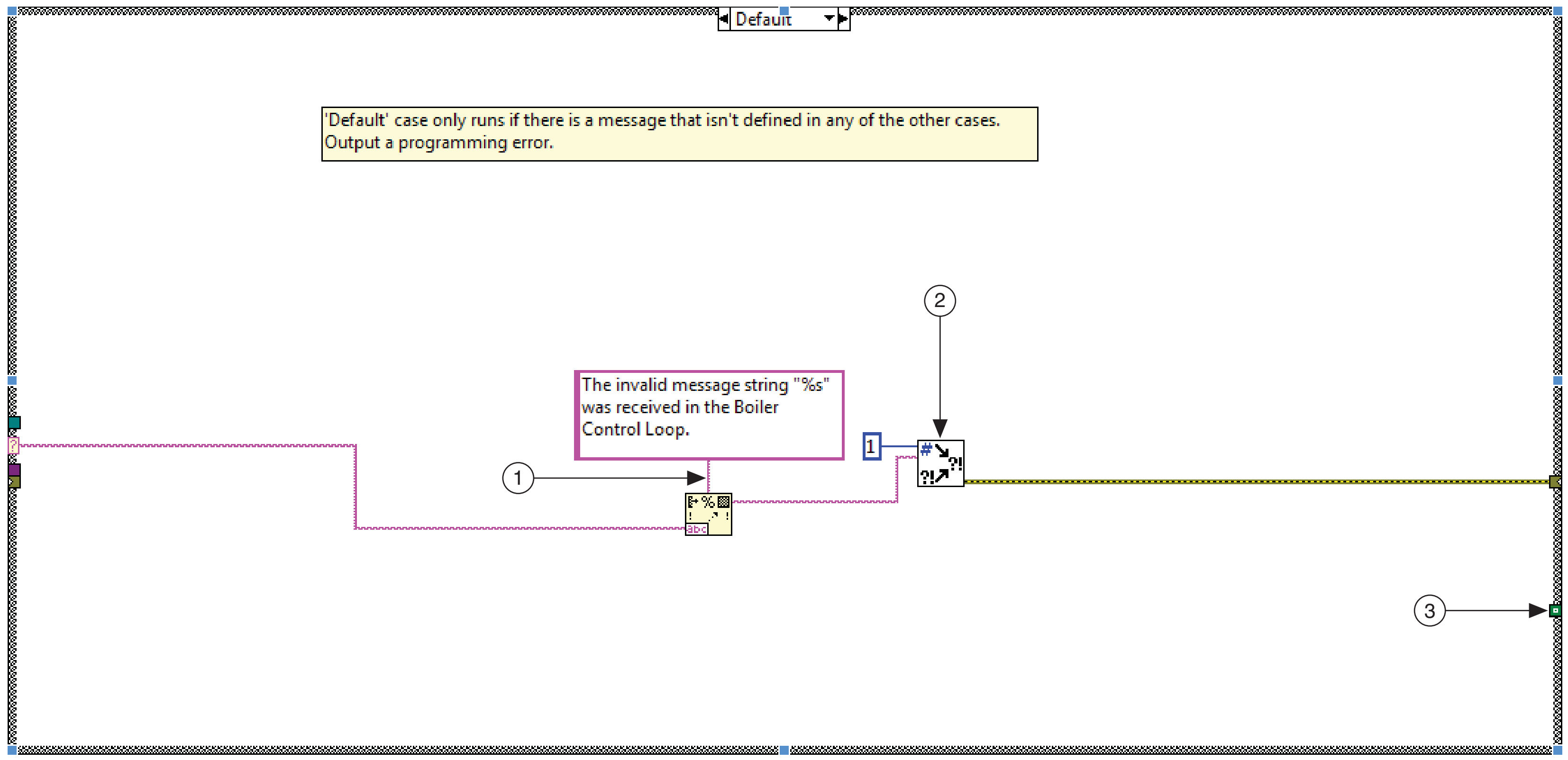


1 **Release Queue**—The Release Queue function releases references to the queue.

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


'Default' case only runs if there is a message that isn't defined in any of the other cases. Output a programming error.

The invalid message string "%s" was received in the Boiler Control Loop.

1 **Format Into String**—Wire a message into the **format string** input of the Format Into String function as shown.

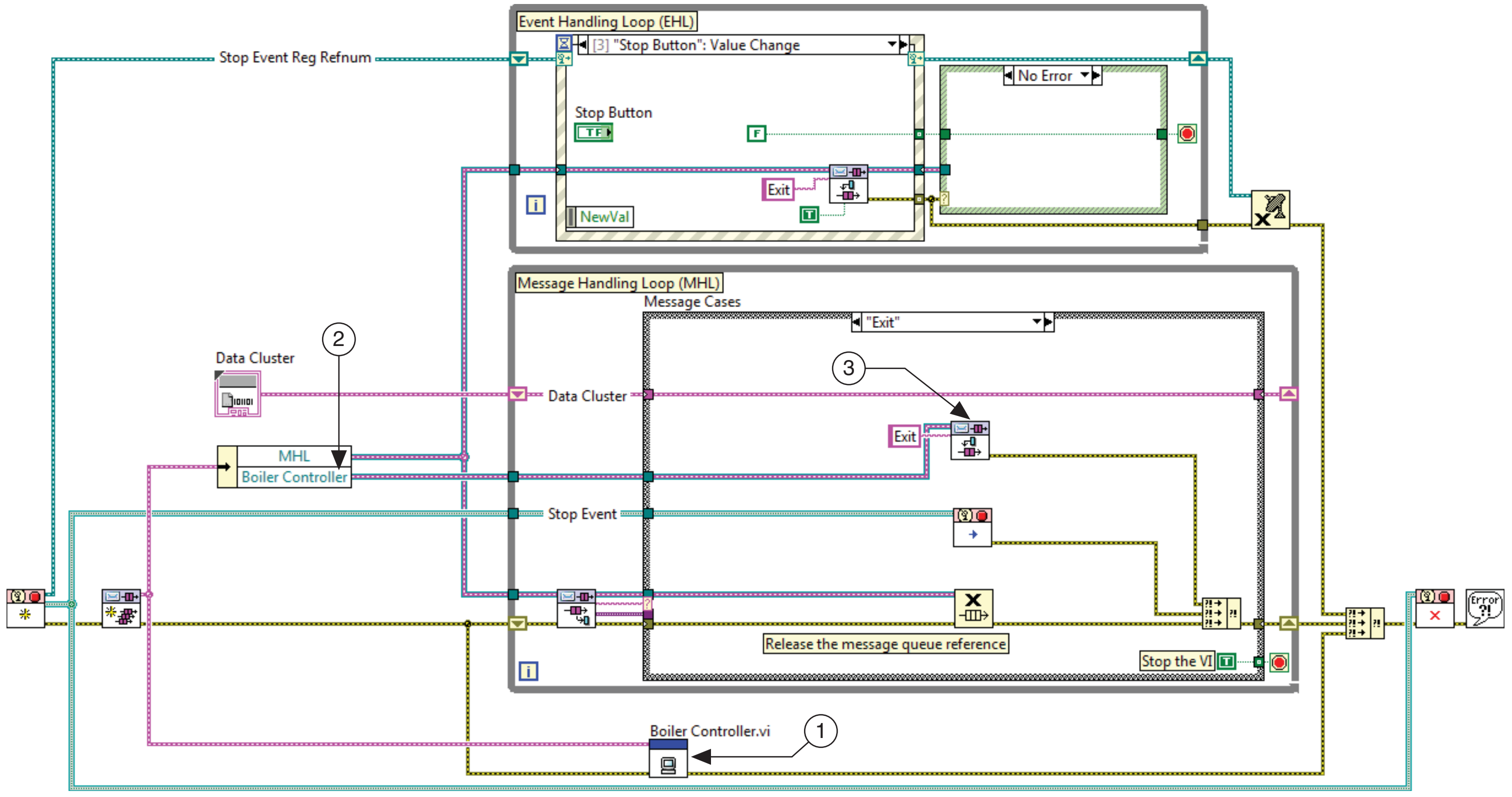
2 **Error Cluster From Error Code**—Converts a code into an error cluster. Wire a constant `1` to the **error code** input.

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.

8. 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



1 **Boiler Controller**—Drag **Boiler Controller.vi** from the **Project Explorer** window.

2 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.

3 **Enqueue Message**—Wire an `Exit` string to the **Message** input to send an Exit message to the Boiler Controller Loop.

9. Save the Main VI.

10. 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.

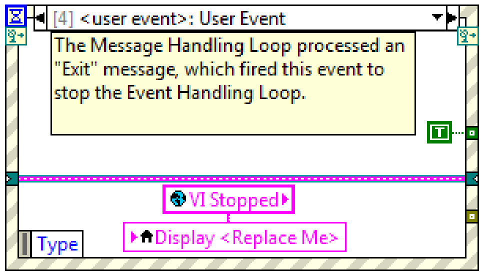
13. Use a global variable to update the display indicating that the VI has been stopped.

    - ☐ 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



The Message Handling Loop processed an "Exit" message, which fired this event to stop the Event Handling Loop.

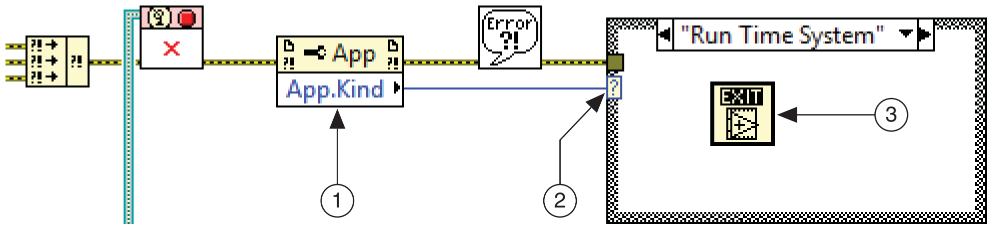
Note By using global variables from which you are only reading in this project, you reduce the number of strings used on the block diagram. This is an advantage if you need to update a string that is used in multiple places because you need to update it in only one place.

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

Exercise 3-2 Handshaking with Notifiers

Goal

Use notifiers to implement handshaking between parallel processes.

Scenario

In the sprint planning meeting for this iteration, the product owner chose to implement two user stories that apply to the interaction of the boiler controller application with the boiler:

- As a boiler operator, I want to ensure that when I shut down the system, the boiler controller does not suspend execution until the boiler finishes shutting down, so that the boiler controller is always aware of the current state of the boiler.
- As a boiler operator, I want to test the functionality of the application by running it with code that simulates the behavior of a boiler, so that I can verify correct operation before I implement the system.

After discussing each user story with the team, you have identified the following development tasks that must be completed to implement these user stories:

1. Obtain the notifiers that will handle handshaking among the Message Handling Loop of `Main.vi`, `Boiler Controller.vi` and the boiler to ensure that the MHL does not turn off until `Boiler Controller.vi` shuts down, which does not happen until the boiler shuts down.
2. Integrate `Boiler.lvlib` into the application.
3. Synchronize the shutdown operation among all four loops.
4. Release the handshaking notifiers when the application is shutting down.

Design

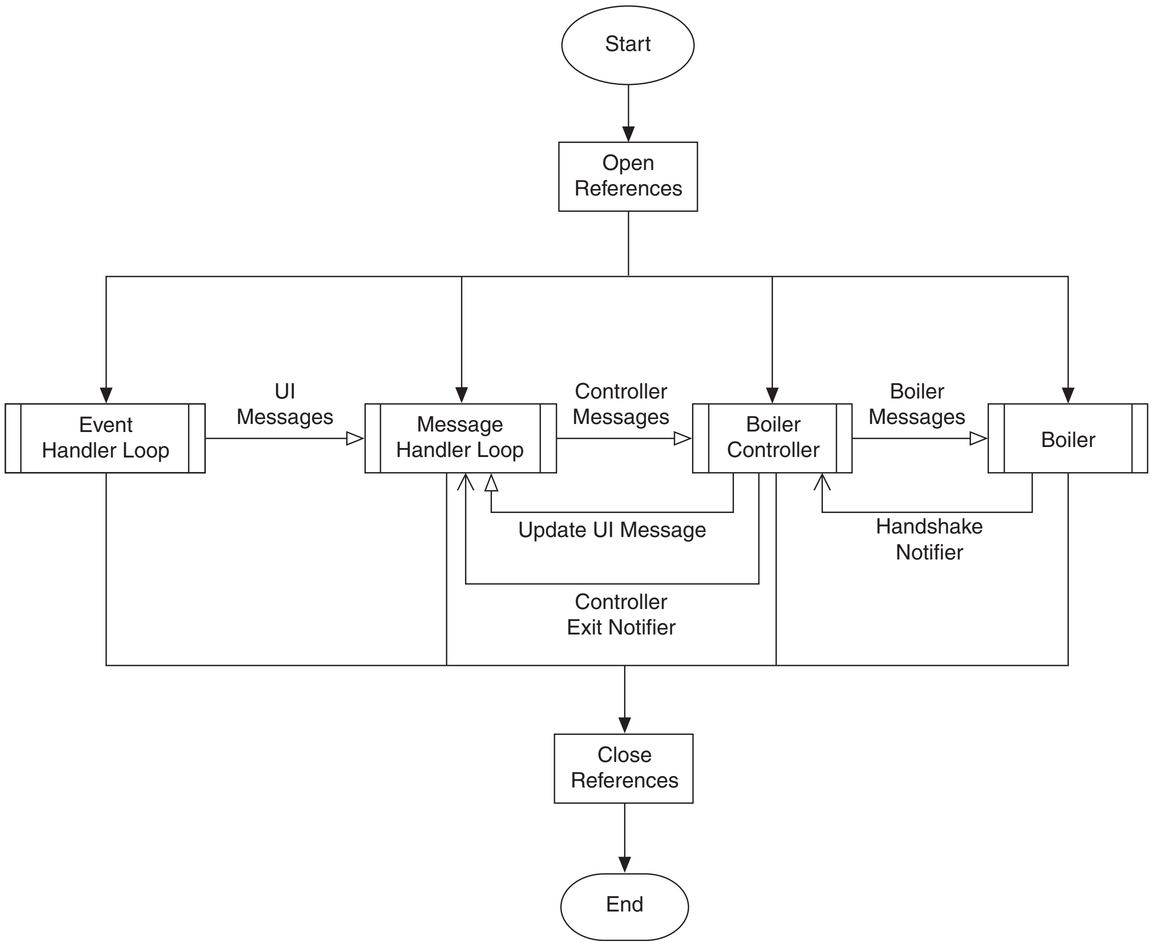
You will create two VIs to handle obtaining and releasing the notifiers that you will use in this application:

- `Create Notifiers.vi`—Obtains two notifiers: one for handshaking between the MHL and `Boiler Controller.vi` and the other for handshaking between `Boiler Controller.vi` and `Boiler.vi`. Bundle these notifiers into a type-defined cluster named `Notifiers.ctl`.
- `Close Notifiers.vi`—Releases the two handshaking notifiers to ensure that the memory for those references is deallocated.

A teammate will develop the code to send a command to the boiler and wait for it to be executed.

In Exercise 2-1, you created a project library that contains code that simulates the behavior of the boiler that the application will control. From this library, you add `Boiler.vi` as a fourth parallel process for the application.

Figure 3-13. Communication Among Event Handler Loop, Message Handler Loop, Boiler Controller, and Boiler



The integration of `Boiler.vi` requires the following additional changes to `Main.vi`:

- Add a message queue for the Boiler Loop. `Boiler Controller.vi` will use this queue to send instructions to `Boiler.vi`.
- Read configuration data for the boiler from `Boiler Init.ini`. To do this, you will create a subVI, `Read Configuration Data.vi`, that reads boiler constants from the INI file.

Due to the amount of start up code that must occur before the parallel loops begin, the team decides to create a subVI, `Boiler System Open.vi`, that will handle the following functionality:

- Create the Stop user event (`Create User Event - Stop.vi`)
- Create the message queues (`Create All Message Queues.vi`)
- Read boiler configuration data (`Read Configuration Data.vi`)
- Create the notifiers for handshaking (`Create Notifiers.vi`)

Implementation

Create Notifiers for Handshaking

Obtain the notifiers that will handle handshaking among the Message Handling Loop of `Main.vi`, `Boiler Controller.vi` and the boiler to ensure that the MHL does not shutdown until `Boiler Controller.vi` shuts down, which does not happen until the boiler shuts down.

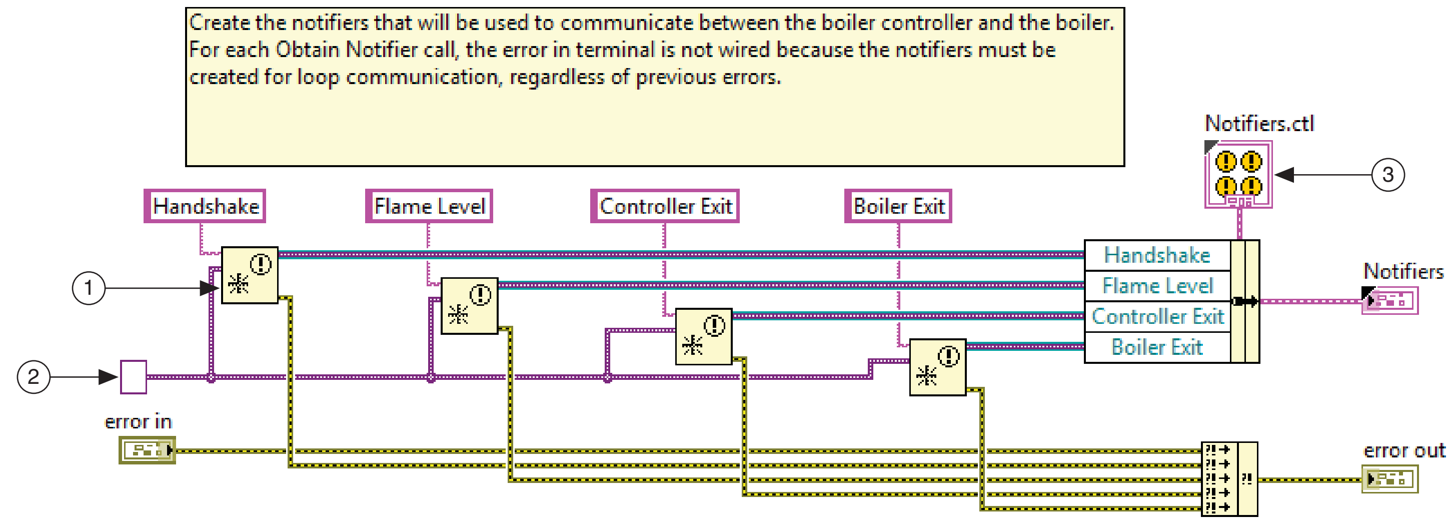
1. Create a VI that obtains the notifiers.

   ☐ In the **Support VIs** directory of the Boiler Controller Project Explorer window, create a virtual folder named `Notifiers`.

   ☐ Create a new VI in the **Notifiers** folder and save it as `Create Notifiers.vi` in `<Exercises>\LabVIEW Core 3\Course Project\support\Notifiers`.

☐ Create the block diagram as shown in Figure 3-14.

Figure 3-14. Create Notifiers Block Diagram


Create the notifiers that will be used to communicate between the boiler controller and the boiler. For each Obtain Notifier call, the error in terminal is not wired because the notifiers must be created for loop communication, regardless of previous errors.
Notifiers.ctl
Handshake
Flame Level
Controller Exit
Boiler Exit
Notifiers
error in
error out
1
2
3

1 Obtain Notifier function

2 Variant constant

3 Notifiers Type Definition

☐ Place a **Cluster Constant** on the block diagram.

☐ Right-click the **notifier out** output of one of the Obtain Notifier functions and select **Create»Constant**.

☐ Create three copies of the constant.

☐ Move all four notifier constants into the cluster constant.

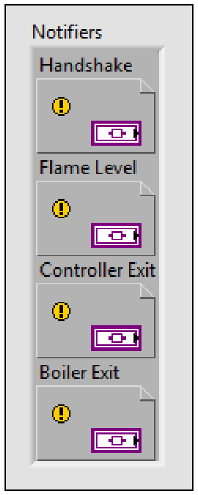
☐ Right-click the cluster constant border and select **AutoSizing»Arrange Vertically**.

☐ Right-click the cluster constant and select Make Type Def.

☐ Right-click the type def control and select Open Type Def.

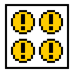
☐ Modify the type def control as shown in Figure 3-15 and save it as `Notifiers.ctl` in the `<Exercises>\LabVIEW Core 3\Course Project\support\Notifiers` directory.

Figure 3-15. Notifiers Type Def Control



☐ Create an icon for the Notifiers type def control as shown in Figure 3-16.

Figure 3-16. Notifiers Type Def Icon



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

☐ Move `Notifiers.ctl` into the **Notifiers** directory of the **Project Explorer** window.

2. Complete the front panel of the Create Notifiers VI as shown in Figure 3-17.

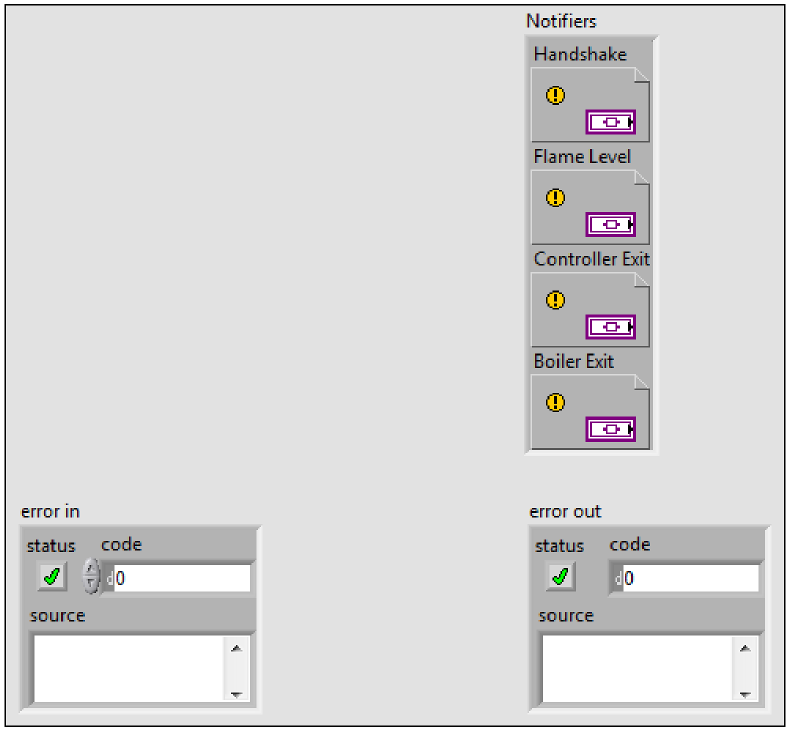


Figure 3-17. Create Notifiers Front Panel

3. Create the an icon and connector pane for `Create Notifiers.vi` as shown in Figure 3-18.

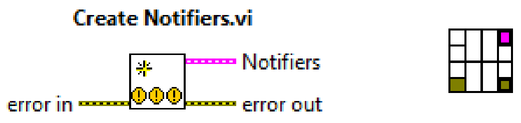


Figure 3-18. Create Notifiers Icon

1 Filter glyphs by keywords, `create` and `notifier`, to find appropriate glyphs for the icon.

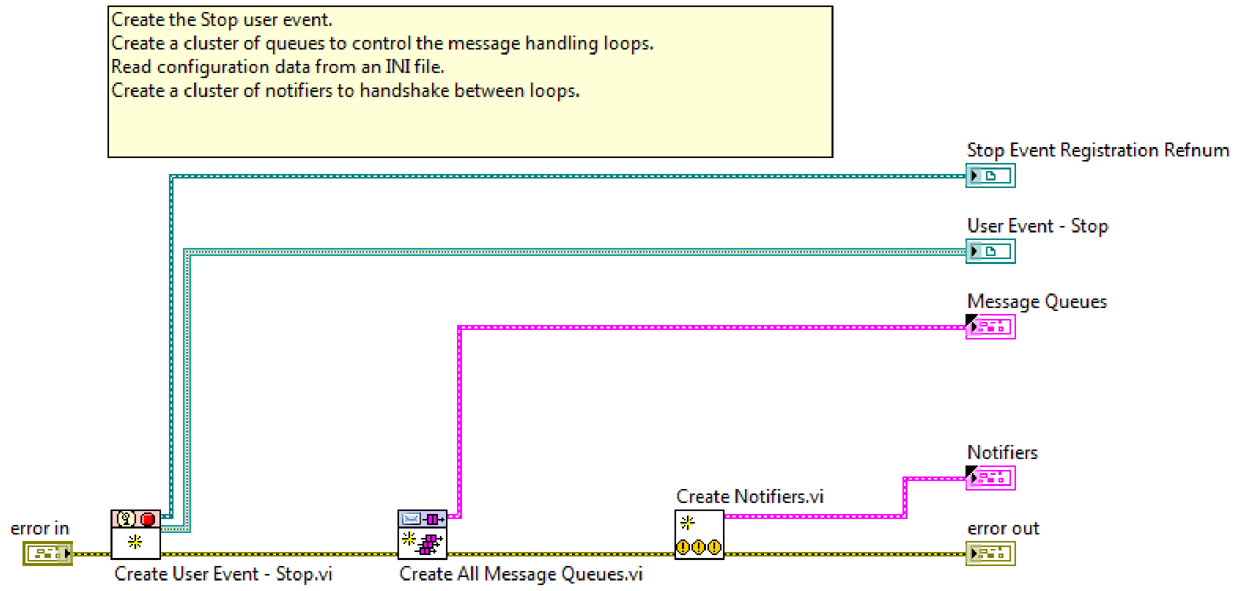
4. Save and close the VI.

5. Create a VI to call the setup VIs.

☐ Create a new VI in the **Support VIs** folder in the Boiler Controller **Project Explorer** window. Save the VI as `Boiler System Open.vi` in the `<Exercises>\LabVIEW Core 3\Course Project\support` directory.

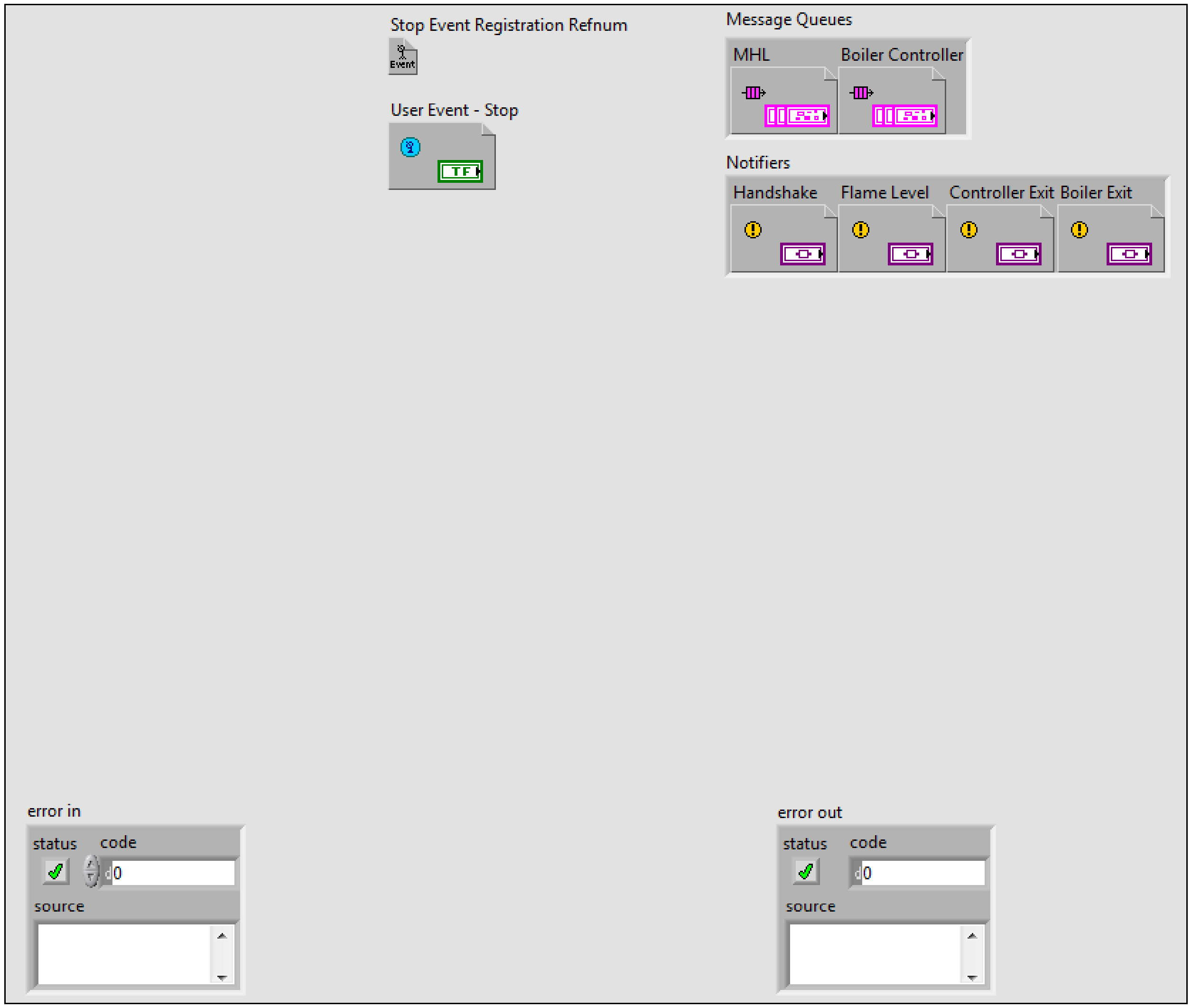
☐ Create the block diagram as shown in Figure 3-19.

Figure 3-19. Boiler System Open Block Diagram


Create the Stop user event.
Create a cluster of queues to control the message handling loops.
Read configuration data from an INI file.
Create a cluster of notifiers to handshake between loops.
Stop Event Registration Refnum
User Event - Stop
Message Queues
Notifiers
Create Notifiers.vi
error in
error out
Create User Event - Stop.vi
Create All Message Queues.vi

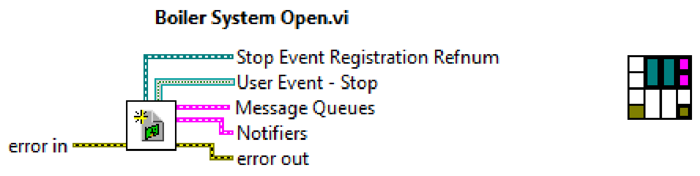
Arrange the Boiler System Open front panel as shown in Figure 3-20.

Figure 3-20. Boiler System Open Front Panel



☐ Create an icon and connector pane for the Boiler System Open VI as shown in Figure 3-21.

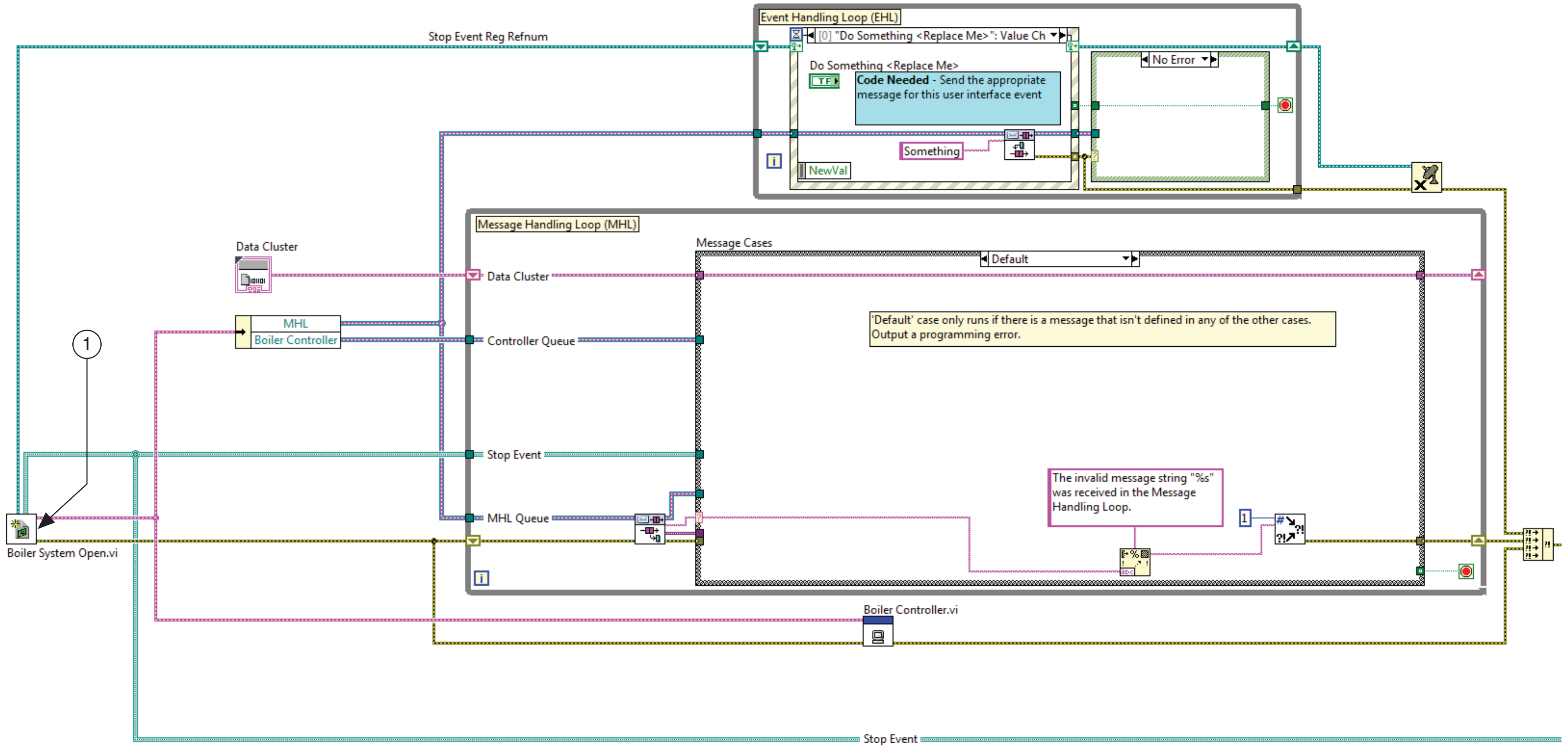
Figure 3-21. Boiler System Open Icon and Connector Pane



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

6. Call the Boiler System Open VI from the Main VI instead of calling each function separately.

Figure 3-22. Main VI Calling Boiler System Open


Stop Event Reg Refnum
Event Handling Loop (EHL)
[0] "Do Something <Replace Me>": Value Ch
Do Something <Replace Me>
Code Needed - Send the appropriate message for this user interface event
Something
NewVal
No Error
Message Handling Loop (MHL)
Message Cases
Default
Data Cluster
Data Cluster
MHL
Boiler Controller
Controller Queue
'Default' case only runs if there is a message that isn't defined in any of the other cases. Output a programming error.
1
Stop Event
MHL Queue
The invalid message string "%s" was received in the Message Handling Loop.
Boiler System Open.vi
Boiler Controller.vi
Stop Event

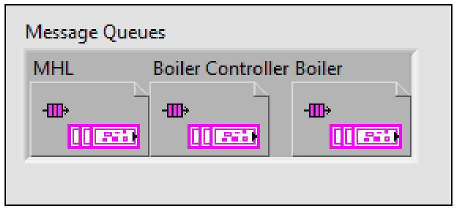
1 **Boiler System Open VI**—Replace `Create User Event - Stop.vi` and `Create All Message Queues.vi` with this VI.

Integrate Boiler.lvlib

This library contains `Boiler.vi`, the top-level VI responsible for simulating the behavior of the boiler, as well as the subVIs and type definitions required for `Boiler.vi` to run. `Boiler.vi` will serve as a fourth parallel process in the application (Event Handling Loop, Message Handling Loop, `Boiler Controller.vi`, and `Boiler.vi`).

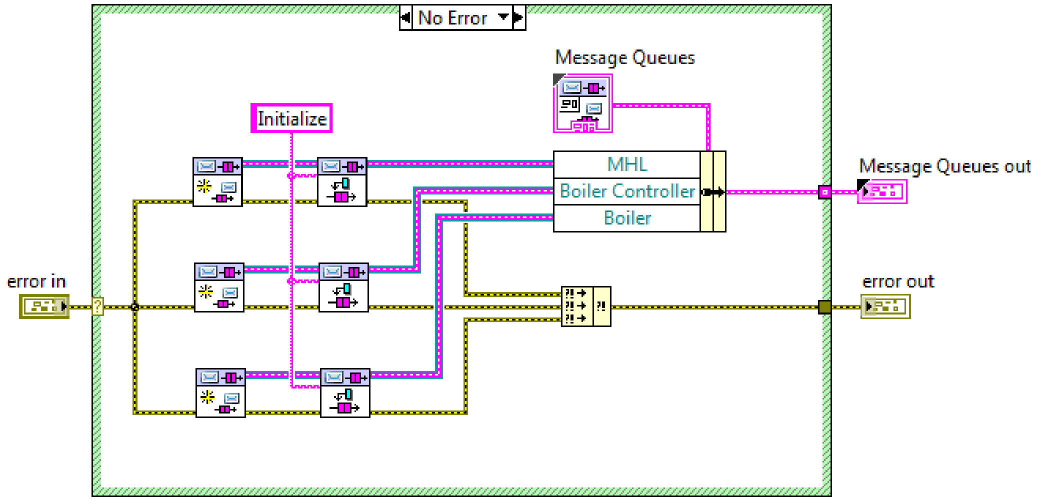
1. Modify `All Message Queues.ctl` to include a Boiler queue as shown in Figure 3-23.

Figure 3-23. All Message Queues Control with Boiler Queue



2. Modify `Create All Message Queues.vi` as shown in Figure 3-24.

Figure 3-24. Create All Message Queues Block Diagram

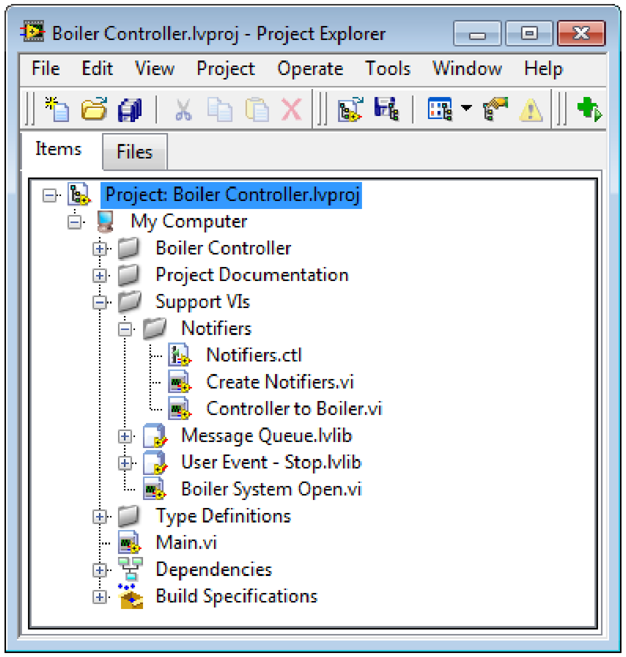


3. Add `Controller to Boiler.vi` to the project.

   ☐ Move a copy of `Controller to Boiler.vi` from `<Exercises>\LabVIEW Core 3\External\Support VIs\Notifiers` to `<Exercises>\LabVIEW Core 3\Course Project\support\Notifiers`.

   ☐ Add `Controller to Boiler.vi` to the Boiler Controller project in the **Support VIs»Notifiers** folder as shown in Figure 3-25.

Figure 3-25. Boiler Controller Project with Controller to Boiler Notifier

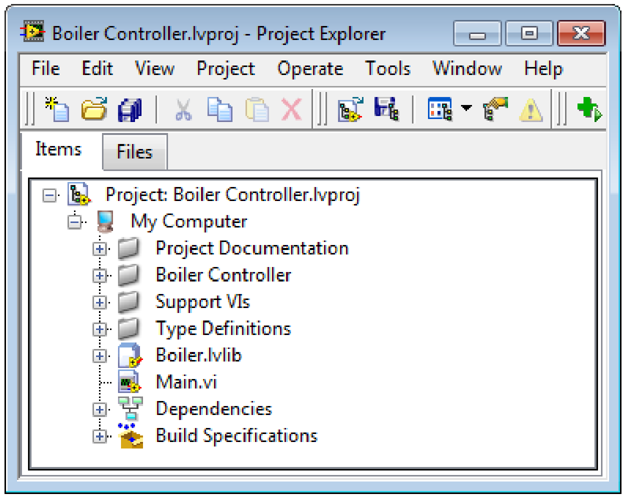

Boiler Controller.lvproj - Project Explorer
File Edit View Project Operate Tools Window Help
Items Files

```
Project: Boiler Controller.lvproj
  My Computer
    Boiler Controller
    Project Documentation
    Support VIs
      Notifiers
        Notifiers.ctl
        Create Notifiers.vi
        Controller to Boiler.vi
      Message Queue.lvlib
      User Event - Stop.lvlib
      Boiler System Open.vi
    Type Definitions
    Main.vi
    Dependencies
    Build Specifications
```

4. Add `Boiler.lvlib` to the project.

☐ Move the contents of the `<Exercises>\LabVIEW Core 3\External\Boiler` directory to the `<Exercises>\LabVIEW Core 3\Course Project\Boiler` directory.

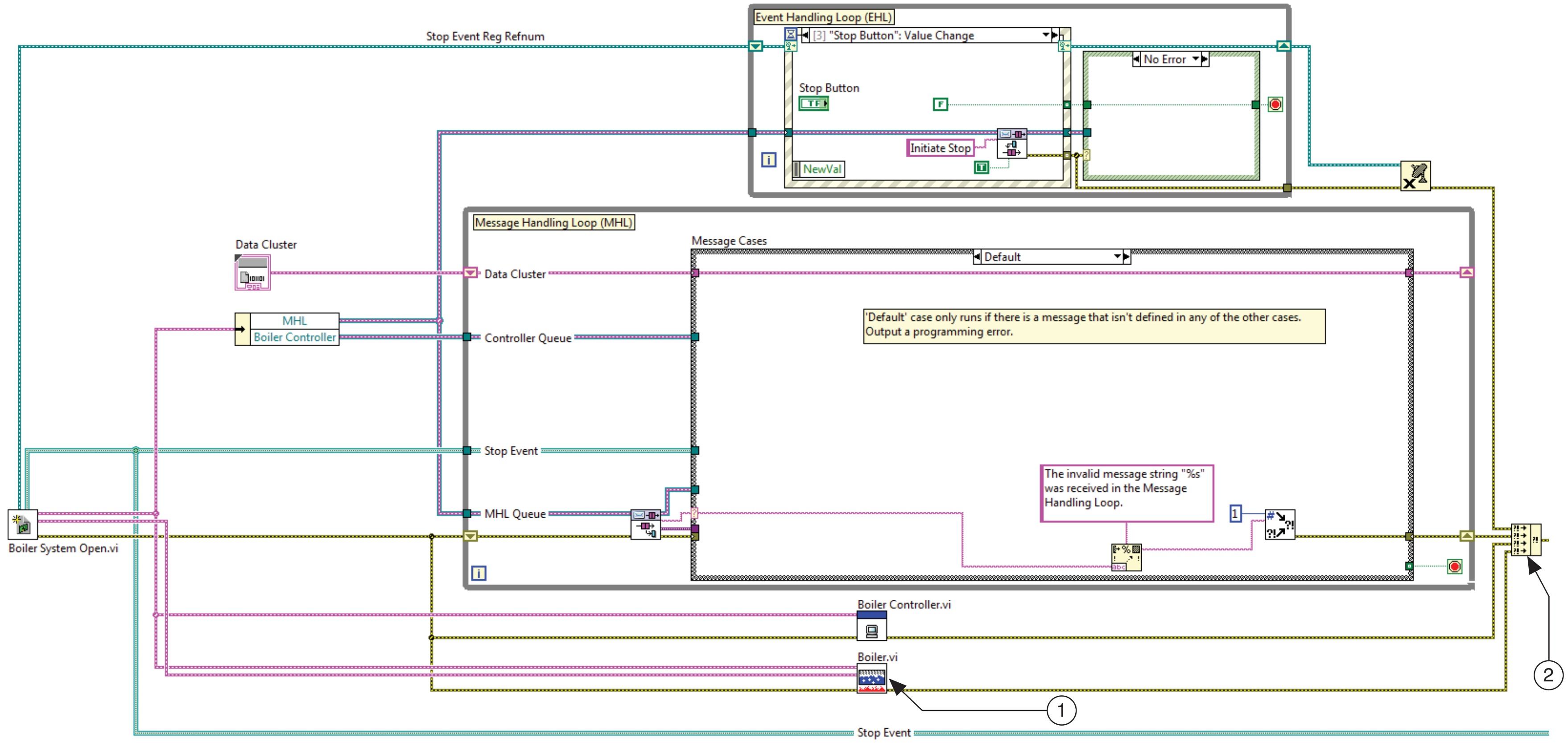
Figure 3-26. Boiler Controller Project with Boiler.lvlib


Boiler Controller.lvproj - Project Explorer
File Edit View Project Operate Tools Window Help
Items
Files

Project: Boiler Controller.lvproj
My Computer
Project Documentation
Boiler Controller
Support VIs
Type Definitions
Boiler.lvlib
Main.vi
Dependencies
Build Specifications

5. Place an instance of `Boiler.vi` from the `Boiler.lvlib` on the block diagram of `Main.vi` and modify `Main.vi` as shown in Figure 3-27.

Figure 3-27. Main VI with Boiler VI


Stop Event Reg Refnum
Event Handling Loop (EHL)
[3] "Stop Button": Value Change
Stop Button
NewVal
Initiate Stop
No Error
Message Handling Loop (MHL)
Data Cluster
Message Cases
Default
MHL
Boiler Controller
Controller Queue
'Default' case only runs if there is a message that isn't defined in any of the other cases. Output a programming error.
Stop Event
MHL Queue
The invalid message string "%s" was received in the Message Handling Loop.
Boiler System Open.vi
Boiler Controller.vi
Boiler.vi
Stop Event
1
2

1 Wire the Message Queues, Notifiers, and Error cluster wires as shown.

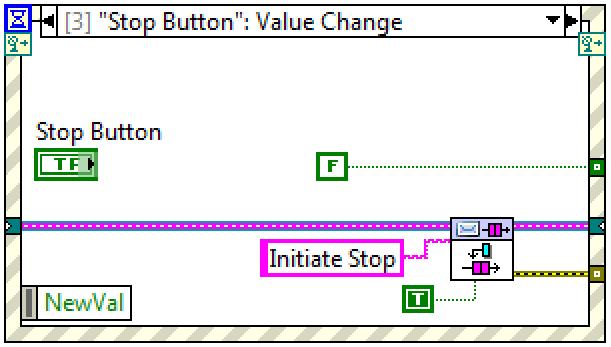
2 Expand the Merge Errors node to accommodate another input.

Synchronize the Shutdown Operation

This operation is initiated through a button-click and results in stopping all four loops.

1. Modify the Stop Button: Value Change event to enqueue the Initiate Stop message as shown in Figure 3-28.

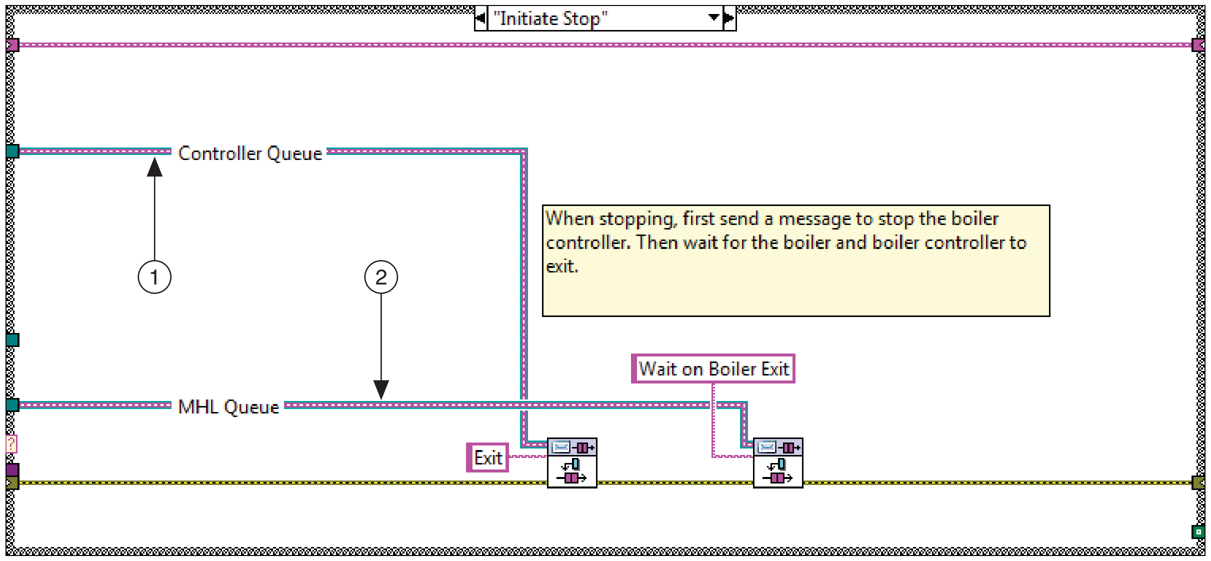
Figure 3-28. Stop Button: Value Change Event



The EHL sends a message to the MHL.

2. Create the Initiate Stop case of the Message Handling Loop as shown in Figure 3-29.

Figure 3-29. Message Handling Loop Initiate Stop Case



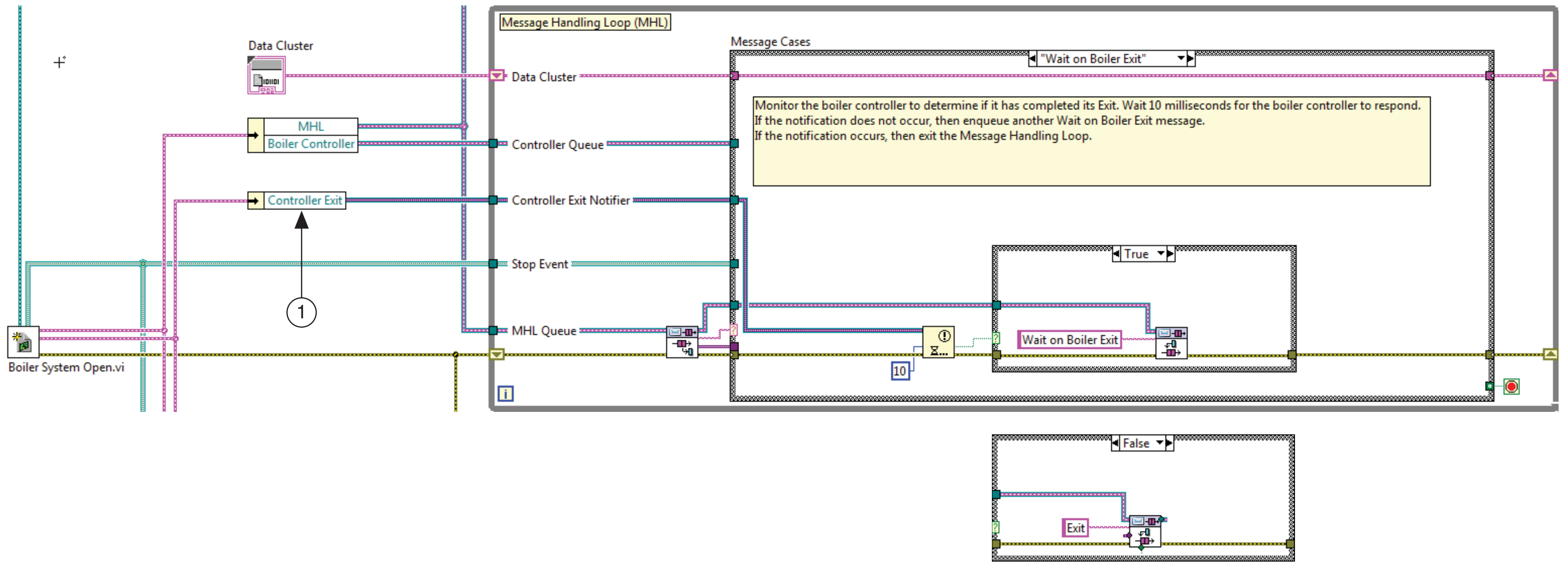
When stopping, first send a message to stop the boiler controller. Then wait for the boiler and boiler controller to exit.

1 The Message Handling Loop sends a message to the Boiler Controller VI.

2 The Message Handling Loop waits for Boiler Controller VI to finish executing the message before proceeding.

3. Create the Wait on Boiler Exit case of the Message Handling Loop as shown in Figure 3-30.

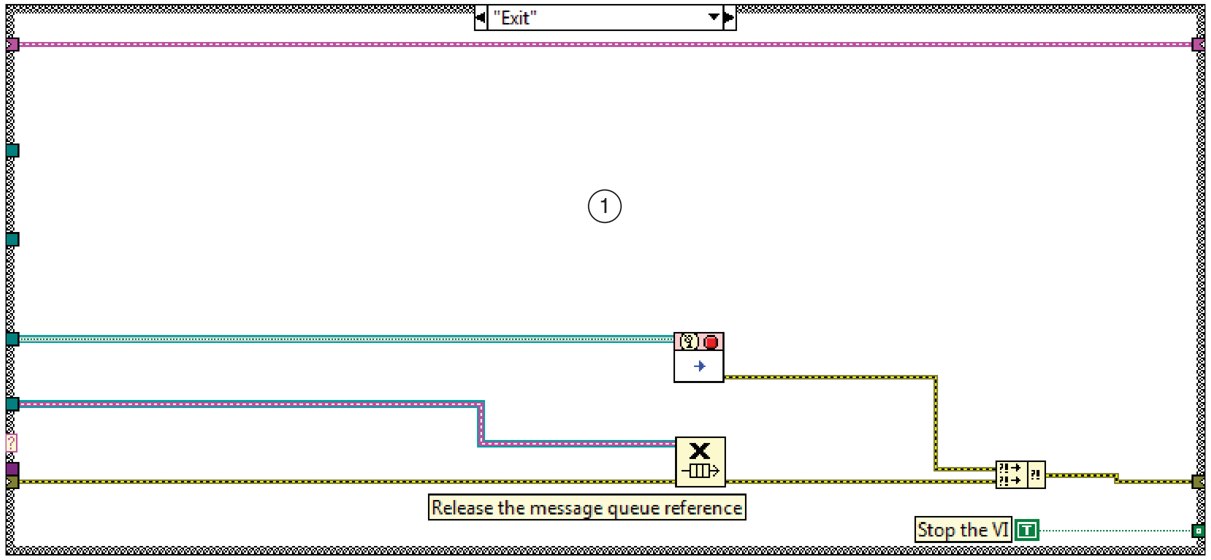
Figure 3-30. Message Handling Loop Wait on Boiler Exit Case True


Monitor the boiler controller to determine if it has completed its Exit. Wait 10 milliseconds for the boiler controller to respond.
If the notification does not occur, then enqueue another Wait on Boiler Exit message.
If the notification occurs, then exit the Message Handling Loop.

1 Unbundle the Controller Exit notifier and complete the wiring as shown.

4. Modify the Exit case of the Message Handling Loop as shown in Figure 3-31.

Figure 3-31. Exit Case

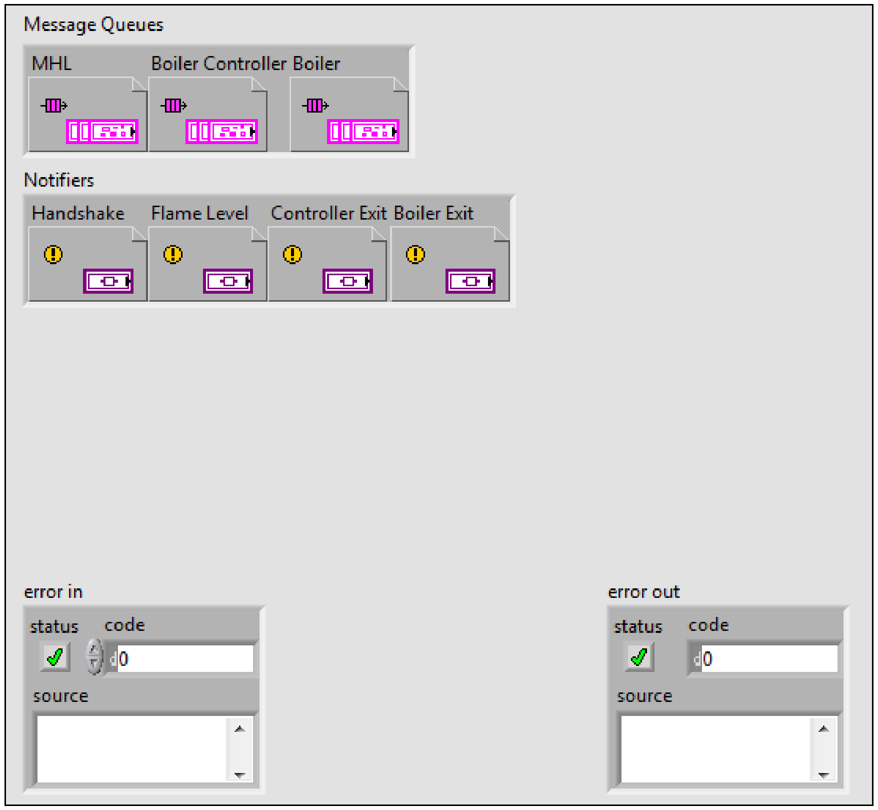


1 Delete the `Enqueue Message.vi` from this case because the Exit message is now being handled in the Wait on Boiler Exit case.

5. Update the Exit case of `Boiler Controller.vi`

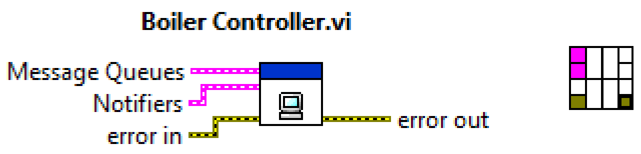
☐ Place a copy of the `Notifiers.ctl` type def control on the front panel of `Boiler Controller.vi` as shown in Figure 3-32

Figure 3-32. Boiler Controller Front Panel with Notifiers



6. Modify the Boiler Controller connector pane to include an input for notifiers as shown in Figure 3-33.

Figure 3-33. Boiler Controller Icon and Connector Pane with Notifiers Input

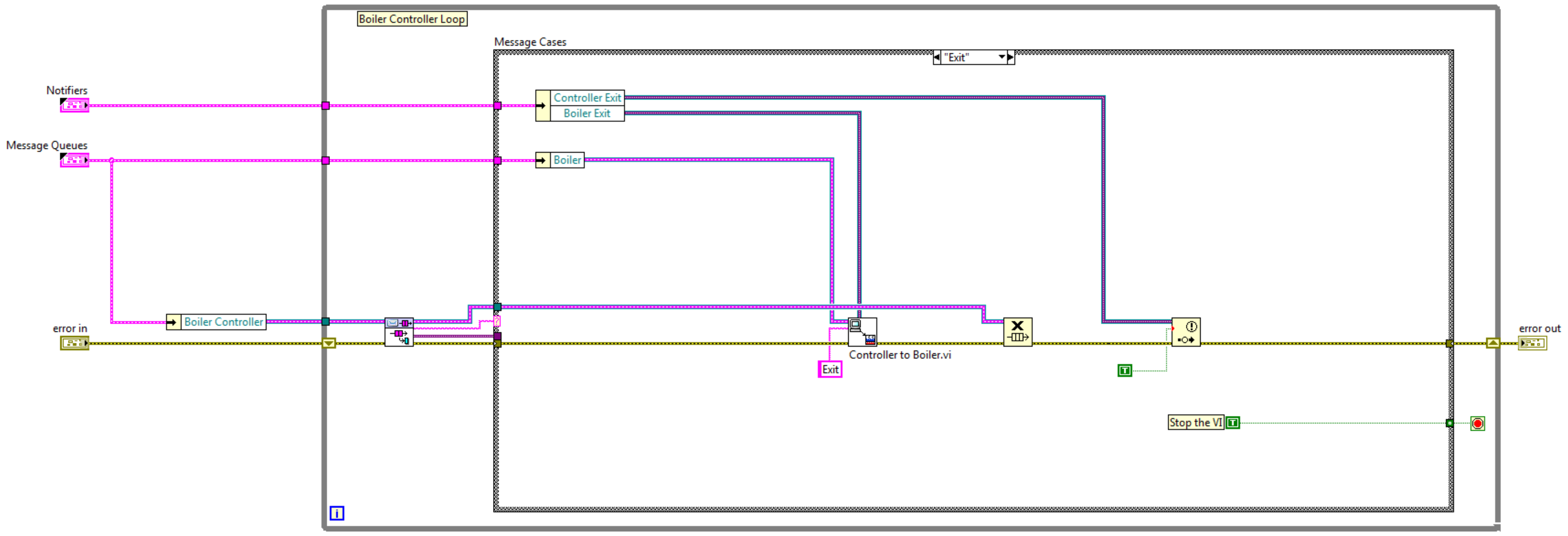


☐ On the block diagram of `Main.vi`, wire the **Notifiers** output of `Boiler System Open.vi` to the **Notifiers** input of `Boiler Controller.vi`.

7. Complete the `Boiler Controller.vi` Exit case as shown in Figure 3-34.

☐ `Boiler Controller.vi` sends a message to `Boiler.vi` and waits for `Boiler.vi` to finish executing that message before proceeding.

Figure 3-34. Boiler Controller VI Exit Case



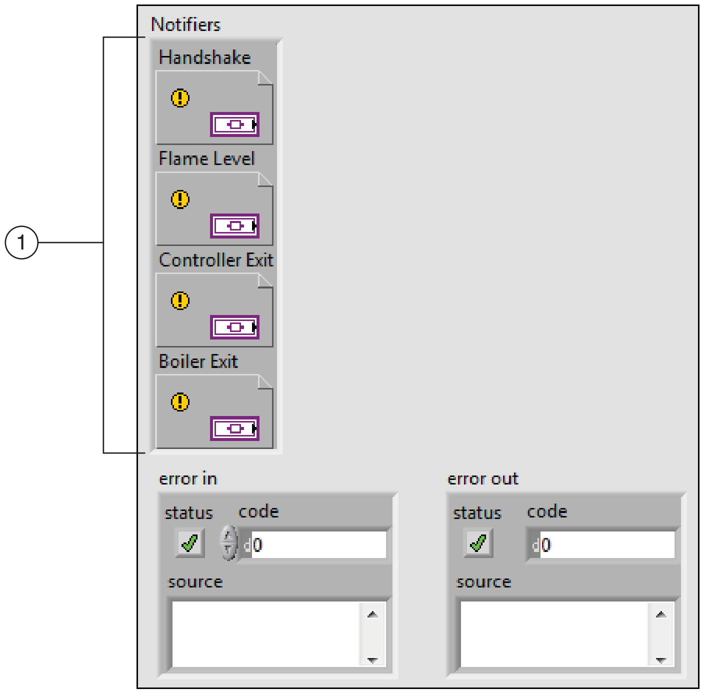
Release the Handshaking Notifiers when the Application Is Shutting Down

1. Close the notifiers that were allocated in `Create Notifiers.vi`.

   ☐ Create a new VI and save it as `Close Notifiers.vi` in the `<Exercises>\LabVIEW Core 3\Course Project\Support\Notifiers` directory and add it to the **Support VIs»Notifiers** directory of the Boiler Controller project.

   ☐ Create the `Close Notifiers.vi` front panel as shown in Figure 3-35.

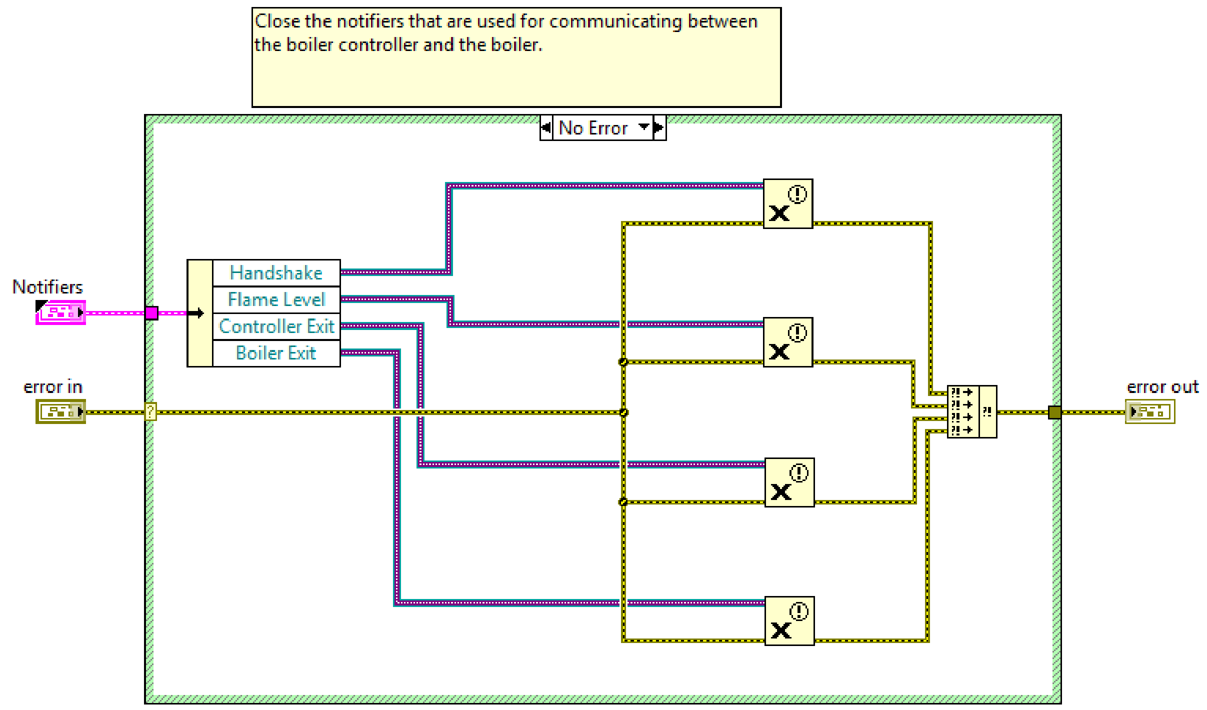
Figure 3-35. Close Notifiers VI Front Panel



1 Notifiers.ctl type def

☐ Modify the Close Notifiers VI block diagram as shown in Figure 3-36.

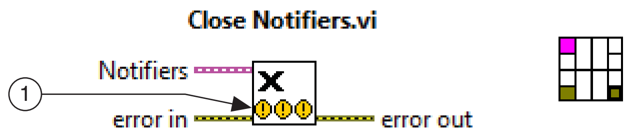
Figure 3-36. Close Notifiers VI Block Diagram


Close the notifiers that are used for communicating between the boiler controller and the boiler.
No Error
Notifiers
error in
Handshake
Flame Level
Controller Exit
Boiler Exit
error out

☐ Wire the error cluster wire directly through the Error case.

☐ Create the icon and connector pane as shown in Figure 3-37.

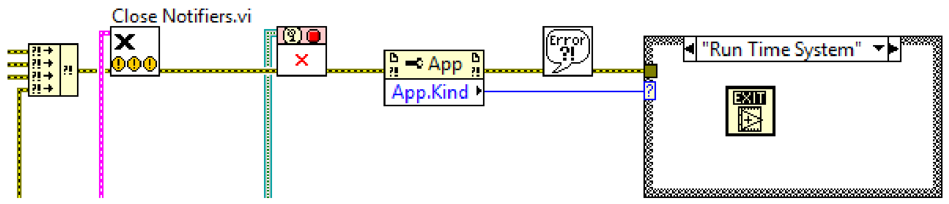
Figure 3-37. Close Notifiers Icon and Connector Pane



1 Filter glyphs by keyword, `notifier`, to find an appropriate glyph for the icon. Use the drawing tools to draw an X on the icon.

2. Modify `Main.vi` to wire Notifiers to `Boiler Controller.vi` and call `Close Notifiers.vi` at the end of execution as shown in Figure 3-38.

Figure 3-38. Main VI with Close Notifiers



Test the Application

Verify that the VI and the application build specification both work as expected before moving on to the next sprint.

1. Test the VI.

   - ☐ Run the VI and verify that the **Boiler Controller - Initialize** dialog box launches.
   - ☐ Click the **Do Something** and **Do Something Else** buttons and verify that the **Display** status string updates appropriately.
   - ☐ Click **Stop** and verify that all loops exit successfully.

2. Test the executable.

   a. Right-click the Main Application build specification and select **Build** from the shortcut menu to build the executable with the changes from this lesson.

   b. Click the **Explore** button when LabVIEW finishes the build.

   c. Double-click `Boiler Controller.exe` to run the application.

   - ☐ Verify that the **Boiler Controller - Initialize** dialog box launches and the **Do Something** and **Do Something Else** buttons update the **Display** status string appropriately.
   - ☐ Click **Stop** and verify all windows close.

End of Exercise 3-2

4

Customizing the User Interface

Topics

Exercise 4-1 Create a User Interface Prototype

Goal

Create a user interface prototype for an application.

Scenario

In the sprint planning meeting for this iteration, the product owner chose to implement the following user story:

- As a boiler operator, I want to control the boiler start-up process from the user interface and observe the current state of the boiler, so that I can readily identify if the boiler behaves correctly at each step.

After discussing this user story with the team, you have identified the following development tasks that must be completed:

1. Modify the front panel of `Main.vi` to include the anticipated controls and indicators.
2. Modify the Event Handling Loop (EHL) to do the following:
   - Handle value change events for the controls.
   - Send messages to the Message Handling Loop (MHL).
3. Modify the data cluster type definition for the MHL to store values for each indicator and include references for all controls.
4. Modify the MHL in the following ways:
   - Modify the Initialize case to initialize the updated user interface.
   - Modify the Update Display case to update the values of the new indicators.
   - Add a case to the MHL for each message from the EHL.
5. Update Boiler Controller VI to add a case for each message from the MHL.

Design

You need to modify the front panel of `Main.vi` to include the following controls and indicators:

Table 4-1. Controls and Indicators for Main VI

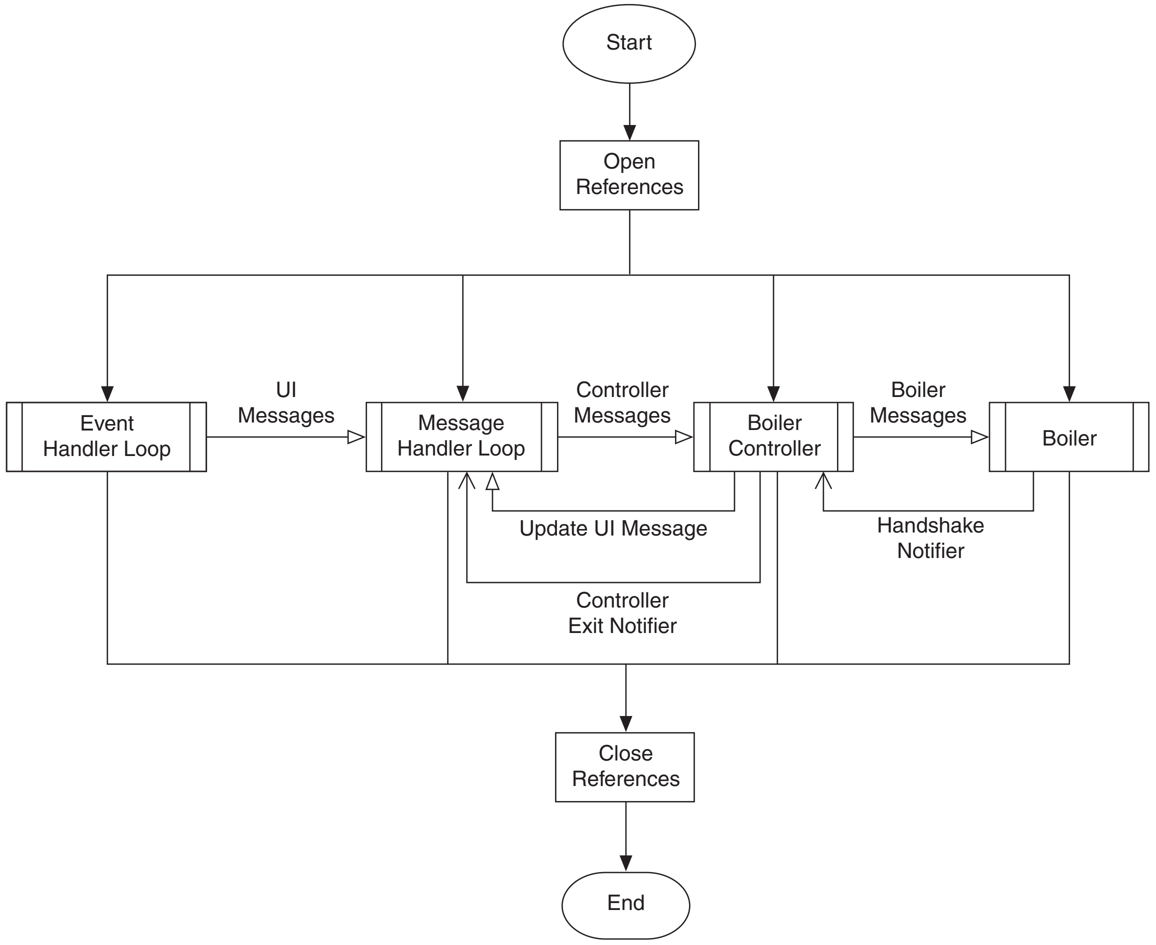
Label	Type	Control or Indicator?	Default Value (if applicable)
Reset	Silver Boolean Blank Button	Control	N/A
Start	Silver Boolean Play Button	Control	N/A
Light Pilot	Silver Boolean Blank Button	Control	N/A
Stop Boiler	Silver Boolean Stop Button	Control	N/A
Emergency Stop	Silver Boolean Process Stop Button	Control	N/A
Fuel Control Valve	Silver Horizontal Pointer Slide	Control	0
Status	Silver String Indicator	Indicator	Blank
Primary Fan	Silver Boolean LED	Indicator	Off
Pilot Gas Valve	Silver Boolean LED	Indicator	Off
Pilot	Silver Boolean LED	Indicator	Off
Forced Draft Fan	Silver Boolean LED	Indicator	Off

You also need to update the Event Handling Loop and the Message Handling Loop to implement the following functionality.

- When you change the value of a control, it triggers an event in the EHL.
- The EHL case for that value change sends a message to the MHL.
- The MHL case for that message sends a message to the boiler controller.
- Eventually, the boiler controller will send messages to the boiler. For now, use a one-button dialog in the boiler controller case to indicate that the message was received from the MHL.

Figure 4-1 illustrates the communication between the EHL, MHL, and boiler controller.

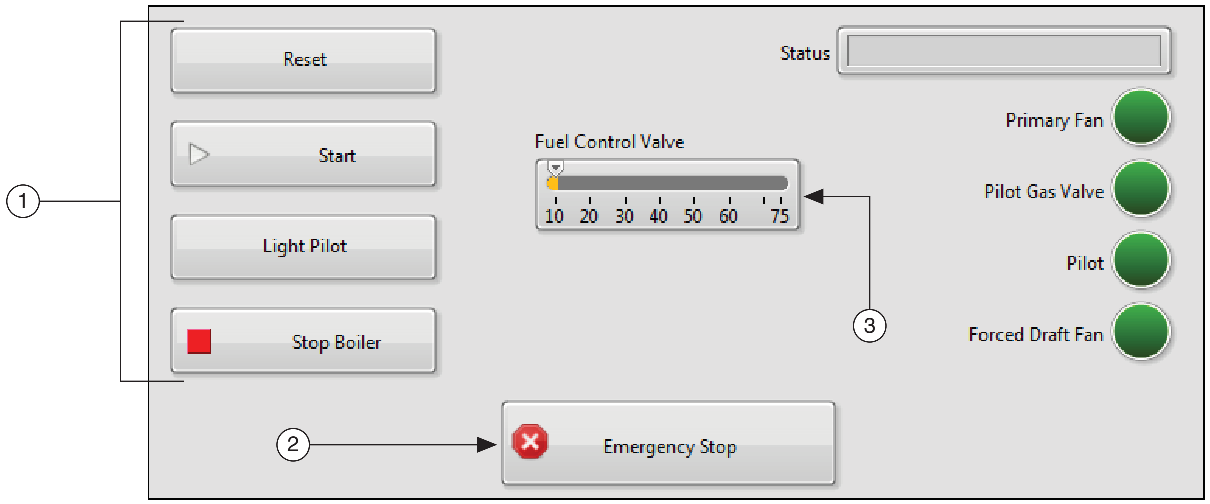
Figure 4-1. Communication Between the EHL, MHL, and Boiler Controller



Implementation

1. Open `Boiler Controller.lvproj` from the `<Exercises>\LabVIEW Core 3\Course Project` directory.
2. Open **Main.vi** from the **Project Explorer** window.
3. Modify the front panel of `Main.vi` as shown in Figure 4-2.

Figure 4-2. Front Panel of Main VI



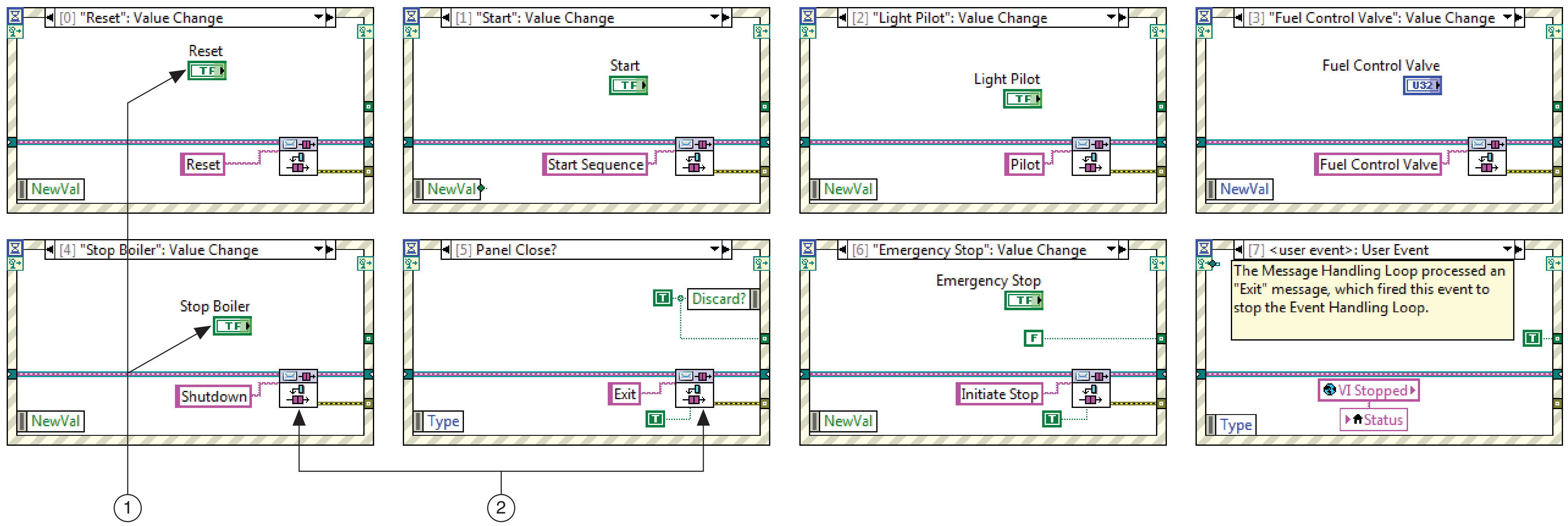
1	Make sure that the label for these buttons match the labels in Table 4-1. Event cases use the label of Boolean controls, not the Boolean text, to identify a control.
2	Use the Process Stop Button to create the Emergency Stop button.
3	Set the representation for the Fuel Control Valve Slide to U32

4. Modify the Event Handling Loop to handle value change events for the controls and send messages to the Message Handling Loop.

   ☐ Modify or add events for each control.

Tip You can right-click the Event structure and select **Duplicate Event Case** to add new cases similar to cases already in the Event structure.

Figure 4-3. New and Updated Events for the Event Handling Loop



The Message Handling Loop processed an "Exit" message, which fired this event to stop the Event Handling Loop.

1	Control terminals—Move the terminals of the controls you created in step 3 into the appropriate event case.
2	Enqueue Message VI—Update the Message input to send the appropriate message for each control.

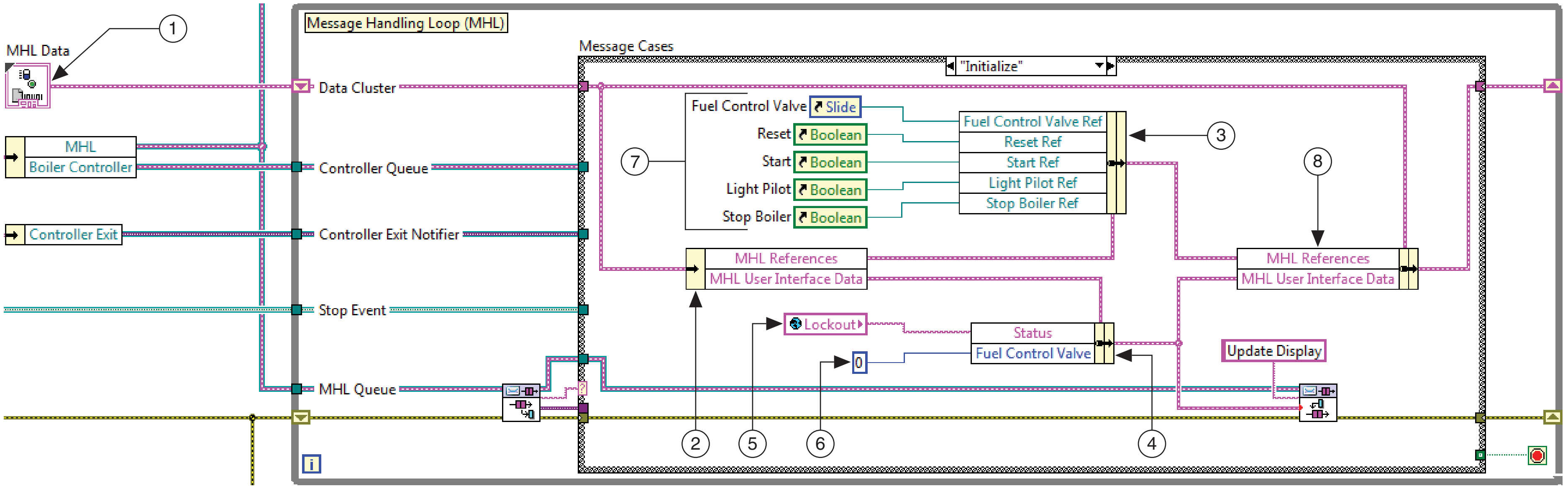
5. Add MHL type def controls to the project.

   ☐ In Windows Explorer, copy the contents of the `<Exercises>\LabVIEW Core 3\External\controls` directory into the `<Exercises>\LabVIEW Core 3\Course Project\controls` directory.

   ☐ In the **Project Explorer** window, right-click the **Type Definitions** and select **Add»File** and then navigate to `<Exercises>\LabVIEW Core 3\Course Project\controls` and select all the files to add to the project.

6. Modify the Initialize case of the Message Handling Loop as shown in Figure 4-4.

Figure 4-4. The Updated Initialize Case of the MHL



1 Replace the Data Cluster with the MHL Data Cluster.

2 **Unbundle By Name**—Wire the Data Cluster to the Unbundle By Name function and expand the function to display **MHL References** and **MHL User Interface Data** elements.

3 **Bundle By Name**—Wire the MHL References output to the **input cluster** terminal of the Bundle By Name function. Expand the function to show all five elements.

4 **Bundle By Name**—Wire the MHL User Interface Data output to the **input cluster** terminal of the Bundle By Name function. Expand the function to show the Status and Fuel Control Valve elements.

5 **Global Variable**—Place a copy of `Boiler System Globals.vi` on the block diagram and select **Lockout**. Right-click the **Lockout** global variable and select **Change To Read** from the shortcut menu.

6 Create a constant for the Fuel Control Valve element.

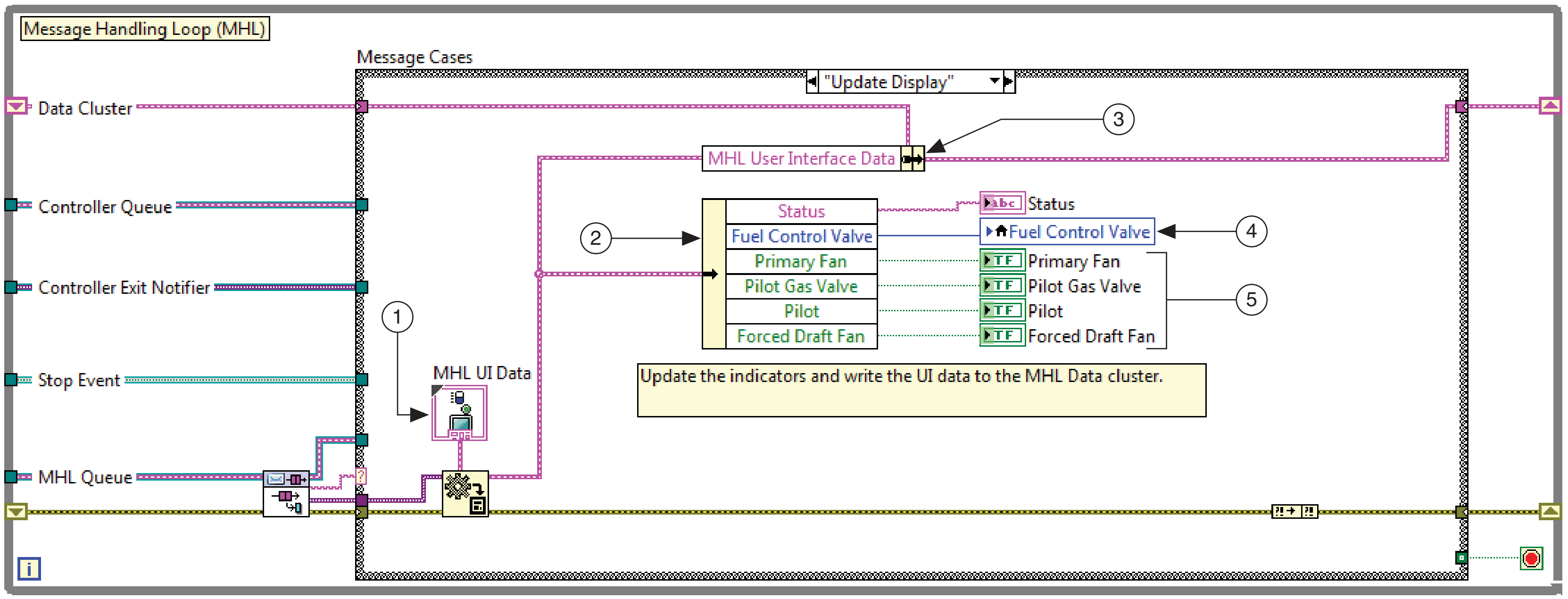
7 Create control references—Select the Reset, Start, Light Pilot, Stop Boiler, and Fuel Control Valve controls on the front panel of Main VI. Right-click one of the controls and select **Create»Reference** from the shortcut menu. Wire these references to the Bundle By Name function.

8 **Bundle By Name**—Expand to display **MHL References** and **MHL User Interface Data** elements.

7. Delete the Initialize Data and Initialize Panel cases of the MHL Case structure.

8. Modify the Update Display case of MHL to update the values of the new indicators, as shown in Figure 4-5.

Figure 4-5. Modified Update Display Case of the MHL



1 Wire an MHL UI Data control to the Variant To Data function—Right-click the empty string constant and select **Replace»Select a VI** from the menu. Navigate to the `Course Project\controls` directory and select `MHL UI Data.ctl`.

2 **Unbundle By Name**—Wire the data output of the Variant To Data function to the Unbundle By Name function. Expand the Unbundle By Name function to display all six terminals.

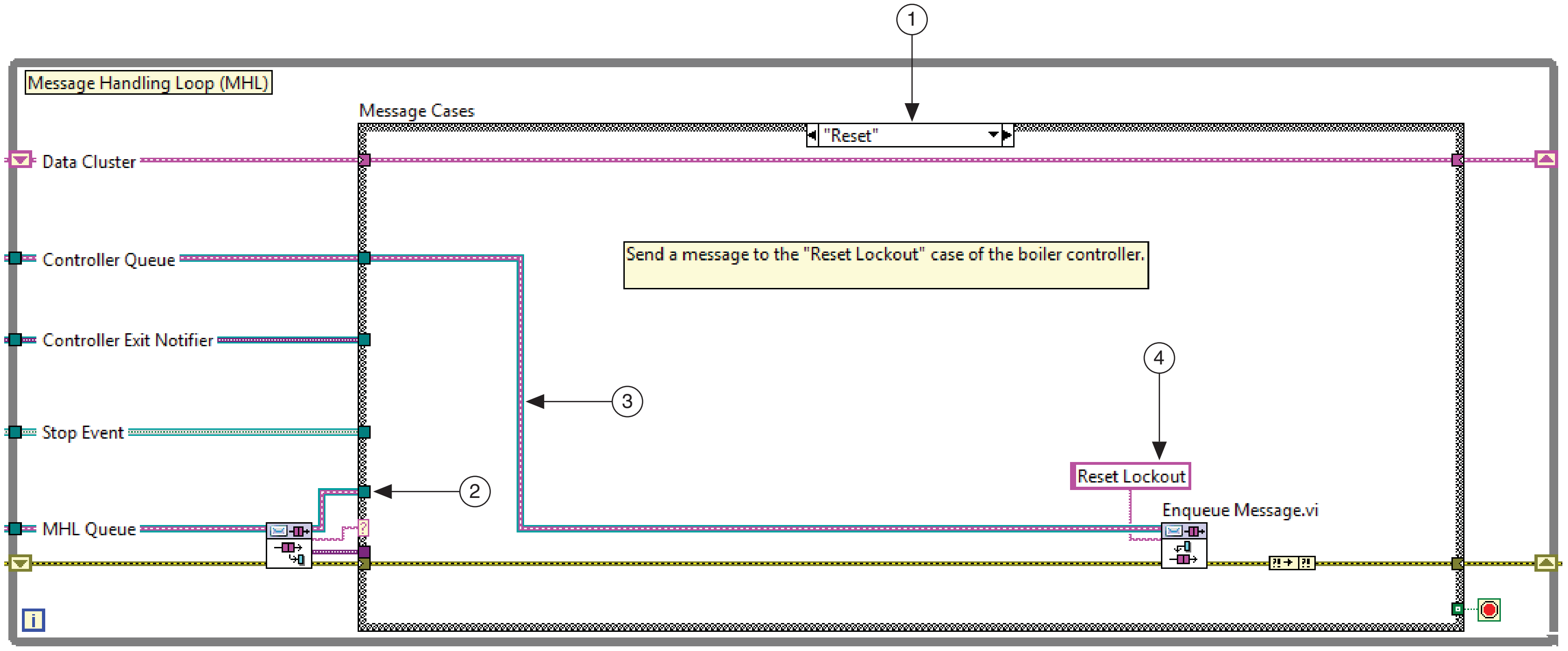
3 **Bundle By Name**—Wire the Data Cluster wire to the input cluster terminal and wire the data output of the Variant To Data function to the MHL User Interface Data terminal. This code bundles UI data from the Data Cluster wire with the MHL UI Data cluster. Because the boiler controller eventually calls the Update Display case, this case needs to be able to read the message data and write it to the shift register.

4 Local Variable—Right-click the Fuel Control Valve control in the Fuel Control Valve event of the EHL and select **Create»Local Variable** from the shortcut menu. Drag the Fuel Control Valve local variable to the Update Display case and wire it to the Unbundle By Name function.

5 Drag the terminals to the indicators you created in step 3 to the Update Display case and wire them to the Unbundle By Name function.

9. Update the Something case of the MHL Case structure to be the Reset case, as shown in Figure 4-6.

Figure 4-6. Modifying the Something Case to Create the Reset Case



1 Change the name of the Something case—Select the Something case and double-click the title to select it. Change the name of the case to `Reset`.

2 Delete the MHL Queue wire.

3 Enqueue message to the Boiler Controller—Wire the Controller Queue data to the **Message Queue** input of the Enqueue Message VI.

4 Change the constant wired to the **Message** input to `Reset Lockout`.

10. Duplicate the Reset case to create the Start Sequence, Pilot, Fuel Control Valve, and Shutdown cases as shown in Figures 4-7 through 4-10.

Note Make sure that the Case text matches the string value used in the EHL. Otherwise, the program enters the Default message case and produces an error.

Figure 4-7. The Start Sequence Case

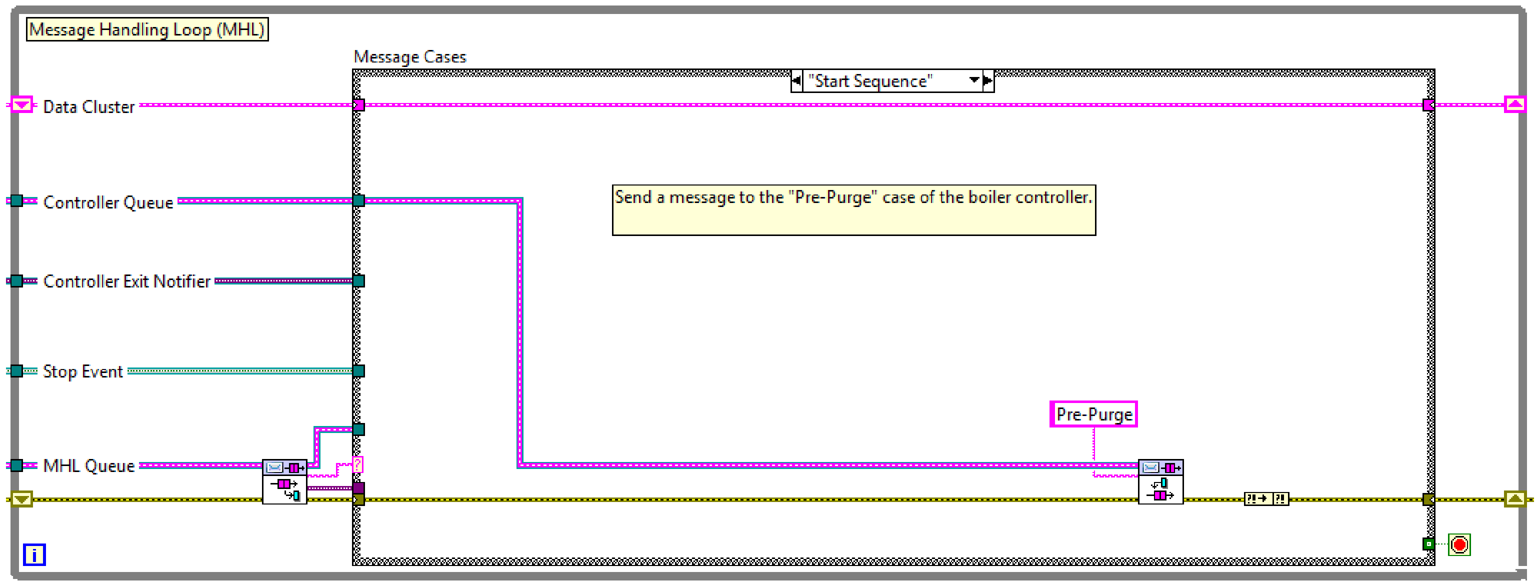


Figure 4-8. The Pilot Case

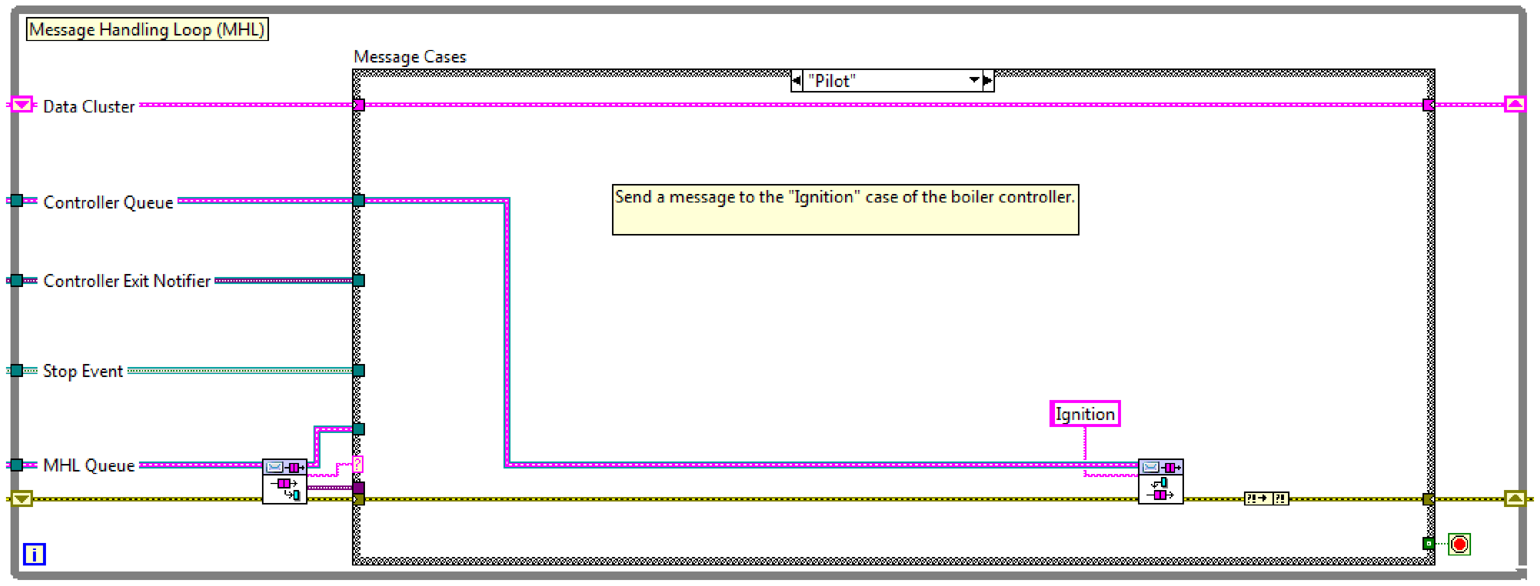
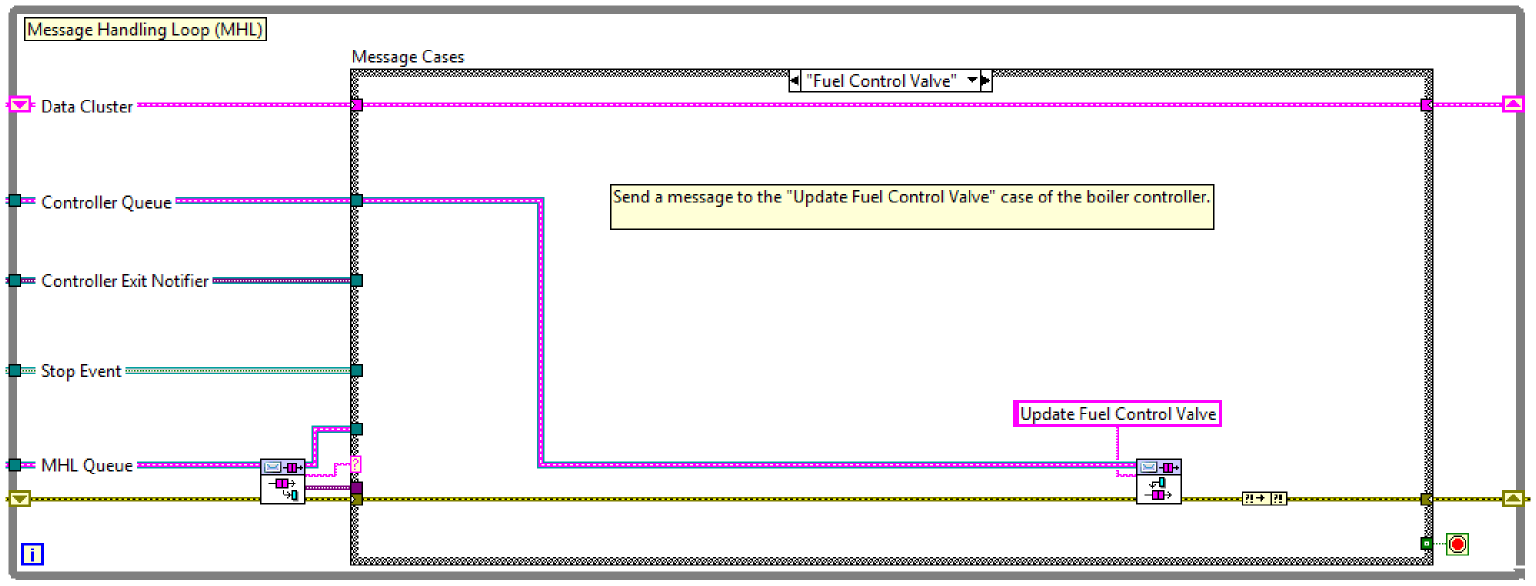
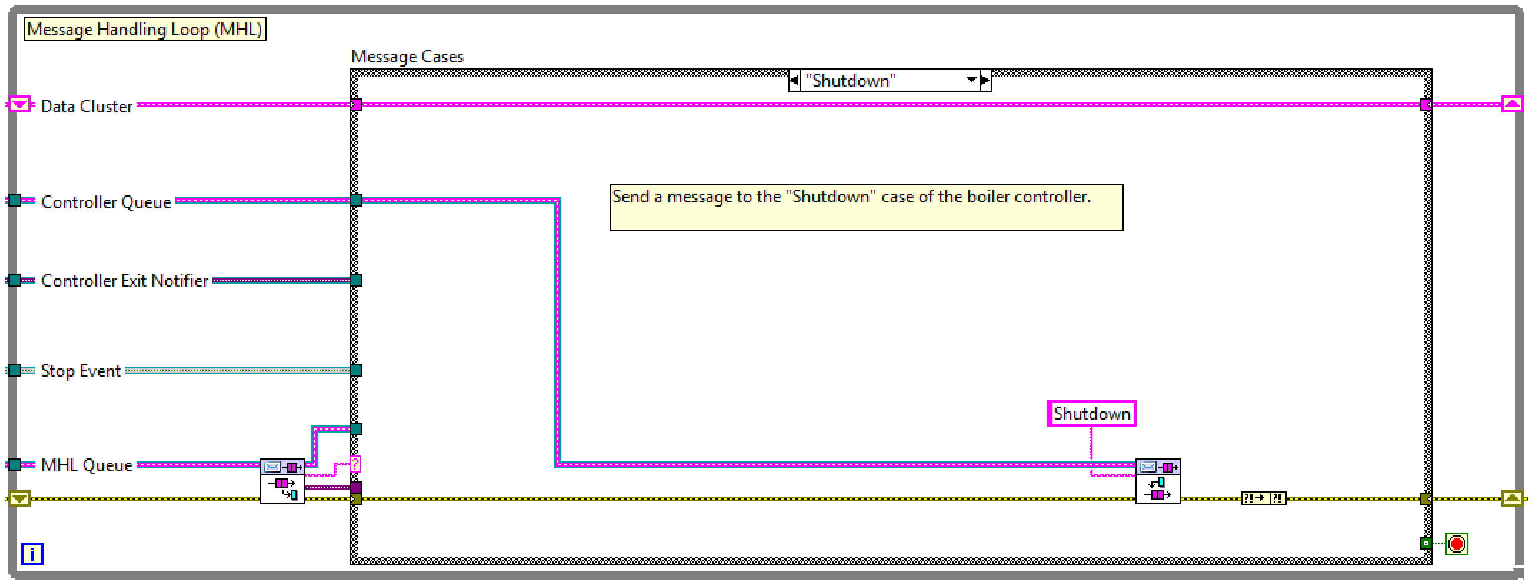


Figure 4-9. The Fuel Control Valve Case



Send a message to the "Update Fuel Control Valve" case of the boiler controller.

Figure 4-10. The Shutdown Case



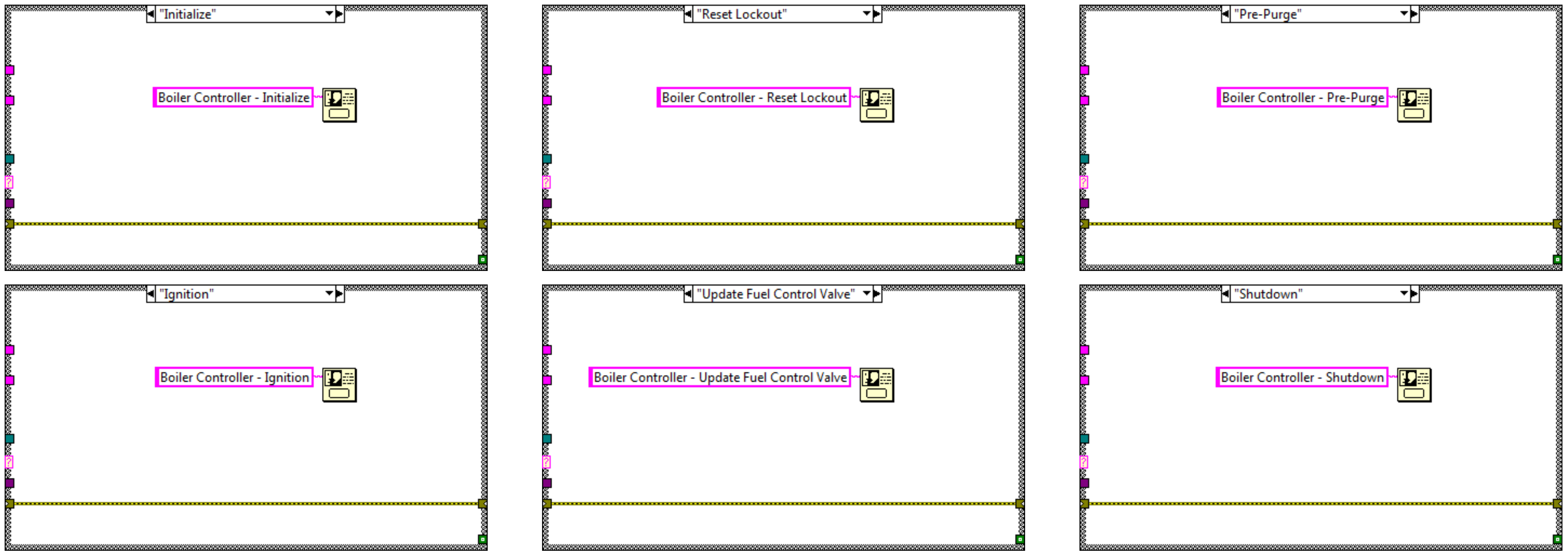
11. Delete the Something Else and the Copy This Frame cases. The MHL Case structure should now have a total of 11 cases.

12. Save Main VI.

13. Open `Boiler Controller.vi` from the **Project Explorer** window.

14. Duplicate the Initialize case in the Boiler Controller VI to add a case for each message from the MHL in Main VI. For each case, the Boiler Controller VI launches a 1-button dialog as shown in Figure 4-11. You add more functionality later in the course.

Figure 4-11. Cases that Handle Each Message from the MHL in Main VI


"Initialize"
Boiler Controller - Initialize
"Reset Lockout"
Boiler Controller - Reset Lockout
"Pre-Purge"
Boiler Controller - Pre-Purge
"Ignition"
Boiler Controller - Ignition
"Update Fuel Control Valve"
Boiler Controller - Update Fuel Control Valve
"Shutdown"
Boiler Controller - Shutdown

Note Make sure that the Case text matches the string value used in the MHL of Main VI.

Test the Application

Verify that previous functionality still works and that user interface events result in execution of the appropriate case in the boiler controller.

1. Test the VI.

   - ☐ Run Main VI.
   - ☐ Verify that the **Boiler Controller - Initialize** dialog box launches on start-up.
   - ☐ Click the **Reset**, **Start**, **Light Pilot**, and **Stop Boiler** buttons and verify that the boiler controller displays the appropriate dialog boxes.
   - ☐ Change the value of the **Fuel Control Valve** control verify that the boiler controller displays the appropriate dialog box.
   - ☐ Click the **Emergency Stop** button and verify that all VIs stop.

2. Update the build specification and test the new executable.

   - ☐ Right-click the Main Application build specification and select **Build** from the shortcut menu. LabVIEW creates a new build incorporating the changes you made.
   - ☐ Click the **Explore** button when LabVIEW finishes the build.
   - ☐ Double-click `Boiler Controller.exe` to run the application.
   - ☐ Verify the behavior you tested in step 1.
   - ☐ Click the **Emergency Stop** button and verify that all VIs stop and the application exits.

End of Exercise 4-1

Exercise 4-2 Create User Documentation

Goal

Document the user interface and create application documentation.

Scenario

After presenting the user interface prototype, the customer representative re-emphasized the desire for documentation in the user interface to improve usability. They also made several cosmetic suggestions for the user interface. The product owner documented the user interface feedback for implementation in a future sprint iteration.

As a result of the customer's feedback, the product owner prioritized implementing the following user story:

- As a boiler operator, I want the system to include sufficient user documentation, so that new users can easily learn to operate the boiler controller.

After discussing this user story with the team, you identify the following development tasks that the team must complete.

- Modify the VI description for `Main.vi`.
- Document each front panel control and indicator.
- Create printable documentation for the application.
- Link to the printable documentation in the VI description.

Design

Include a detailed overview of the functionality of the application and instructions for how to use the Main VI.

Document each control so that the user understands the results of modifying each control. Document limitations or appropriate values for each control, if appropriate.

Document each indicator so that the user understands the information that the indicator conveys.

Using LabVIEW, generate HTML or RTF documents that include the VI, control, and indicator descriptions. You can manually edit the documents later, if necessary. Link the **VI Properties** dialog box to this documentation.

Implementation

1. Modify the VI description for `Main.vi` to include a detailed overview of the VI functionality.

   ☐ In the **Project Explorer** window, open **Main.vi**.

   ☐ Select **File»VI Properties** from the menu.

   ☐ Select **Documentation** from the **Category** pull-down menu.

   ☐ Delete the current description and enter information about the functionality of the application. Good documentation includes the following information.

   – An overview of the VI

   – Instructions on how to use the VI, such as how to start the boiler and how to shut down the system in case of emergency.

   – Use the following example to document this VI.

   ```
   The boiler controller allows a user to start and shut down a boiler.

   Use the Reset, Start, and Light Pilot controls to sequentially start the boiler.

   Use the Fuel Control Valve to change the flow of fuel into the boiler.

   Use the Stop Boiler button to shut down the boiler.

   To shut down the system or in the event of an emergency, use the Emergency Stop button to close the
   application completely.
   ```

   ☐ Click the **OK** button in the **VI Properties** dialog box when you finish documenting the VI.

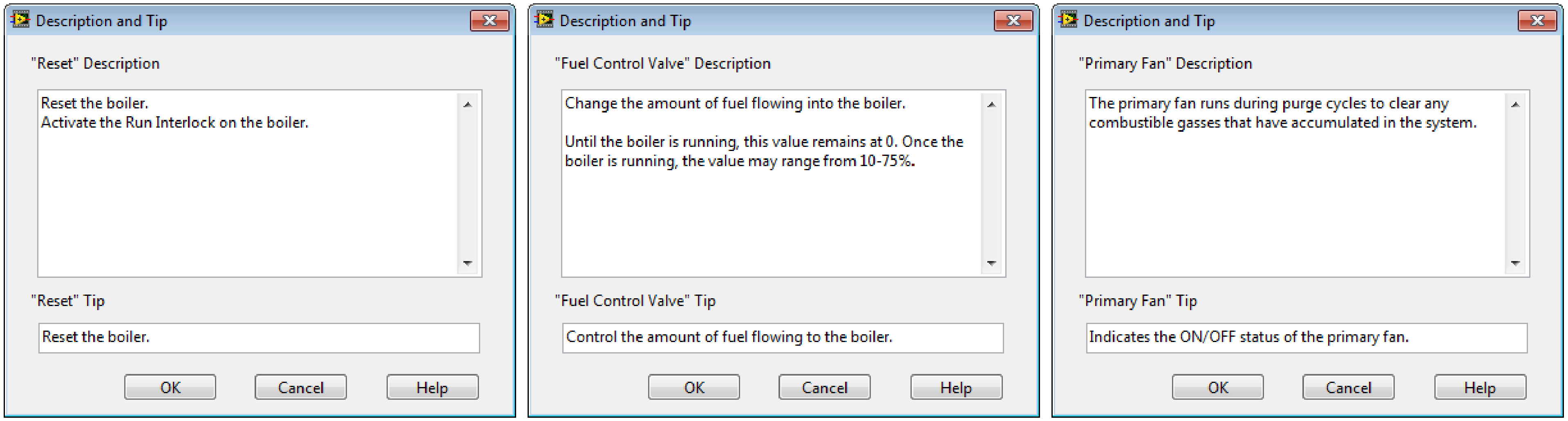
2. Enter documentation for the **Context Help** window for each control and indicator on the Main VI front panel. Make sure the documentation includes functionality and, if applicable, valid ranges and default values.

   ☐ Right-click the control or indicator and select **Description and Tip** from the shortcut menu.

   ☐ Create a description for each control and indicator. Figure 4-12 shows a few examples of control and indicator documentation.

Tip Refer to the requirements document for information about each control and indicator.

Figure 4-12. Examples of Control and Indicator Documentation



"Reset" Description

Reset the boiler.
Activate the Run Interlock on the boiler.

"Reset" Tip

Reset the boiler.

"Fuel Control Valve" Description

Change the amount of fuel flowing into the boiler.

Until the boiler is running, this value remains at 0. Once the boiler is running, the value may range from 10-75%.

"Fuel Control Valve" Tip

Control the amount of fuel flowing to the boiler.

"Primary Fan" Description

The primary fan runs during purge cycles to clear any combustible gasses that have accumulated in the system.

"Primary Fan" Tip

Indicates the ON/OFF status of the primary fan.

3. Update documentation folders on disk.

   ☐ In the **Project Explorer** window, right-click **Queued Message Handler Documentation.html** under the **Project Documentation** folder and select **Explore** from the shortcut menu.

   ☐ Create a new folder, name it `Queued Message Handler template` and move all the files into the new folder.

   ☐ Create a new folder, called `Boiler Controller`, in the `<Exercises>\LabVIEW Core 3\Course Project\documentation`.

4. Update the LabVIEW project to recognize the new documentation folders.

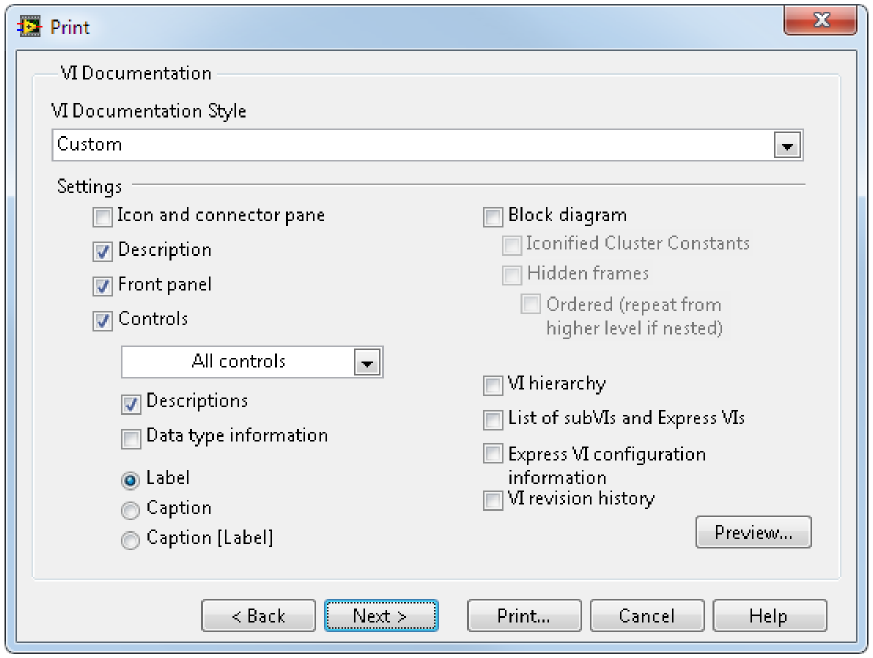
   - ☐ In the **Project Explorer** window, select **Queued Message Handler Documentation.html** and the **Documentation Images** virtual folder and select **Remove from Project**.
   - ☐ Right-click **Project Documentation** and select **Convert to Auto-populating folder**.
   - ☐ In the Select Folder to Connect dialog box, select documentation and click the **Select Folder** button.

5. Create printable documentation for the application.

   - ☐ Open `Main.vi`.
   - ☐ Select **File»Print** from the menu to display the **Print** wizard.
   - ☐ Select the `Main.vi` radio button and click **Next**.
   - ☐ Select the **VI documentation** radio button and click **Next**.

☐ Select the options as shown in Figure 4-13 and click **Next**.

Figure 4-13. Print VI Documentation Options


Print

VI Documentation

VI Documentation Style: Custom

Settings

- [ ] Icon and connector pane
- [x] Description
- [x] Front panel
- [x] Controls
  - All controls
  - [x] Descriptions
  - [ ] Data type information
  - ● Label
  - ○ Caption
  - ○ Caption [Label]
- [ ] Block diagram
  - [ ] Iconified Cluster Constants
  - [ ] Hidden frames
    - [ ] Ordered (repeat from higher level if nested)
- [ ] VI hierarchy
- [ ] List of subVIs and Express VIs
- [ ] Express VI configuration information
- [ ] VI revision history

☐ Select **HTML file** and click **Next**.

☐ Leave the default settings in the **HTML** page of the wizard and click **Save**.

☐ Save the file to `<Exercises>\LabVIEW Core 3\Course Project\documentation\Boiler Controller\Main.html`.

6. Link to the printable documentation in the VI description.

☐ In the Main VI, select **File»VI Properties**.

☐ Select **Documentation** from the **Category** pull-down menu.

- ☐ Modify the **Help path** to use a relative path to the file you created in step 5. by entering `..\Course Project\documentation\Boiler Controller\Main.html` in the **Help path** text box.
- ☐ Click **OK**.

7. Rename the **documentation** virtual folder to Project Documentation to better describe the contents.

- ☐ Right-click the **documentation** folder and select **Stop Auto-populating**. The name of an auto-populating folder must always match the name of the folder on disk.
- ☐ Rename the folder `Project Documentation`.

8. Save Main VI.

Test the Application

Verify that previous functionality still works and that user interface events result in execution of the appropriate case in the boiler controller.

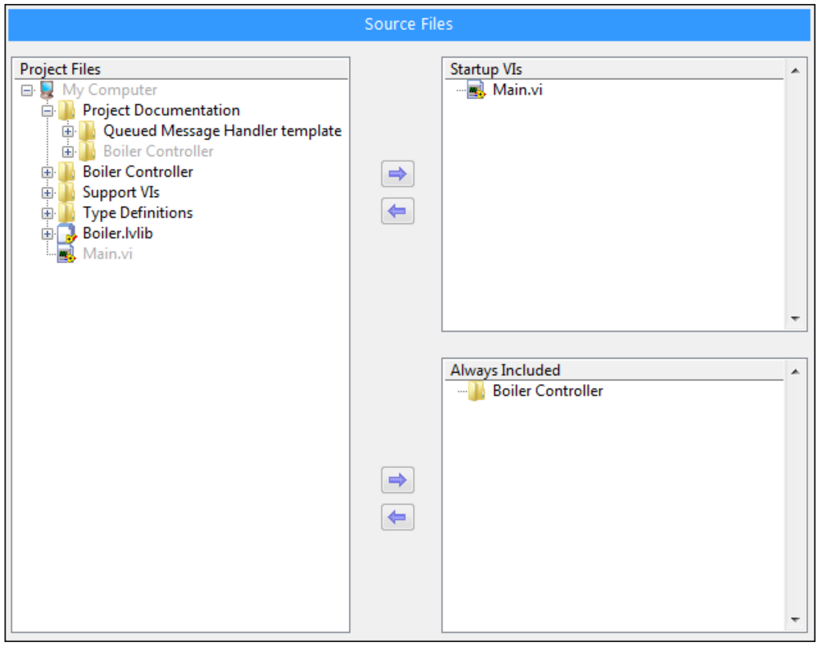
1. Test the VI.

- ☐ Run Main VI.
- ☐ Verify that the previous functionality still works.
- ☐ Verify that control and indicator descriptions appear in the **Context Help** window and LabVIEW displays tip strips the user hovers the cursor over controls and indicators.
- ☐ Verify that updated VI description appears in the **Context Help** window when you hover over the VI icon.
- ☐ Click the **Detailed Help** link in the **Context Help** window for one of the controls or indicators and review the documentation from the HTML file.

2. Update the build specification and test the new executable.

- ☐ Right-click the Main Application build specification and select **Properties** from the shortcut menu.
- ☐ Click **Source Files** in the **Category** list.
- ☐ Select **Project Documentation»Boiler Controller** in the **Project Files** tree and move this folder to the **Always Included** section, as shown in Figure 4-14. Including this folder ensures that the HTML file and all images will be included in the build.

Figure 4-14. Including the Boiler Controller Documentation in the Build Specification



☐ Click **OK** to save the build specification.

☐ Right-click the Main Application build specification and select **Build** from the shortcut menu.

☐ Click the **Explore** button when LabVIEW finishes the build.

☐ Double-click `Boiler Controller.exe` to run the application.

☐ Verify the behavior you tested in step 1.

☐ View the VI documentation by selecting **Help»Help for this VI** from the menu.

☐ Click the **Emergency Stop** button and verify that all VIs stop and the application exits.

End of Exercise 4-2

Exercise 4-3 Initialize an Application from a File

Goal

Initialize your application by reading configuration data from an INI file.

Scenario

In the sprint planning meeting for this iteration, the product owner chose to implement the following user story:

- As a boiler operator, I want to specify the configuration settings that the boiler controller uses, so that I can reuse the controller for boilers with different configuration settings.

After discussing this user story with the team, you have decided to read the configuration settings from an INI file. The requirements document indicates that the Fuel Control Valve should be limited to values between 10 and 75%. Since this may vary from boiler to boiler, those values should be read from a configuration file instead of written as constants on the block diagram.

Your team identified the following development tasks that must be completed:

- Update `MHL Data.ctl` to contain the Fuel Control Valve limits that you want to read from the INI file.
- Create the INI file that contains configuration constants for the system.
- Create a VI that allows the user to specify the INI file, reads configuration constants from that file, and passes those values to the rest of the application.
- Modify existing code to use the configuration constants.

Design

While it is possible to create the INI file manually, your team decides that would be a good idea to create a VI that programmatically creates the INI file. That way, you can easily add more constants to the file or reuse the code to create INI files for other boilers that require different constant values.

Develop the INI file and the VI that reads it in a manner that allows for future expansion in case you want to scale it to read other constants for the MHL, the boiler controller, or the boiler. To do this, you create a subVI that reads the boiler constants. If you eventually need to read constants for the other loops, you can create separate subVIs to read the constants for each of those loops.

Implementation

1. Add configuration files to the project.

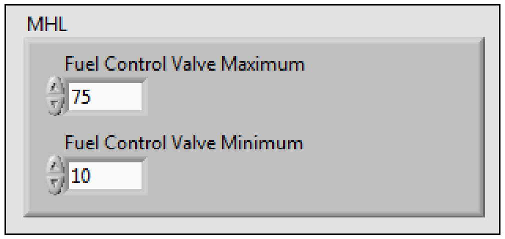
   ☐ In Windows Explorer, copy the `Configuration` directory located in the `<Exercises>\LabVIEW Core 3\External\Support VI` directory and paste it into the `<Exercises>\LabVIEW Core 3\Course Project\support` directory.

   ☐ Create a virtual folder called `Configuration` in the Support VIs folder of the Boiler Controller project. Add only `Write Configuration Settings File.vi` to the virtual folder.

2. Update the INI file with new default values.

   ☐ Open the Write Configuration Settings File VI and verify the **MHL** default values are set to the values shown in Figure 4-15.

Figure 4-15. Write Configuration Settings File VI Front Panel



   ☐ Run the Write Configuration Settings File VI and save the file as `Boiler Init.ini` in the `<Exercises>\LabVIEW Core 3\Course Project` directory.

   ☐ Add `Boiler Init.ini` to the project.

   – Right-click **My Computer** in the **Project Explorer** window and select **Add»File** from the shortcut menu.

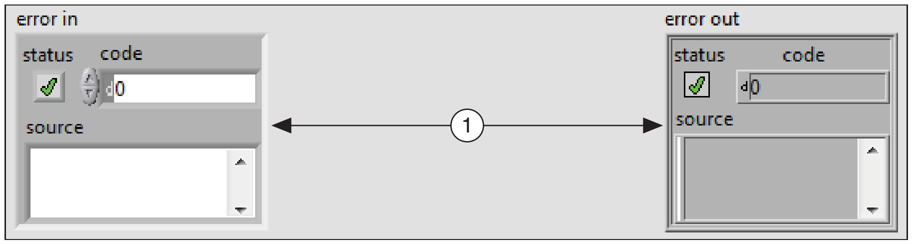
   – Navigate to `Boiler Init.ini` and click the **Add File** button to add the file to the project.

3. Create a VI that allows the user to specify the INI file, read configuration constants from that file, and pass those values to the rest of the application.

   ☐ Add a new VI to the project under the **Support VIs»Configuration** virtual folder and save the VI as `Read UI Constants.vi`.

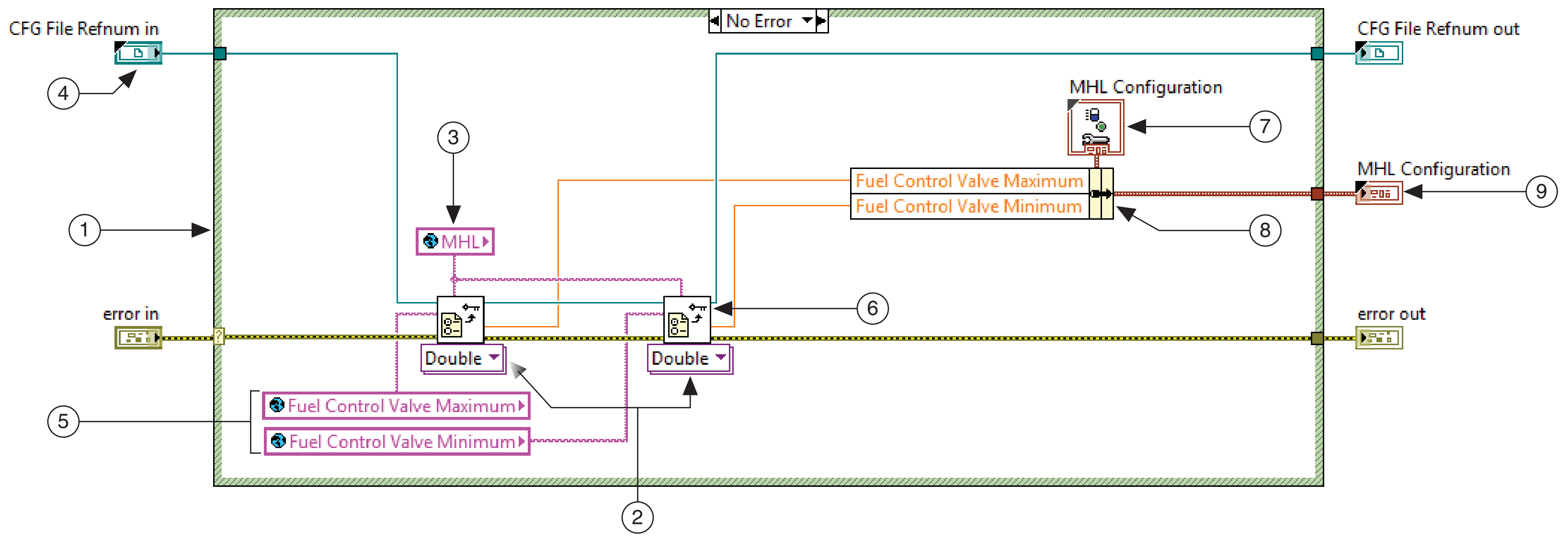
   ☐ Create the front panel and block diagram of the Read UI Constants VI as shown in Figures 4-16 through 4-19.

Figure 4-16. Read UI Constants VI Front Panel



1	**Error In** and **Error Out**—Place an Error In and Error Out control and indicator on the front panel. After you complete the block diagram, the front panel will contain more items.

Figure 4-17. Read UI Constants VI Block Diagram No Error Case



1 **Case Structure**—Place a Case structure on the diagram and wire the **error in** cluster to the case selector.

2 **Read Key**—This VI reads a value from the key in a MHL section of the configuration data. Select **Double** from the polymorphic selector.

3 **Boiler System Globals variable**—Drag **Boiler System Globals.vi** from the **Project Explorer** window and select **MHL** from the list. Right-click the global variable and select **Change To Read** from the shortcut menu. Wire this global variable to the **section** input of each of the Read Key VIs.

4 Refnum—Right-click the **refnum** input of the Read Key VI and select **Create»Control**. Rename the refnum `CFG File Refnum in` and move the refnum outside the Case structure. Re-wire the refnum if necessary.

5 **Boiler System Globals variable**—Change the global variables to read and select **Fuel Control Valve Minimum** and **Fuel Control Valve Maximum** from the list. Wire these variables to the **key** input of the Read Key VI.

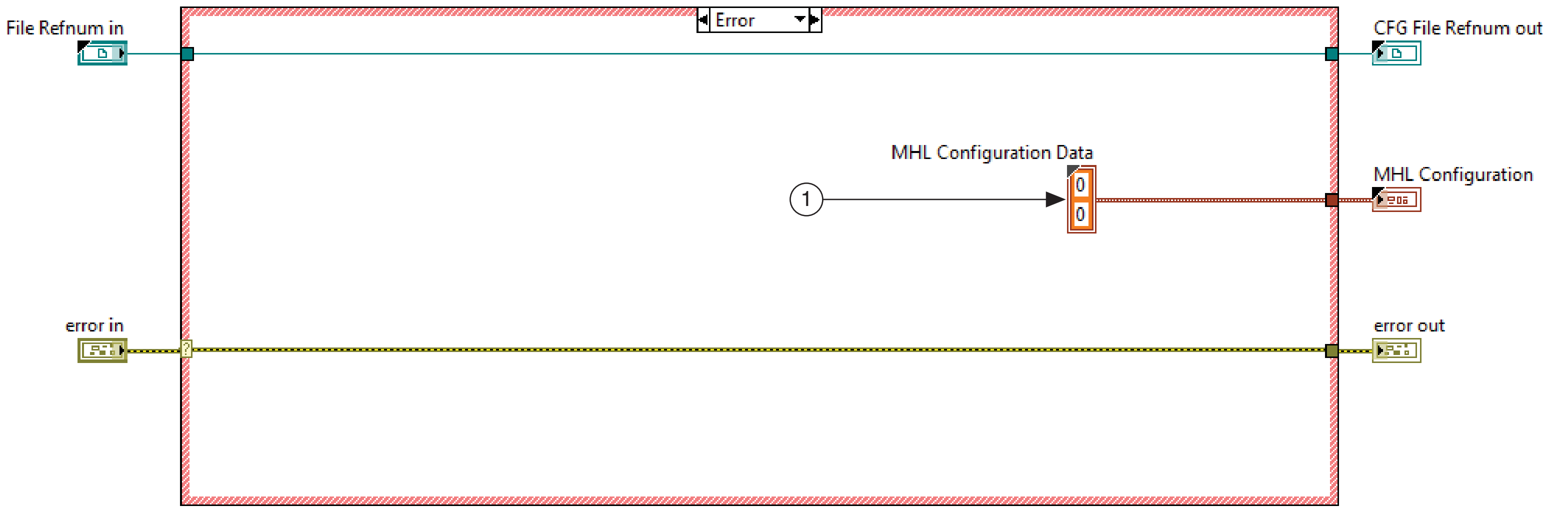
6 Refnum—Right-click the **refnum out** output of the Read Key VI and select **Create»Indicator**. Rename the refnum `CFG File Refnum out` and move the refnum outside the Case structure. Re-wire the refnum if necessary.

7 **MHL Configuration constant**—Drag **MHL Configuration.ctl** from the **Type Definitions** folder in the **Project Explorer** window.

8 **Bundle By Name**—Wire the MHL Configuration constant to the **input cluster** input of the Bundle By Name function. Expand the function to display two terminals. Wire the **value** output of each of the Read Key VIs to the Bundle By Name function.

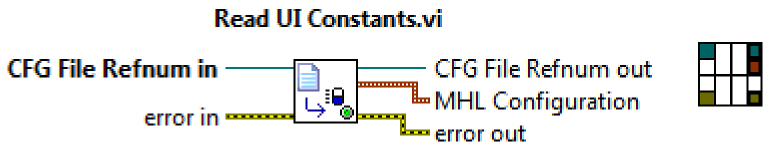
9 **MHL Configuration indicator**—Right-click the **output cluster** output of the Bundle by Name function and select **Create»Indicator**. Rename the indicator `MHL Configuration` and move it outside the Case structure. Wire the Bundle by Name function to the indicator.

Figure 4-18. Read UI Constants VI Block Diagram Error Case



1 **MHL Configuration constant**—Right-click the tunnel and select **Create»Constant** to create this constant.

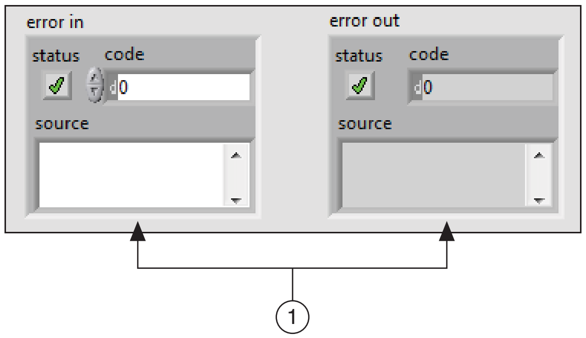
Figure 4-19. Read UI Constants VI Connector Pane



1 Filter glyphs by keywords, `file`, `get`, and `control` to find appropriate glyphs for the icon.

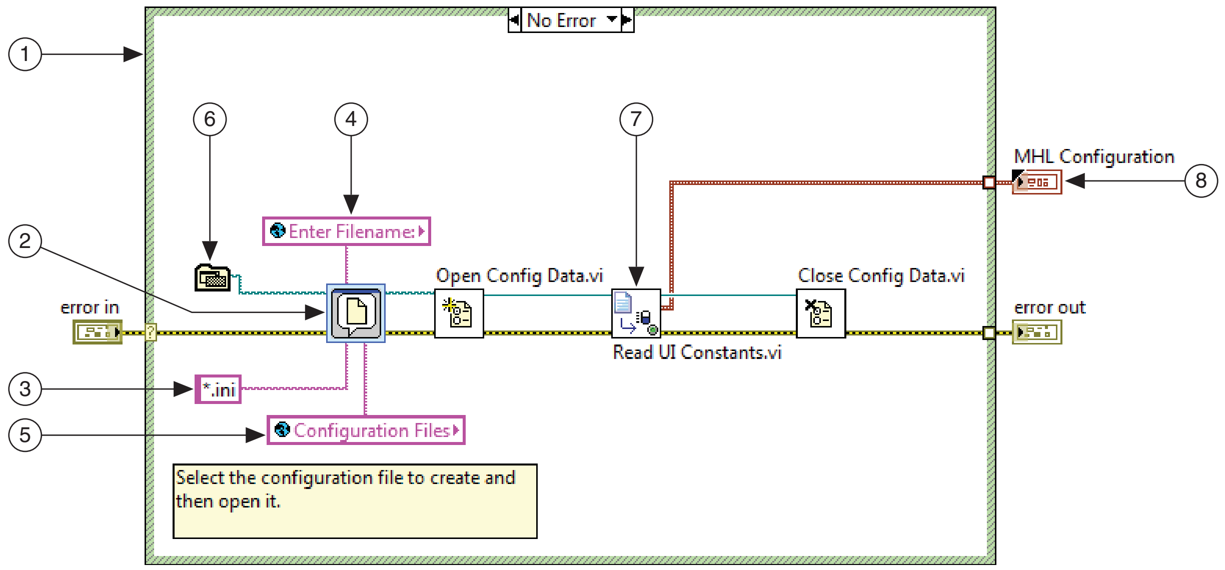
- ☐ Save the Read UI Constants VI.
- ☐ Add a new VI to the project under the **Support VIs»Configuration** virtual folder and save the VI as `Read Configuration Data.vi` in the `<Exercises>\LabVIEW Core 3\Course Project\support\Configuration` directory.
- ☐ Create the front panel and block diagram of the Read Configuration Data VI as shown in Figures 4-20 through 4-23. The Read Configuration Data VI reads the MHL configuration constants from the INI file and passes those values to the MHL.

Figure 4-20. Read Configuration Data VI Front Panel



1 **Error In** and **Error Out**—Place an Error In and Error Out control and indicator on the front panel. After you complete the block diagram, the front panel will contain more items.

Figure 4-21. Read Configuration Data VI Block Diagram No Error Case



1 **Case Structure**—Place a Case structure on the diagram and wire the **error in** cluster to the case selector.

2 **File Dialog**—In the **Configure File Dialog** dialog box, place a checkmark in the **Limit selection to single item** checkbox and select the **File** and **New or existing** radio buttons. View the Express VI as an icon to save room on the block diagram.

3 String constant—Create a constant for the **pattern (all files)** input of the File Dialog Express VI and enter `*.ini`.

4 **Boiler System Globals variable**—Drag **Boiler System Globals.vi** from the **Project Explorer** window and select **Enter Filename** from the list. Right-click the global variable and select **Change To Read** from the shortcut menu. Wire this global variable to the **prompt** input of the File Dialog Express VI.

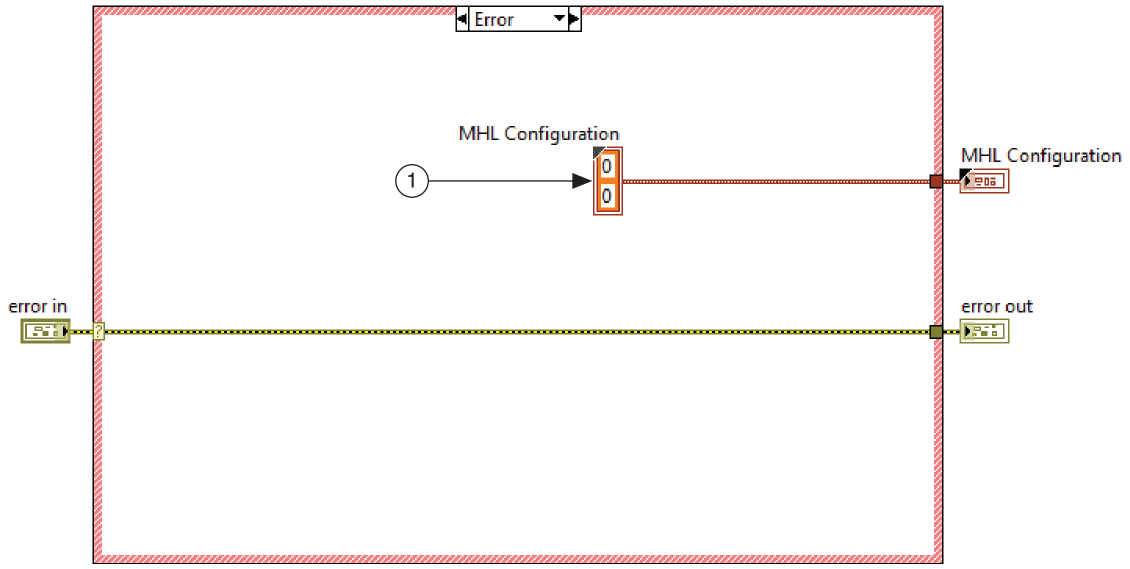
5 **Boiler System Globals variable**—Change the global variable to read and select **Configuration Files** from the list. Wire it to the **pattern label** input of the File Dialog Express VI.

6 **Application Directory**—Wire to the **start path** input of the File Dialog Express VI. Application Directory returns the path to the directory containing the application.

7 **Read UI Constants**—Drag this VI from the **Support VIs»Configuration** folder in **Project Explorer** window to the block diagram. You created this VI in step 3.

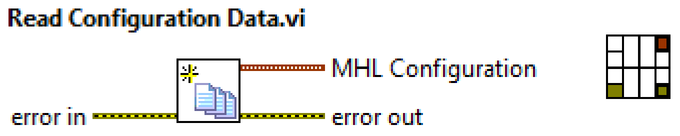
8 **MHL Configuration indicator**—Right-click the **MHL Configuration** output of the Read UI Constants VI and select **Create»Indicator**. Move the indicator outside of the Case structure and wire Read UI Constants VI to the indicator.

Figure 4-22. Create Read Configuration Data VI Block Diagram Error Case



1 **MHL Configuration constant**—Right-click the tunnel and select **Create»Constant** to create this constant.

Figure 4-23. Read Configuration Data VI Connector Pane

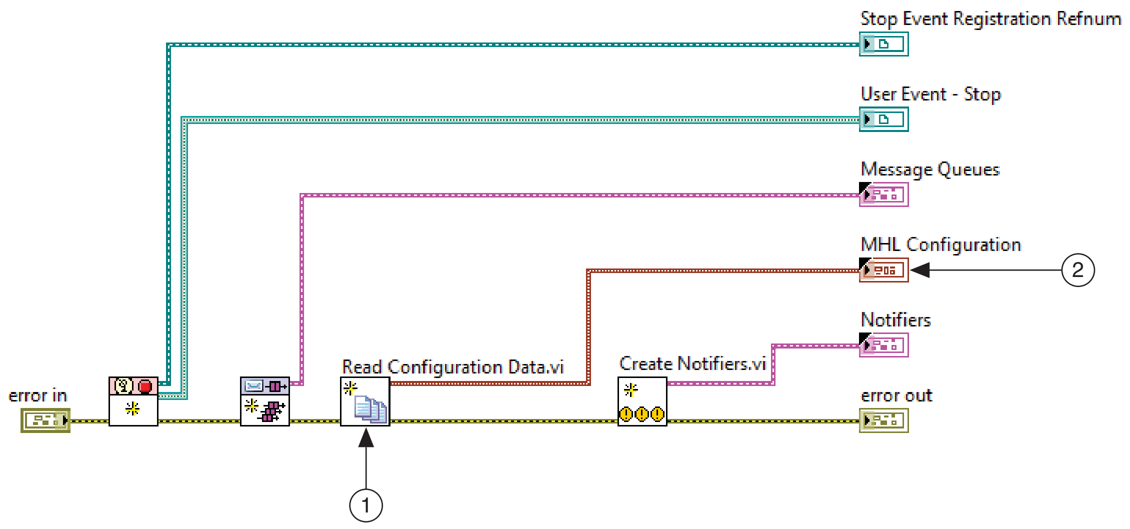


1	Filter glyphs by keywords, `new`, and `file` to find appropriate glyphs for the icon.

4. Update `Boiler System Open.vi` to include `Read Configuration Data.vi`.

   - ☐ Open **Support VIs»Boiler System Open.vi** from the **Project Explorer** window.
   - ☐ Update the VI as shown in Figure 4-24.

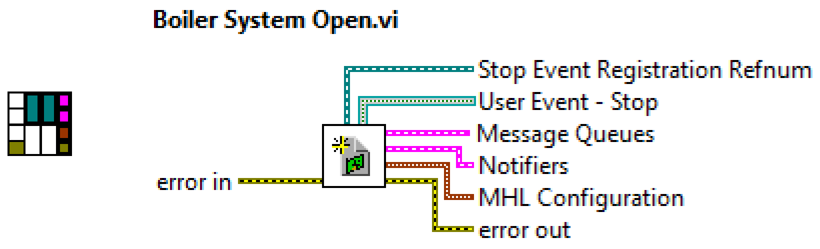
Figure 4-24. Boiler System Open VI including Read Configuration Data VI



1	Read Configuration Data—Drag **Read Configuration Data.vi** from the **Project Explorer** window.
2	**MHL Configuration indicator**—Right-click the **MHL Configuration** output of the Read Configuration Data VI and select **Create»Indicator**.

☐ Add MHL Configuration to the connector pane as shown in Figure 4-25.

Figure 4-25. Boiler System Open Icon and Connector Pane



5. Add a VI to initialize the user interface, set the range for the Fuel Control Valve, and build the MHL Data cluster to the project.

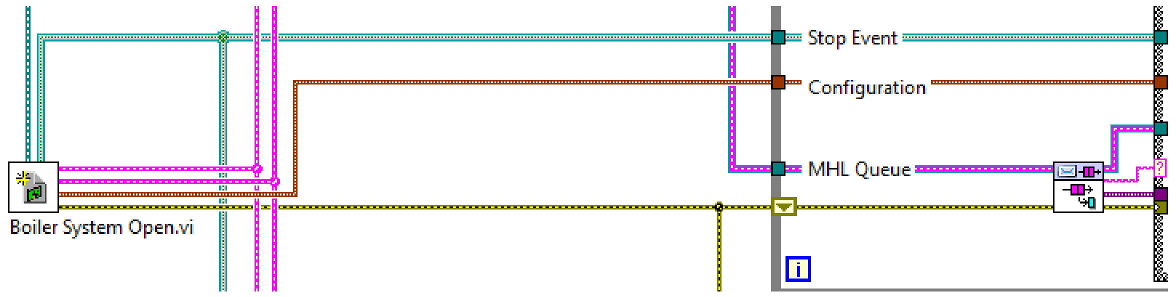
   ☐ Navigate to `<Exercises>\LabVIEW Core 3\External\Support VIs` and copy `Initialize Panel.vi` to `<Exercises>\LabVIEW Core 3\Course Project\support`.

   ☐ Add `Initialize Panel.vi` to the **Support VIs** virtual folder in the Boiler Controller Project Explorer window.

6. Update the Main VI to read the configuration data and pass the data to the Initialize case of the MHL.

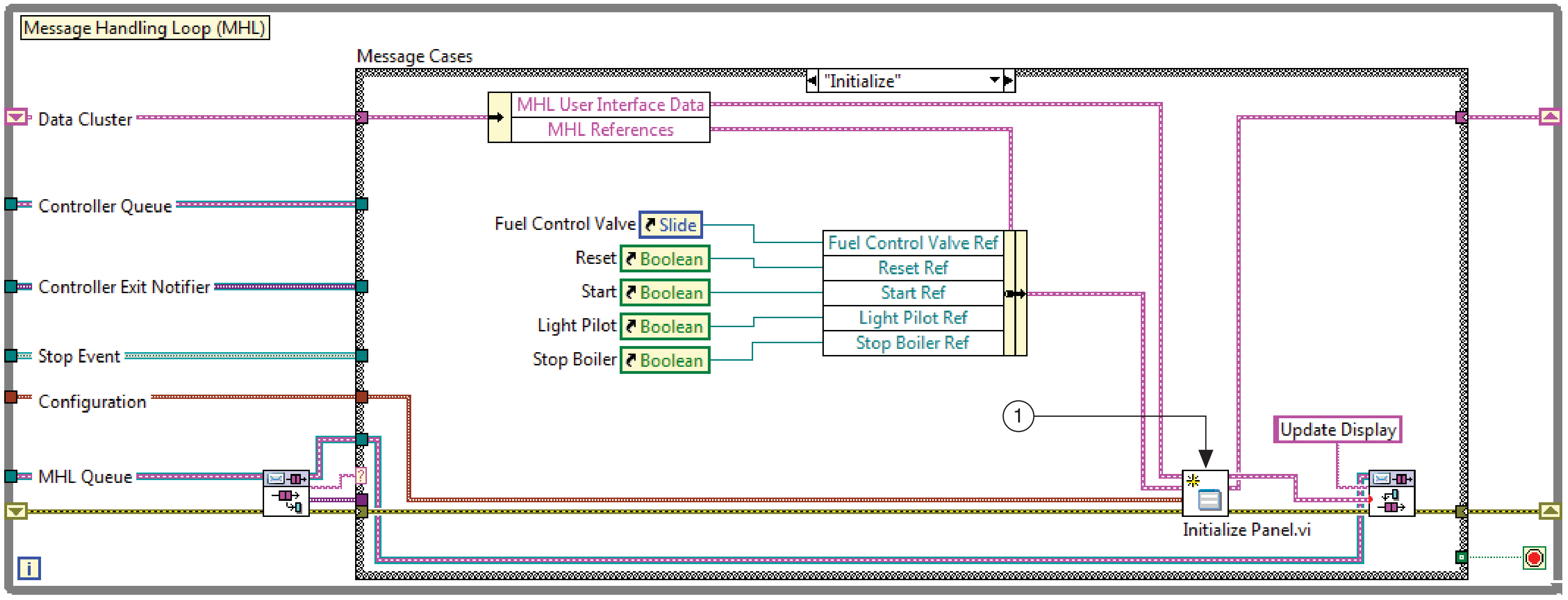
   ☐ Wire the **MHL Configuration** output from the Boiler System Open VI to the MHL as shown in Figure 4-26.

Figure 4-26. Wiring the MHL Configuration Data to the MHL



☐ Update the Initialize case of the MHL as shown in Figure 4-27.

Figure 4-27. Updated Initialize Case of the Main MHL



1 **Initialize Panel**—Drag the Initialize panel VI from the **Project Explorer** window. Wire the Initialize Panel VI as shown.

☐ Save the Main VI.

Test the Application

Verify that previous functionality still works and that user interface events result in execution of the appropriate case in the boiler controller.

1. Test the VI.

   ☐ Run Main VI.

   ☐ Verify that a dialog prompts you to select an INI file.

   ☐ Select `Boiler Init.ini` in the `<Exercises>\LabVIEW Core 3\Course Project\` directory.

   ☐ Verify previous functionality still works, such as the display of one-button dialog boxes when you click the **Reset**, **Start**, or **Light Pilot** buttons.

   ☐ Verify the functionality of the **Fuel Control Valve** control.

   – The **Fuel Control Valve** control is initialized to `10`.

   – The range of the **Fuel Control Valve** horizontal slider control is limited to values between 10 and 75.

   – Changing the value displays the one-button dialog box.

2. Update the build specification and test the new executable.

   ☐ Right-click the Main Application build specification and select **Properties** from the shortcut menu.

   ☐ Click **Source Files** in the **Category** list.

   ☐ Select **Boiler Init.ini** in the **Project Files** tree and move this file to the **Always Included** section.

   ☐ Click **OK** to save the build specification.

   ☐ Right-click the Main Application build specification and select **Build** from the shortcut menu.

   ☐ Click the **Explore** button when LabVIEW finishes the build.

☐ Double-click `Boiler Controller.exe` to run the application.

☐ Verify the behavior you tested in step 1.

Note The starting directory to browse to the INI file has changed. The build now creates the `Boiler Controller.ini`. This file is NOT the INI file that you created. `Boiler Init.ini` is inside the data folder.

☐ Click the **Emergency Stop** button and verify that all VIs stop and the application exits.

End of Exercise 4-3

Exercise 4-4 Improve Application Usability

Goal

Customize the user interface to improve its usability.

Scenario

The product owner is finally ready to implement the user interface feedback that the customer representative provided after viewing the user interface prototype.

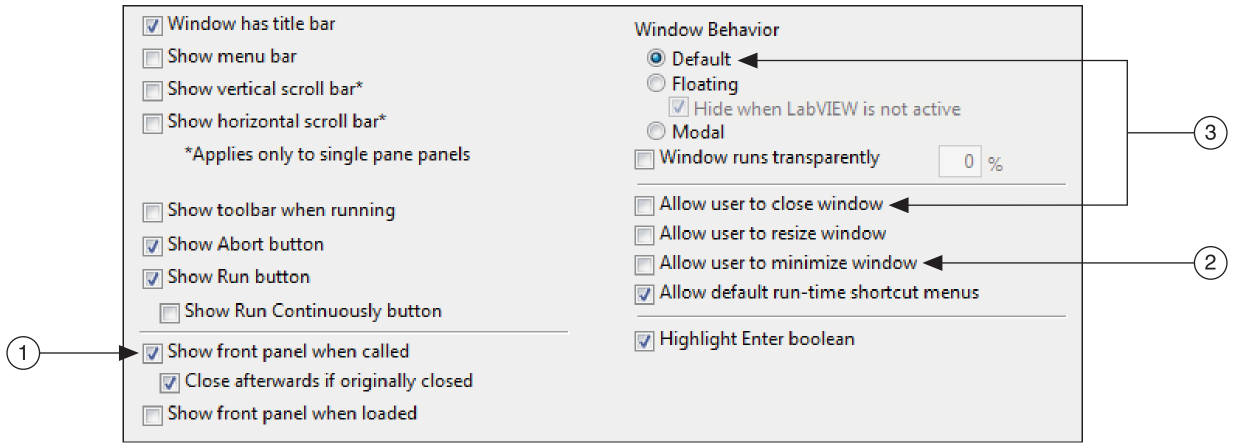
- Launch the front panel of `Boiler.vi` as a separate window.
- Customize the run-time window appearance to look less like LabVIEW.
- The **Fuel Control Valve** control is not very intuitive. Either customize its appearance or replace it with a different type of numeric control.
- Reorganize the UI to more logically group controls.
- Create a custom run-time menu to stop the application or load the Help file.

Implementation

1. Update the Boiler VI so that it launches as a separate window.

   - ☐ Open **Boiler.vi** from **Boiler.lvlib** in the **Project Explorer** window.
   - ☐ Select **File»VI Properties** from the menu and select **Window Appearance** from the **Category** pull-down menu.
   - ☐ Uncheck the **Same as VI name** checkbox and enter `Boiler` in the Window title field.
   - ☐ Select the **Dialog** radio button.
   - ☐ Click the **Customize** button and update the **Customize Window Appearance** dialog box as shown in Figure 4-28.

Figure 4-28. Customize Window Appearance Dialog Box for Boiler VI



☑ Window has title bar
☐ Show menu bar
☐ Show vertical scroll bar*
☐ Show horizontal scroll bar*
*Applies only to single pane panels

☐ Show toolbar when running
☑ Show Abort button
☑ Show Run button
☐ Show Run Continuously button

☑ Show front panel when called
☑ Close afterwards if originally closed
☐ Show front panel when loaded

Window Behavior
● Default
○ Floating
☑ Hide when LabVIEW is not active
○ Modal
☐ Window runs transparently 0 %

☐ Allow user to close window
☐ Allow user to resize window
☐ Allow user to minimize window
☑ Allow default run-time shortcut menus

☑ Highlight Enter boolean

1 Note that the front panel displays when the Boiler VI is called and closes when the Boiler VI finishes executing.
2 The user does not have the ability to minimize or resize the Boiler VI window.
3 Select **Default** as the **Window Behavior** and remove the checkmark from the **Allow user to close window** checkbox.

- ☐ Click **OK** to accept changes in the **Customize Window Appearance** and **VI Properties** dialog boxes.
- ☐ Save and close the Boiler VI.
- ☐ Run the Main VI to view the result of these changes.

2. Customize the run-time window appearance of Main VI so that it looks less like a LabVIEW dialog box.

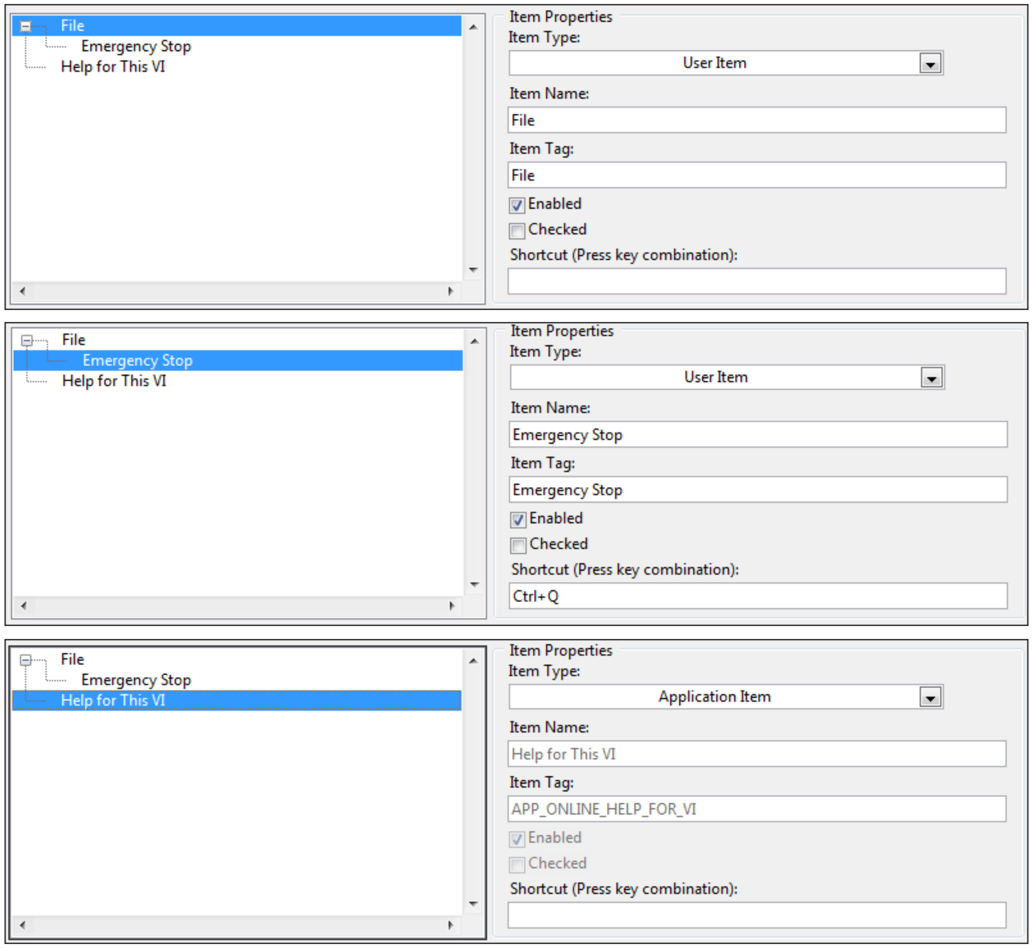
   - ☐ Open **Main.vi** from the **Project Explorer** window.
   - ☐ Select **File»VI Properties** from the menu and select **Window Appearance** from the **Category** pull-down menu.
   - ☐ Uncheck the **Same as VI name** check box and enter `Boiler Controller` in the Window title field.
   - ☐ Select the **Top-level application window** radio button.
   - ☐ Click **OK** to accept changes in the **VI Properties** dialog boxes.
   - ☐ Run the Main VI to view the changes to the UI when executing as a top-level application window.

3. Create a custom run-time menu to stop the application and load the Help file.

   - ☐ In the Main VI select **Edit»Run-Time Menu** from the LabVIEW menu.
   - ☐ In the **Menu Editor** dialog box, select **Custom** from the pull-down menu.

☐ Add menu items as shown in Figure 4-29.

Figure 4-29. Custom Run-Time Menu for the Main VI



- File
  - Emergency Stop
- Help for This VI

Item Properties (File)
Item Type: User Item
Item Name: File
Item Tag: File
☑ Enabled
☐ Checked
Shortcut (Press key combination):

Item Properties (Emergency Stop)
Item Type: User Item
Item Name: Emergency Stop
Item Tag: Emergency Stop
☑ Enabled
☐ Checked
Shortcut (Press key combination): Ctrl+Q

Item Properties (Help for This VI)
Item Type: Application Item
Item Name: Help for This VI
Item Tag: APP_ONLINE_HELP_FOR_VI
☑ Enabled
☐ Checked
Shortcut (Press key combination):

☐ Select **File»Save** and save the custom menu as `Main.rtm` in the `<Exercises>\LabVIEW Core 3\Course Project` directory.

☐ Close the menu editor.

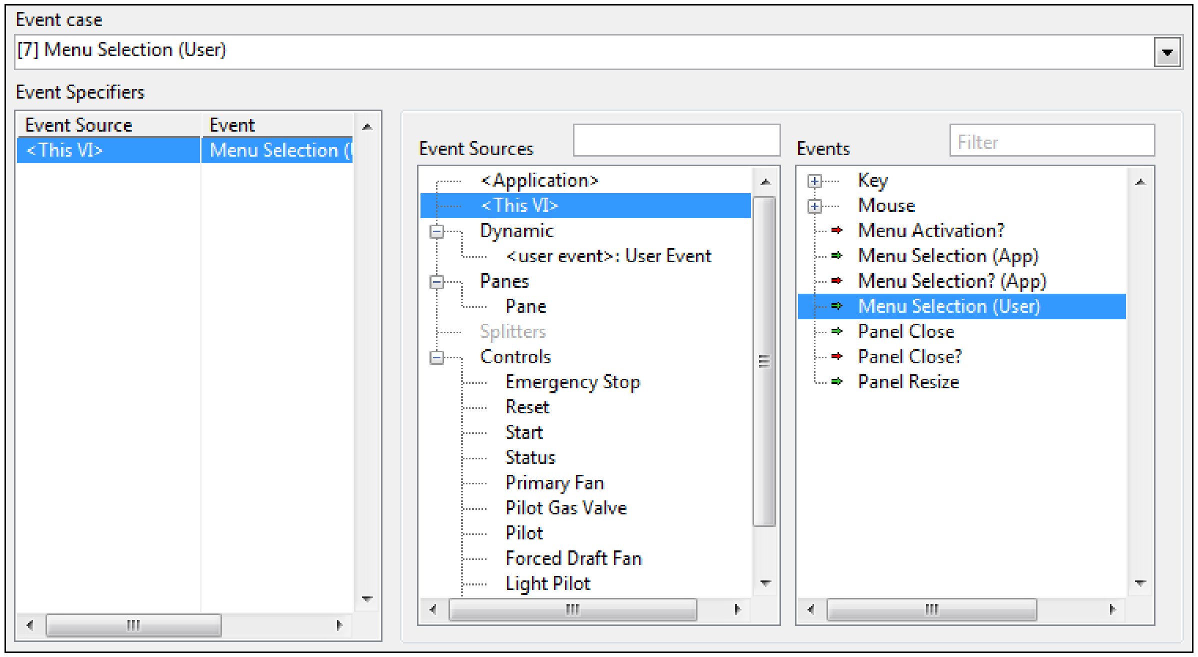
☐ Click **Yes** when prompted to change the run-time menu.

4. Modify the EHL to include an event case for menu selection.

☐ In the EHL, right-click the Emergency Stop: Value Change event and select **Duplicate Event Case** from the shortcut menu.

☐ Add a new event to the EHL Event structure as shown in Figure 4-30.

Figure 4-30. Menu Selection Event for the EHL



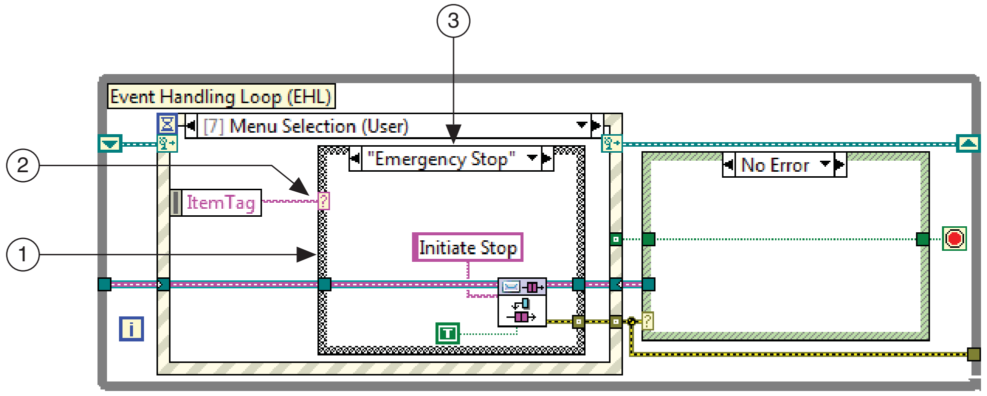
Event case
[7] Menu Selection (User)

Event Specifiers

Event Source	Event
<This VI>	Menu Selection (

☐ Modify the event as shown in Figure 4-31 and Figure 4-32.

Figure 4-31. Block Diagram for Menu Selection Emergency Stop Event

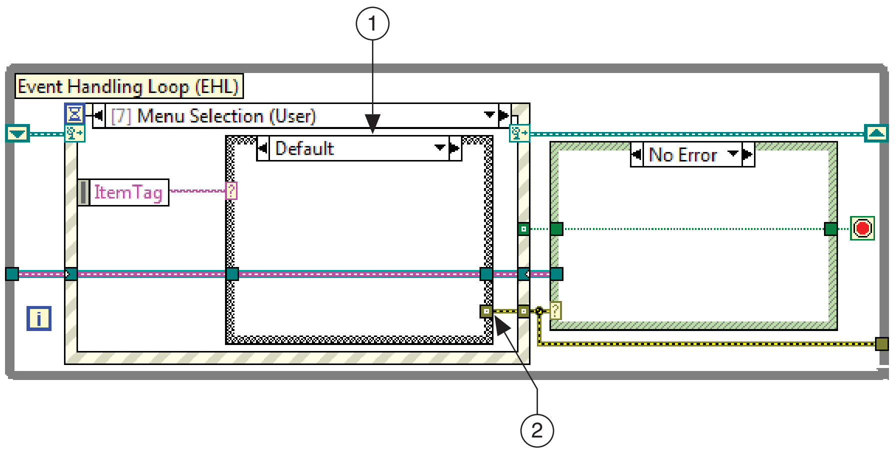


1 **Case Structure**—Place a Case structure around the Enqueue Message VI.

2 Wire **ItemTag** to the case selector. This case sends an Initiate Stop message to the MHL if the ItemTag is Emergency Stop.

3 Change the name to `Emergency Stop`.

Figure 4-32. Block Diagram for Menu Selection Default Event



1	Change the name of the default case to `Default`.
2	Right-click the tunnel and select **Use Default If Unwired** from the shortcut menu.

☐ Run the Main VI to test your changes.

Test the Application

Verify that previous functionality still works and that user interface events result in execution of the appropriate case in the boiler controller. There is no need to test the entire VI, since you've done that after each step of this exercise.

1. Update the build specification and test the new executable.

   ☐ Right-click the Main Application build specification and select **Build** from the shortcut menu.

   ☐ Click the **Explore** button when LabVIEW finishes the build.

   ☐ Double-click `Boiler Controller.exe` to run the application.

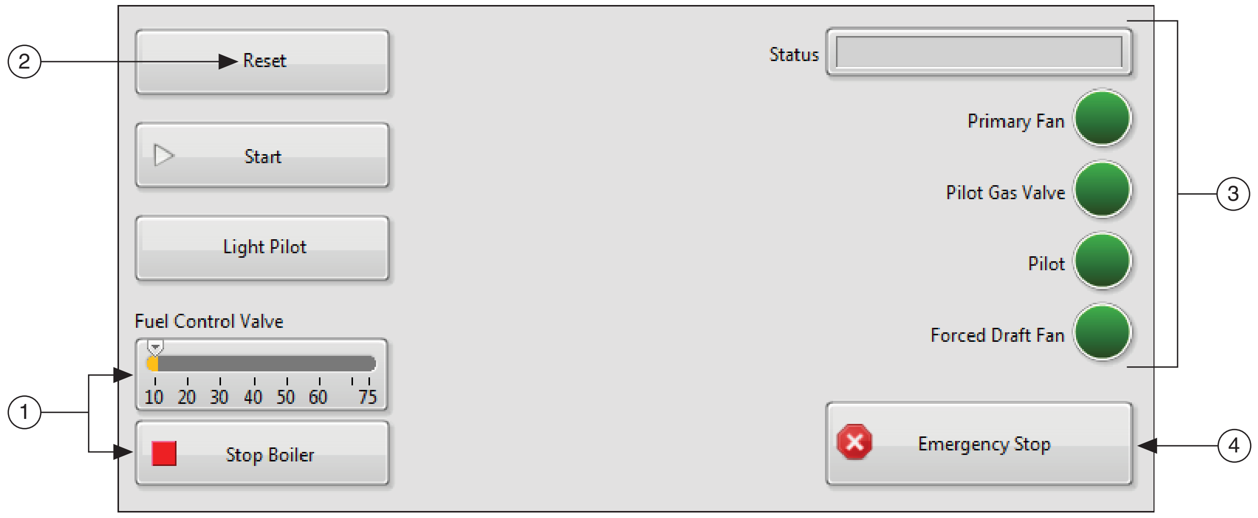
   ☐ Verify the previously implemented functionality.

- ☐ Verify the changes you made in this exercise.
  - – Is the window appearance as expected?
  - – Does the front panel of the Boiler VI appear when you run the Main VI?
  - – Use the slider to change the value of Fuel Control Valve.
  - – View the Help file from the menu.
- ☐ Verify that selecting **File»Emergency Stop** from the menu or pressing <Ctrl-Q> halt the application and close both windows.

Challenge

1. Reorganize the UI to more logically group controls as shown in Figure 4-33.

Figure 4-33. Reorganized Main VI Front Panel



1 Group these controls together because you only use them after the boiler is running.

2 Center-align the text for each Boolean button.

3 Right-justify all indicators.

4 Emergency Stop in the bottom-right fits the general expectation for the Stop buttons to be in the bottom-right corner of the window.

End of Exercise 4-4

5

Managing and Logging Errors

Topics

Exercise 5-1 Manage Errors

Goal

Modify your code to gracefully handle both critical and non-critical errors.

Scenario

In the sprint planning meeting for this iteration, the product owner chose to implement the following user stories:

- As a boiler operator, I want the boiler controller to prevent the boiler from entering an unsafe state as a result of a critical error.
- As a boiler operator, I want the system to be robust enough that minor errors do not result in system shut down, so that the boiler only shuts down if the error prevents safe operation.

Critical errors require shutting down the application because they cannot be otherwise managed. If they are not handled, critical errors may result in injury to personnel or damage to equipment. They may also place the system into an unmanageable state.

Non-critical errors are more akin to inconveniences that should not prevent continued execution of the application.

After discussing these user stories with the team, you identified and classified the following errors:

Table 5-1. Errors

Cause of Error	Critical or Non-Critical
Cancelling the Open File dialog when loading the INI file	Non-critical
The user chooses an invalid INI file	Critical
During development, you accidentally mistype a message string or case name	Critical

To address these errors, your team identifies the following tasks:

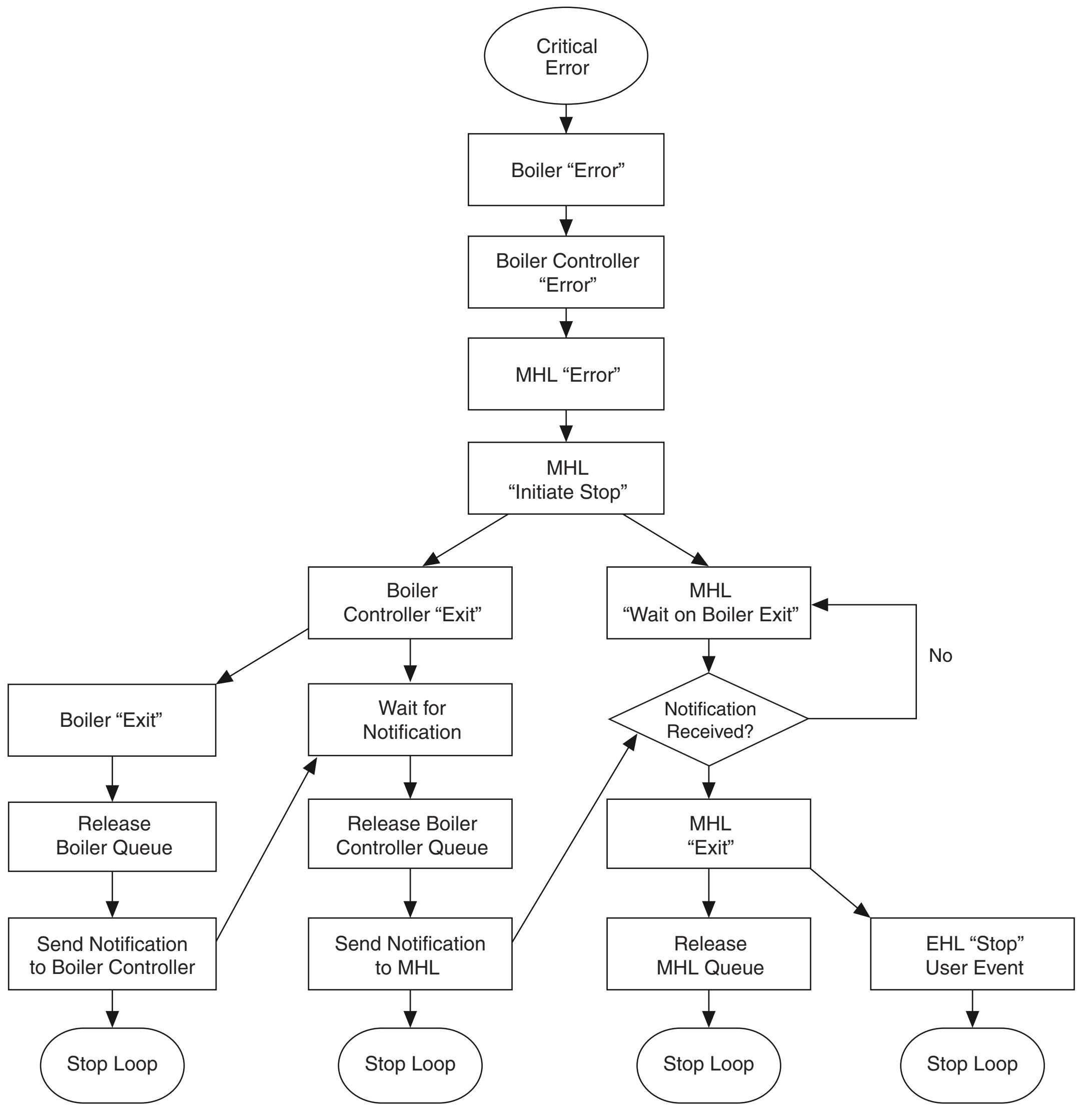
1. Modify `Read Configuration Data.vi` to handle cancellation of the Open File dialog.
2. Modify `Read UI Constants.vi` to handle selection of an invalid INI file.
3. Modify the Default case of the Message Handling Loop and Boiler Controller to handle typos when sending messages.
4. Modify the Message Handling Loop and Boiler Controller Loop to handle shutting down in the event of a critical error.

Design

The team decides on the following strategies for approaching each task:

1. Modify `Read Configuration Data.vi` to handle cancellation of the Open File dialog.
    - Modify `Read Configuration Data.vi` to load a default INI file if the user cancels the Open File dialog.
2. Modify `Read UI Constants.vi` to handle selection of an invalid INI file.
    - Modify `Read UI Constants.vi` to check that each key was properly read from the INI file.
    - Use an error ring to generate a custom error to indicate that the INI file is not valid.
3. Modify the Default case of the Message Handling Loop and Boiler Controller to handle typos when sending messages.
    - Use an error ring to generate a custom error to indicate that an invalid message was received by that loop.
4. Modify the Message Handling Loop and Boiler Controller Loop to handle shutting down in the event of a critical error.
    - Create a separate Error case in the Message Handling Loop and Boiler Controller Loop.
    - Modify `Dequeue Message.vi` to send you to the Error case instead of the Exit case in the event of an error.
    - Critical errors should be passed as message data from the bottom up (Boiler»Boiler Controller»Message Handling Loop)
    - The Error case of the Message Handling Loop sends you to the Initiate Stop case, as if you clicked Emergency Stop.

Figure 5-1. Errors Passed up to the Message Handling Loop, which then Stops All Loops



Implementation

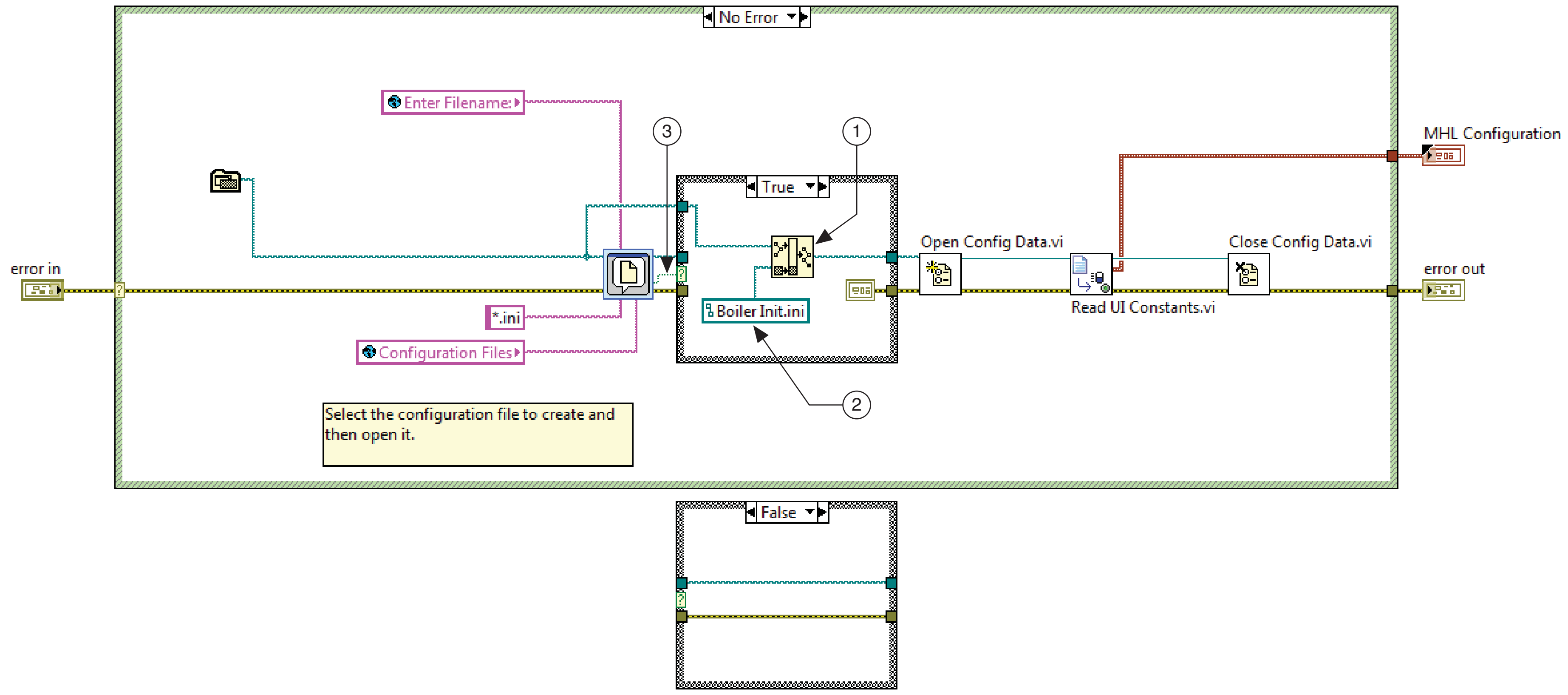
Non-Critical Errors

1. Modify `Read Configuration Data.vi` to handle cancellation of the Open File dialog.

   ☐ Run `Main.vi`. Cancel the Open File dialog where you would normally select the INI file. Notice that the File Dialog express VI generates Error 43 to indicate that the operation was cancelled. This error causes the application to exit. This is *not* a desirable behavior.

   ☐ Modify `Read Configuration Data.vi` to load a default INI file if the user cancels the Open File dialog as shown in Figure 5-2.

Figure 5-2. Read Configuration Data VI Block Diagram Load Default INI File



1	Build Path function
2	Path constant
3	Wire the **cancelled** output of the File Dialog express VI to the case selector.

2. Test the VI.

☐ Run `Main.vi` and cancel the Open File dialog. Notice that the code continues executing normally, using the default file.

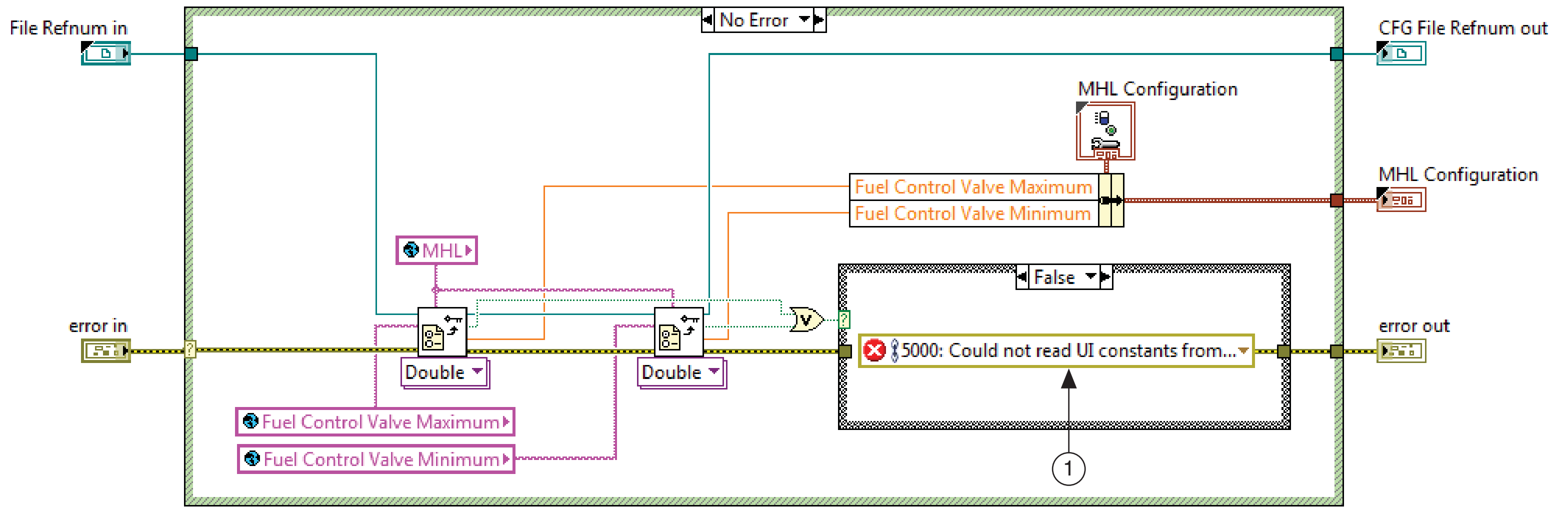
Critical Errors

1. Modify Read UI Constants to handle selection of an invalid INI file.

   ☐ Run `Main.vi`. Select any file other than the INI file that you created. Notice that the range of Fuel Control Valve updated to 0-1 instead of 10-75. This did not generate an error, but the application cannot function without proper configuration of the Fuel Control Valve limits.

   ☐ Modify `Read UI Constants.vi` as shown in Figure 5-3 to check that each key was properly read from the INI file.

Figure 5-3. Read UI Constants Using Error Ring


File Refnum in
No Error
CFG File Refnum out
MHL Configuration
Fuel Control Valve Maximum
Fuel Control Valve Minimum
MHL Configuration
MHL
False
error in
error out
5000: Could not read UI constants from...
Double
Double
Fuel Control Valve Maximum
Fuel Control Valve Minimum
1

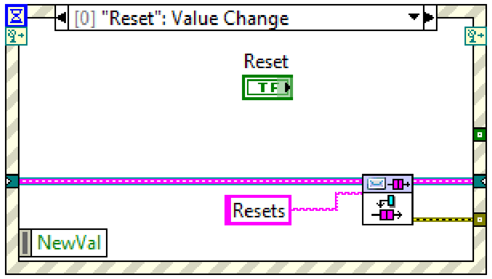
1 Error Ring—Generate a custom error to indicate that the INI file is not valid. This error shuts down the system.

☐ Configure the Error ring with custom error code `5000` and the message `Could not read UI constants from INI file`.

2. Modify the Default case of the Message Handling Loop and Boiler Controller to handle typos when sending messages.

- ☐ Modify the Reset event case of the Event Handling Loop to send the incorrect message, `Resets`, to the Message Handling Loop.

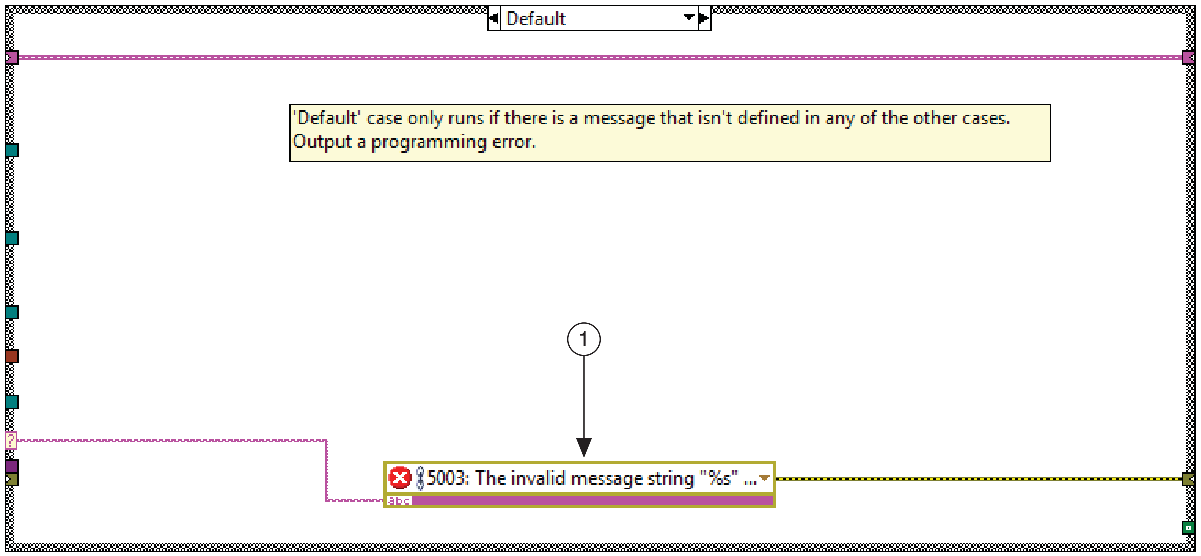
Figure 5-4. Event Handling Loop Reset: Value Change Case with Typo



1 Change the Message string constant input to `Resets`.

☐ Open the Default case of the Message Handling Loop. Notice that the code is already generating an error. Replace the Format into String and Error Cluster From Error Code functions with an error ring, as shown in Figure 5-5. This method of generating an error allows greater flexibility in how you generate the error.

Figure 5-5. Message Handling Loop Default Case with Error Ring

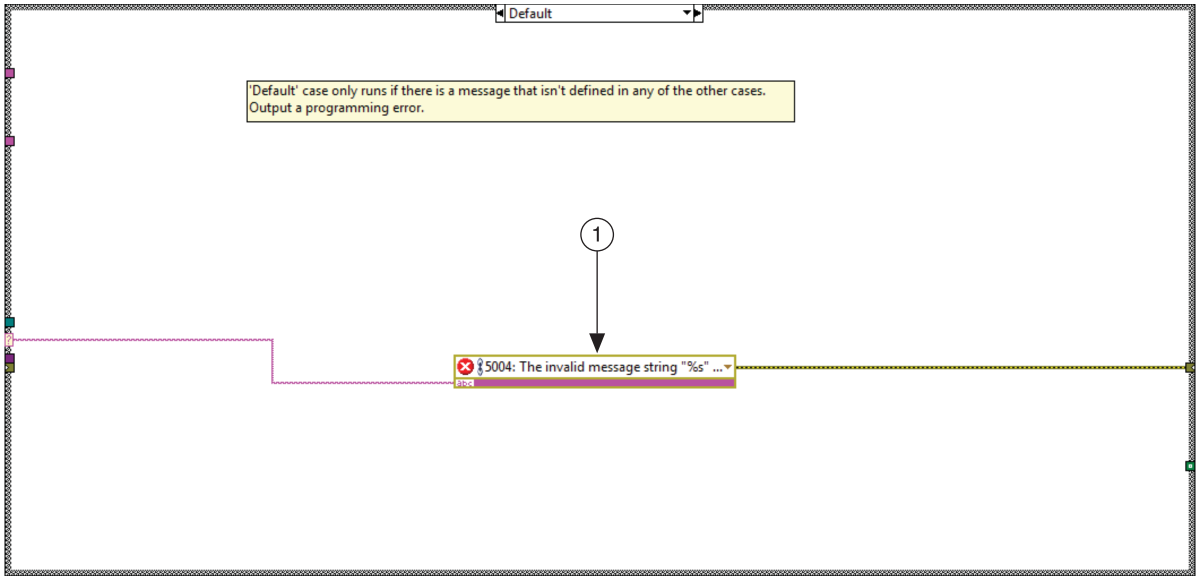


'Default' case only runs if there is a message that isn't defined in any of the other cases.
Output a programming error.

1 Error Ring—Configure the error ring with Custom Error Code `5003` and the message `The invalid message string "%s" was received in the Message Handling Loop`.

3. Modify the Default case of the `Boiler Controller.vi` as shown in Figure 5-6.

Figure 5-6. Boiler Controller VI Default Case with Error Ring


Default
'Default' case only runs if there is a message that isn't defined in any of the other cases.
Output a programming error.
1
5004: The invalid message string "%s" ...

1 Error ring—Configure the error ring with custom error code `5004` and the message `The invalid message string “%s” was received in the Boiler Controller Loop`.

- ☐ Run `Main.vi`. Click **Reset**. Notice that the application hangs because `Dequeue Message.vi` sends the Message Handling Loop directly into the Exit case which causes the Message Handling Loop and Event Handling Loop stop, but no message is sent to the boiler controller to stop it and the boiler.
- ☐ To halt the application, select the Boiler window and press <Ctrl **-.**>.

4. Modify the Message Handling Loop and Boiler Controller Loop to handle shutting down in the event of a critical error.

   □ Create an Error case in the Message Handling Loop and Boiler Controller Loop.

Figure 5-7. Message Handling Loop Error Case

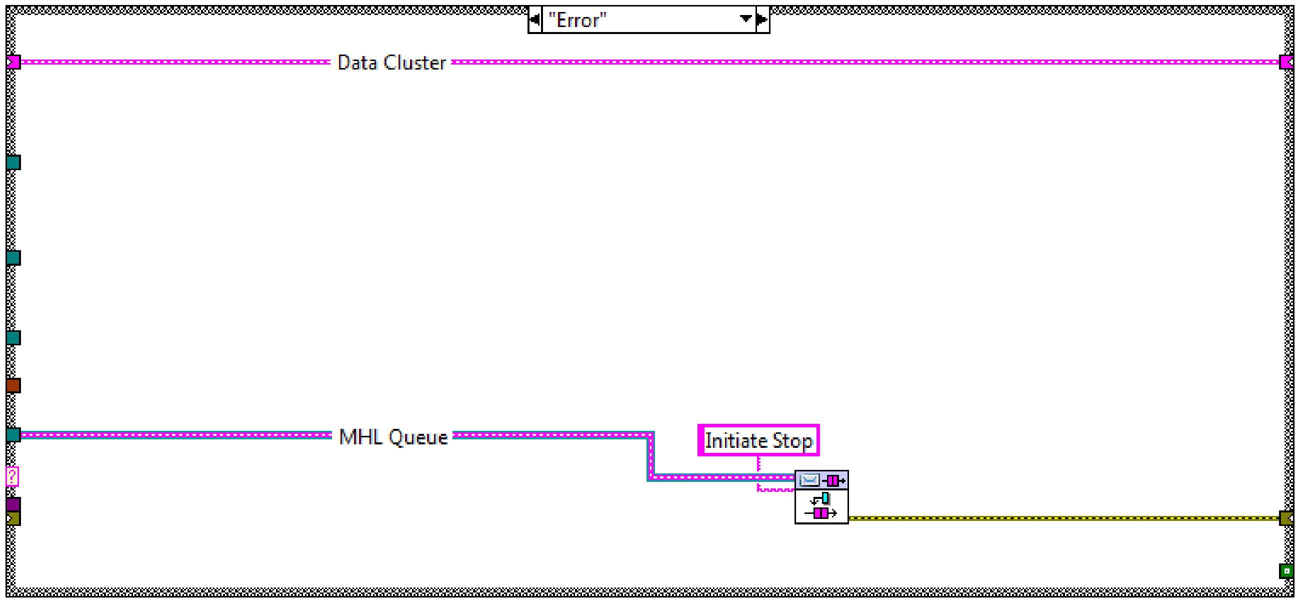
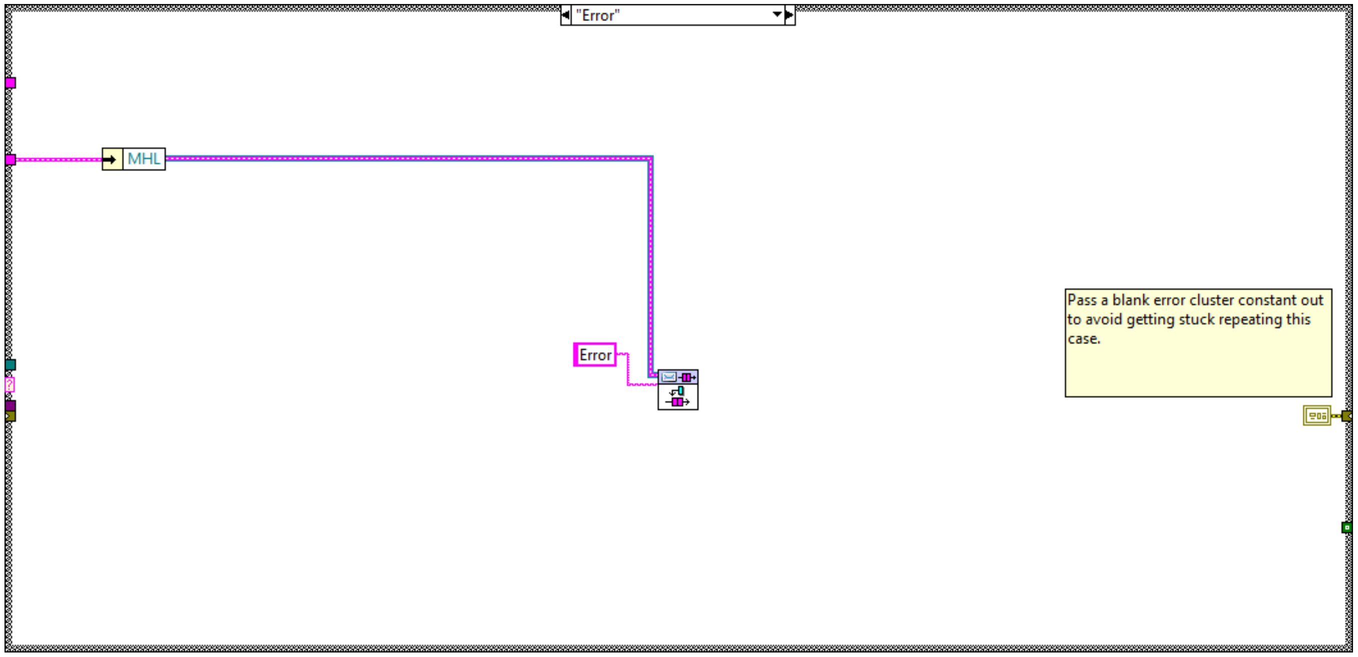


Figure 5-8. Boiler Controller Loop Error Case

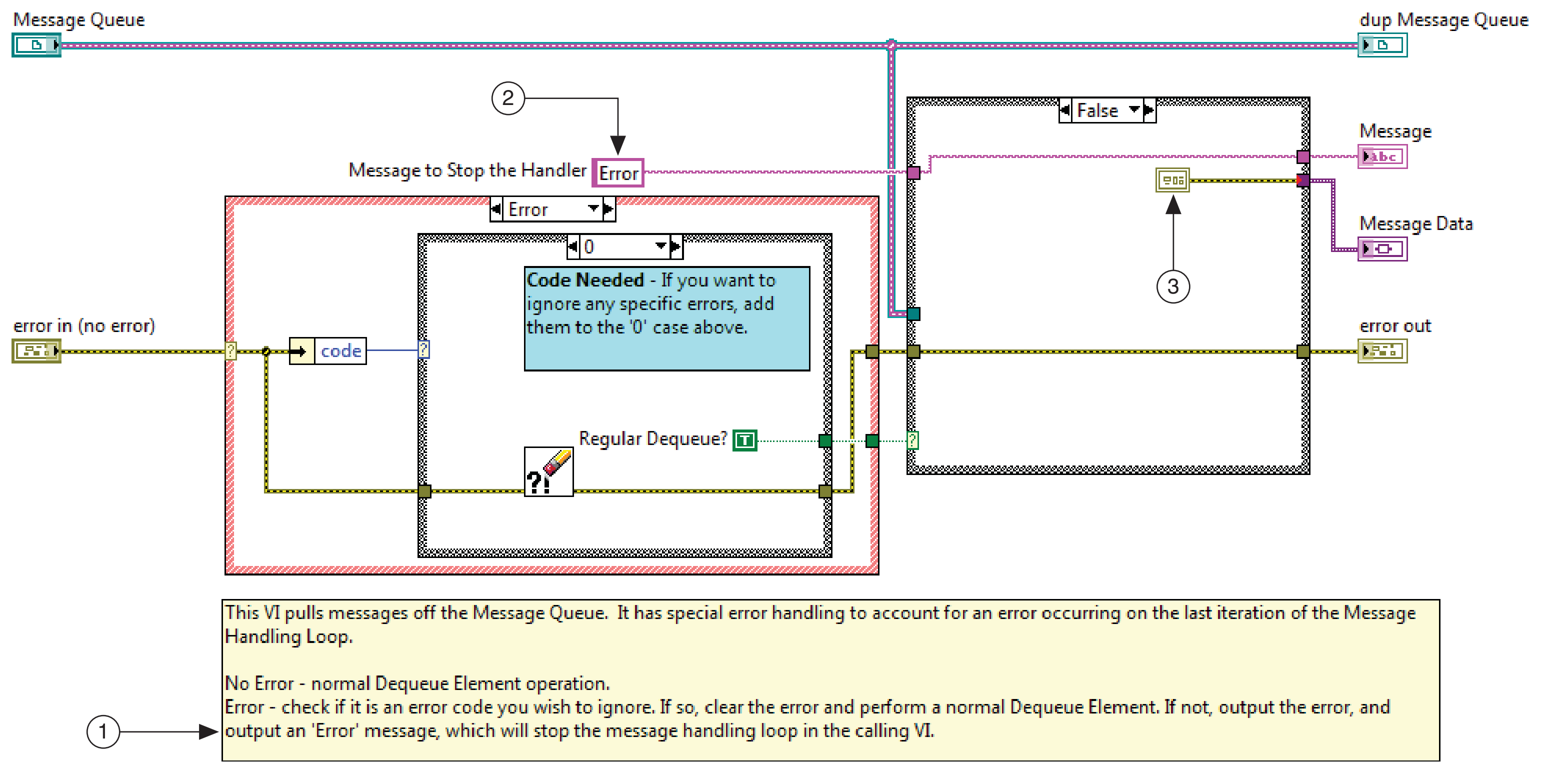


Pass a blank error cluster constant out to avoid getting stuck repeating this case.

5. Modify `Dequeue Message.vi`, as shown in Figure 5-9 to send you to the Error case instead of the Exit case in the event of an error.

Note You must close all open VIs and modify `Dequeue Message.vi` from the LabVIEW project. This VI is reentrant, so you must ensure that there is only one copy in memory before you can edit it.

Figure 5-9. Dequeue Message Block Diagram



This VI pulls messages off the Message Queue. It has special error handling to account for an error occurring on the last iteration of the Message Handling Loop.

No Error - normal Dequeue Element operation.
Error - check if it is an error code you wish to ignore. If so, clear the error and perform a normal Dequeue Element. If not, output the error, and output an 'Error' message, which will stop the message handling loop in the calling VI.

1 Update the comment to say that the output is an Error message instead of an Exit message
2 Update the **Message to Stop the Handler** to `Error`.
3 Error cluster constant

☐ Critical errors should be passed as message data from the bottom up—Boiler»Boiler Controller»Message Handling Loop.

Note Boiler Controller Error sends a message to the Message Handling Loop Error. Errors in the boiler are passed up to the Boiler Controller in the same way.

The Error cases should *not* wire the error cluster into the Enqueue Message. For the Boiler Controller, wire an error cluster constant to the shift register for each loop to avoid an endless error cycle. (The desired effect for the error will happen by launching the shutdown process.)

☐ The Error case of the Message Handling Loop sends you to the Initiate Stop case, as if you clicked Emergency Stop.

Test the Application

1. Test the VI.

   ☐ Run the VI.

   ☐ Verify that:

   – Cancelling the Open File dialog does not stop the application.

   – Specifying an invalid INI file stops the application.

   – All previously-implemented functionality except for Reset still works.

   – Clicking **Reset** generates a critical error that halts the application.

2. Test the build specification.

   ☐ Build the executable from the build specification.

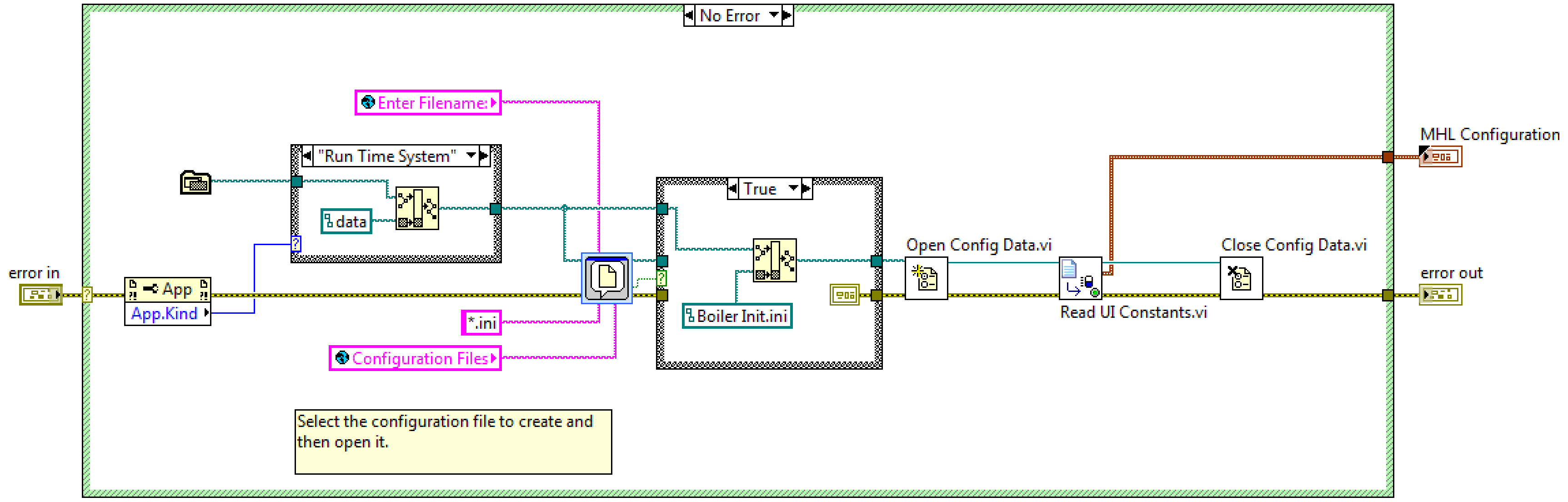
   ☐ Run `Boiler Controller.exe`.

   ☐ Verify the behavior from Step 1.

   – Cancelling the Open File dialog fails because the relative path to the default INI file is different for the EXE—in the data folder, instead of at the same level.

3. Modify `Read Configuration Data.vi`, as shown in Figure 5-10, to use a different relative path for EXEs.

Figure 5-10. Modify Read Configuration Data Block Diagram



4. Fix the forced error.

   ☐ Update the Reset case of the Event Handling Loop to call `Reset` instead of `Resets`.

End of Exercise 5-1

Exercise 5-2 Log Errors

Goal

Log information about any error that causes a system shutdown.

Scenario

In the sprint planning meeting for this iteration, the product owner chose to implement the following user story:

- As a boiler operator, I want information about critical errors to be logged to disk, so that I can identify and troubleshoot the cause of the error at a later date.

As it is currently implemented, critical errors cause the application to shut down without providing any additional information to the end-user.

To implement this user story, your team identifies the following tasks:

1. Create the error log file in the same directory as the main application.
2. When a critical error occurs, log the error code and source.
3. If you cannot write to the application directory, display the error information to the user.
4. After logging the error, close the error log file reference.

Design

The team decides on the following strategies for approaching each task:

1. Create the error log file in the same directory as the main application.
    - The error log should only be created if a critical error actually occurs.
    - Create the error file in the Error case of the Message Handling Loop.
2. When a critical error occurs, log the current time, error code and source.
    - Pass the error cluster as Message Data when you send the Error message.
    - Extract the error code and source from the error cluster.
    - Create a subVI that writes the time, error code, and source to the error log file.

3. If you cannot write to the application directory, display the error information to the user.
    - If the error log reference is invalid, then you cannot write to the file.
    - Instead, launch a one-button dialog and display the information to the user.
4. After logging the error, close the error log file reference.

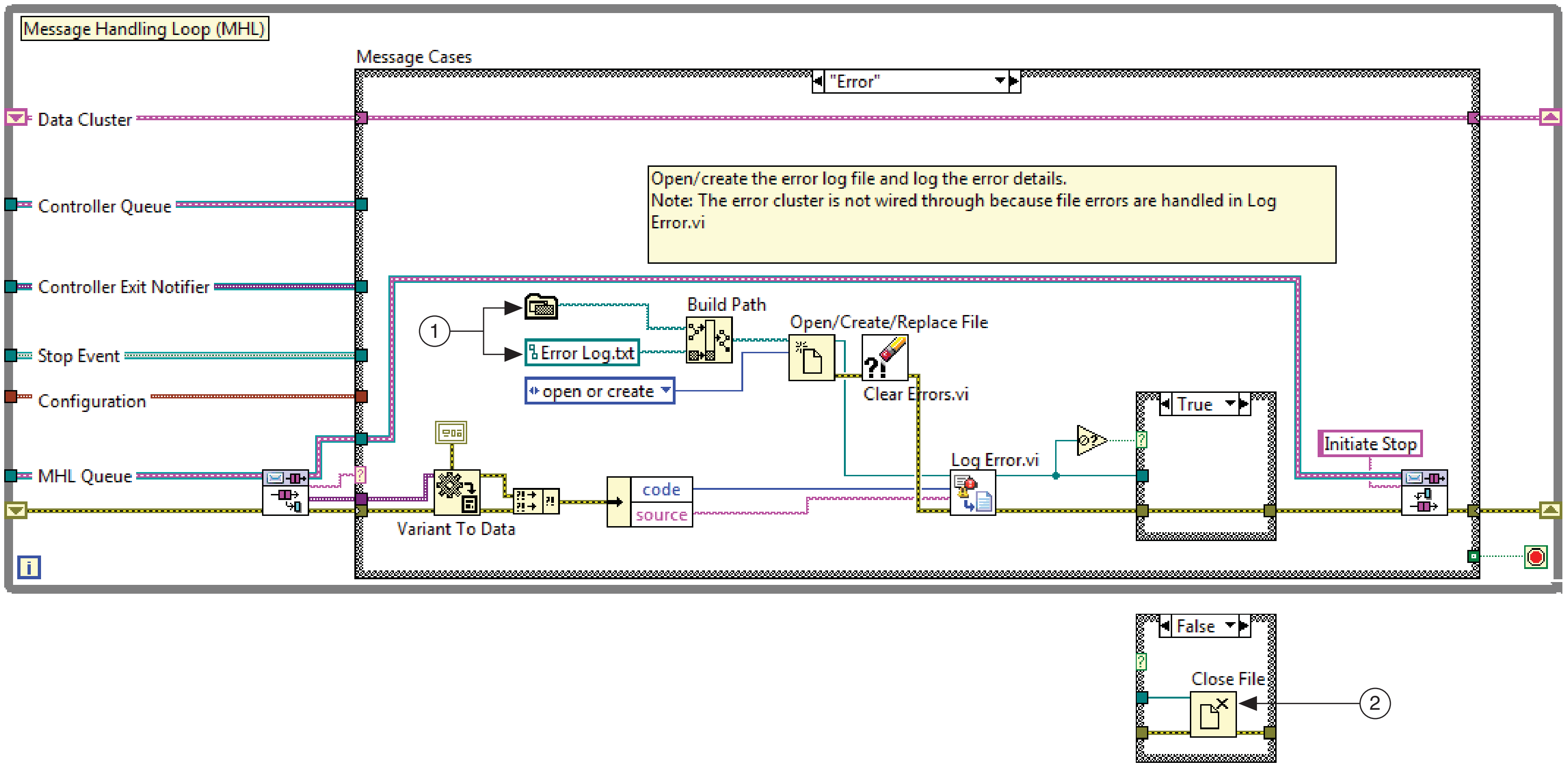
Implementation

1. Add a VI to write the time, error code, and source to the error log file.
    - ☐ Create a virtual folder called `Logging` in **Support VIs** folder the Boiler Controller Project Explorer window.
    - ☐ Navigate to `<Exercises>\LabVIEW Core 3\External\Support VIs\Logging` and copy `Log Error.vi` into the `<Exercises>\LabVIEW Core 3\Course Project\support\Logging` directory.
    - ☐ Add `Log Error.vi` to the **Logging** virtual folder of the Boiler Controller Project Explorer window.

2. If you cannot write to the application directory, display the error information to the user.

☐ Modify the error file in the Error case of the Message Handling Loop as shown in Figure 5-11.

Figure 5-11. Message Handling Loop Error Case



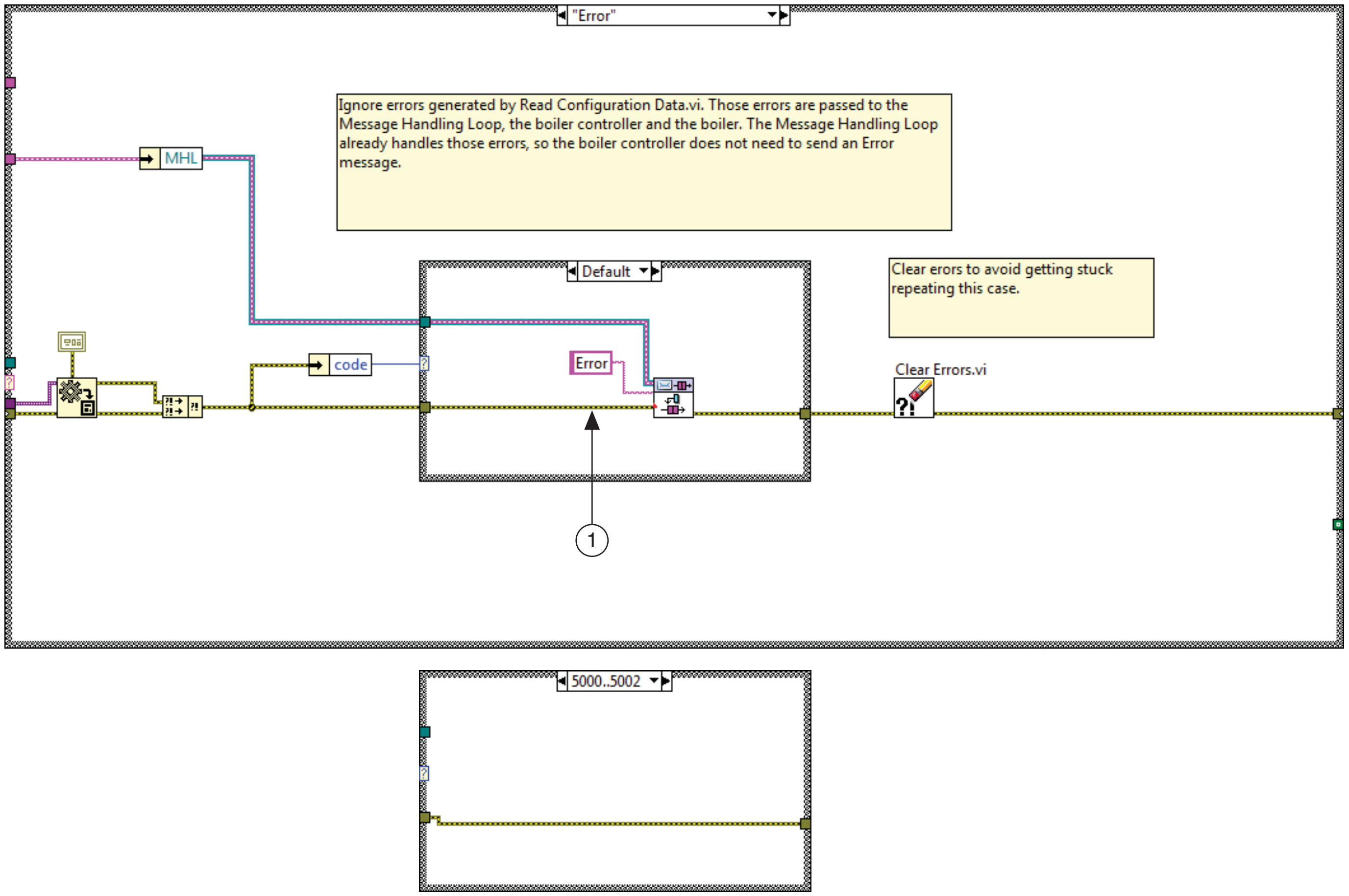
1 Create the error log file in the same directory as the main application.

2 After logging the error, close the error log file reference

3. When a critical error occurs, log the current time, error code and source.

☐ Modify the Error case of `Boiler Controller.vi`, as shown in Figure 5-12. You use the custom errors 5001 and 5002 in future exercises.

Figure 5-12. Boiler Controller Error Case


"Error"
Ignore errors generated by Read Configuration Data.vi. Those errors are passed to the Message Handling Loop, the boiler controller and the boiler. The Message Handling Loop already handles those errors, so the boiler controller does not need to send an Error message.
MHL
Default
Clear erors to avoid getting stuck repeating this case.
code
Error
Clear Errors.vi
1
5000..5002

1 Pass the error cluster as Message Data when you send the Error message.

Test the Application

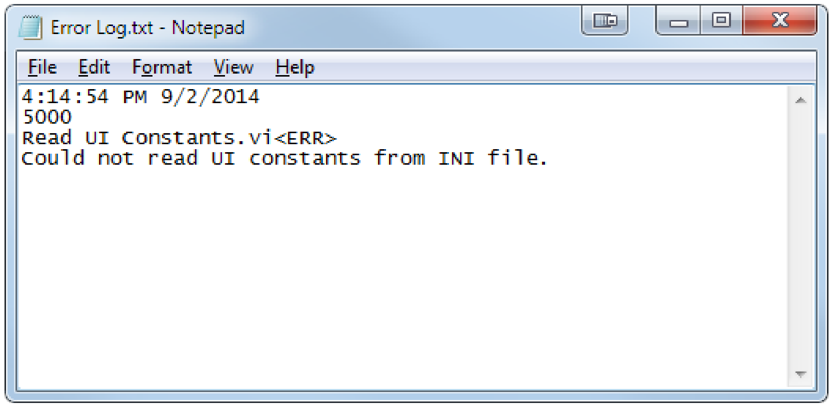
1. Test the VI.

   ☐ Run the VI. Verify the following:

   – Specifying an invalid INI file stops the application and creates `Error Log.txt`.

   – `Error Log.txt` contains time, error code, and error source information similar to Figure 5-13.

Figure 5-13. Error Log.txt



```
4:14:54 PM 9/2/2014
5000
Read UI Constants.vi<ERR>
Could not read UI constants from INI file.
```

   ☐ Modify the Reset event case of the Event Handling Loop to send a **`Resets`** message to the Message Handling Loop.

   – Run the VI again.

   – Clicking Reset halts the application with a critical error.

   – Open `Error Log.txt` and view the new entry to the file.

   – Close the file.

   – Fix the Reset case to send a **`Reset`** message.

   ☐ Verify the behavior when the application cannot write to the log file.

   – Set `Error Log.txt` to read-only.

   – Run the VI.

   - Specify an invalid INI file. A dialog appears indicating the time, error code, and source.
   - Set `Error Log.txt` to be writable.

2. Test the build specification.

   ☐ Build the executable from the build specification.

   ☐ Run `Boiler Controller.exe`.

   ☐ Verify the behavior from Step 1.

   - Specify an invalid INI file. You do not need to change the Reset case.

End of Exercise 5-2

6 Creating Modular Code

Topics

Exercise 6-1 Implement and Test a Data Logging Module

Goal

Log status information to a file to assist with tracing through execution.

Scenario

In the sprint planning meeting for this iteration, the product owner chose to implement the following user stories:

- As a boiler operator, I want the boiler controller to write status information when the boiler transitions between operating states, so that I can review the state information at a later date if an error occurs.
- As a boiler operator, I want to click a button to reset the boiler to a known good state, so that the run interlock (safety control) on the boiler indicates that conditions have been satisfied for the boiler to run safely.
- As a boiler operator, I want the user interface to let me modify only the controls that I am allowed to use at that stage of boiler operation so that I do not accidentally execute any start-up steps out of sequence.

To implement these user stories, the team identifies the following tasks:

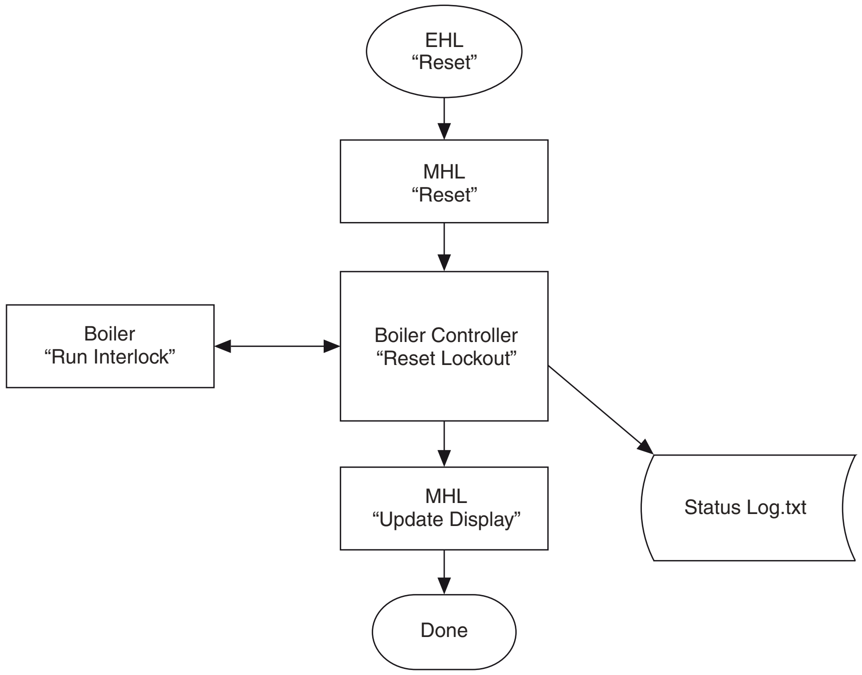
1. Create a code module, `Create Status Log Header.vi`, to write header information for the status log.
2. Create a code module, `Log Status.vi`, to log status information to disk.
3. Develop code to test the functionality of `Create Status Log Header.vi` and `Log Status.vi`.
4. Create a log entry when the Boiler Controller completes initialization.
5. Implement functionality for the **Reset** button. Create a log entry when this operation completes.
6. Disable any controls that are not valid for the current state of the boiler.

Design

The team decides on the following strategies for approaching each task:

1. Create a code module, Create Status Log Header VI, to write header information for the status log. The Create Status Log Header VI must do the following:
   - Create a tab-delimited text file to log data.
   - Write header information to the status log if the log file is empty. Otherwise, the VI passes the refnum through.
2. Create a code module, Log Status VI, to log status information to disk. The Log Status VI must format the data into a string and write that string to `Status Log.txt`.
3. Develop code to verify the functionality of the Create Status Log Header VI and the Log Status VI by creating a VI that opens a file reference, calls the Create Status Log Header VI, calls the Log Status VI, and then exits.
4. Create a log entry when the Boiler Controller VI completes initialization. The Initialize case of the Boiler Controller VI MHL must include the Create Status Log Header VI and the Log Status VI.
5. Implement functionality for the **Reset** button to ensure that the boiler is in a safe state prior to starting it. To implement this functionality, the Boiler Controller VI must do the following:
   - Send a message to the boiler to turn on the run interlock.
   - Send a message to the MHL to update the status of the boiler controller.
   - Create a log entry when this operation completes.
6. Disable any controls that are not valid for the current state of the boiler by modifying the Update Display case of the MHL to include code that enables and disables controls based on the current state of the system.

Figure 6-1. Implementing the Status Log and Reset Functionality



Implementation

Use Code Modules to Write Header Information to the Status Log and Log Status Information to Disk

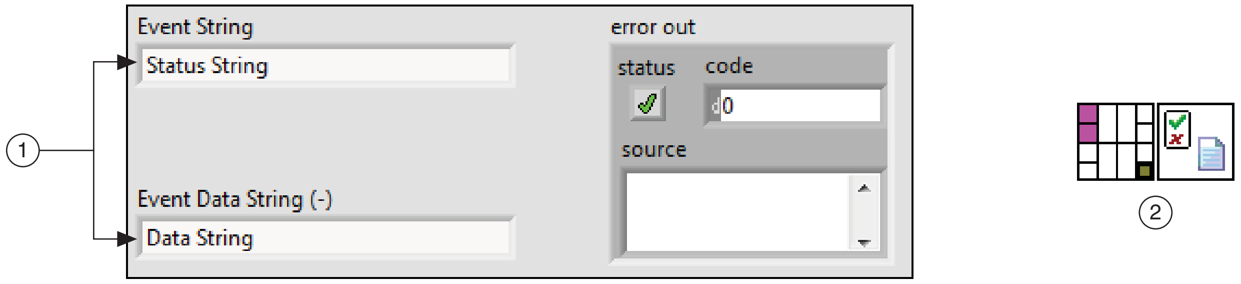
1. In Windows Explorer, copy `Create Status Log Header.vi` and `Log Status.vi` from the `<Exercises>\LabVIEW Core 3\External\Support VIs\Logging` directory and paste them into the `<Exercises>\Course Project\support\Logging` directory.

2. Add `Create Status Log Header.vi`, and `Log Status.vi` to the **Logging** virtual folder in the Boiler Controller Project Explorer window.

Develop Code to Verify the Functionality of the New VIs

1. In the **Project Explorer** window, create a new virtual folder and name it `Test VIs`.
2. In the virtual folder, create a new VI and save the VI to `<Exercises>\LabVIEW Core 3\Course Project\Test VIs\Test - Log Status.vi`.
3. Create the Test - Log Status VI front panel and connector pane as shown in Figure 6-2.

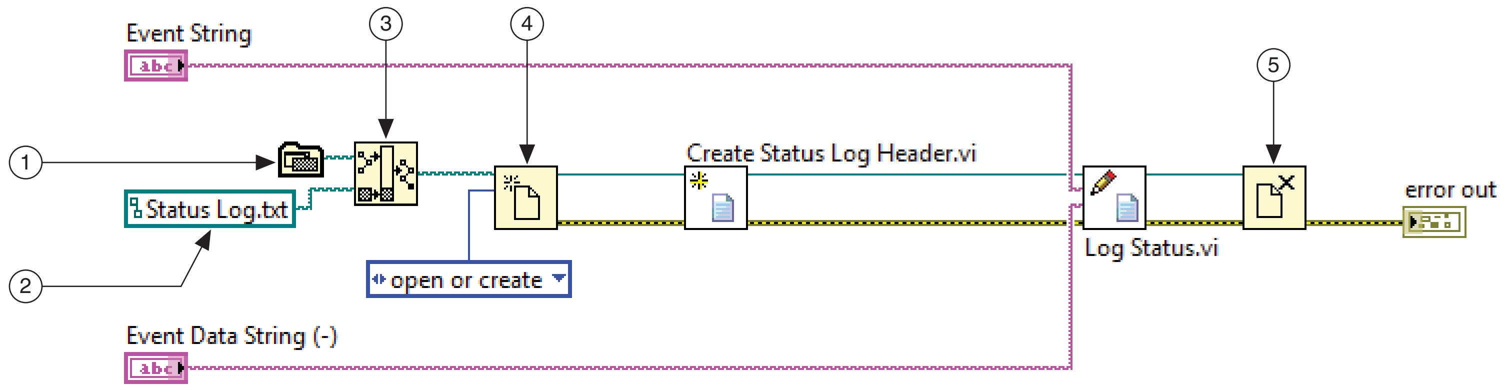
Figure 6-2. Test - Log Status VI Front Panel and Connector Pane



1 **String control**—After entering values in the controls, select **Edit»Make Current Values Default** from the LabVIEW menu.
2 Filter glyphs by keywords, `test` and `file`, to find appropriate glyphs for the icon.

4. Create the Test - Log Status VI block diagram as shown in Figure 6-3.

Figure 6-3. Test - Log Status VI Block Diagram



1 **Application Directory**—Specifies to use the directory of the VI or executable.

2 **Path Constant**—Place a Path constant on the block diagram and enter `Status Log.txt`.

3 **Build Path**—Build the path to the `Status Log.txt` file.

4 **Open/Create/Replace File**—Right-click the **operation (0:open)** input and select **Create»Constant** from the shortcut menu.

5 **Close File**—Closes `Status Log.txt`.

5. Run the Test - Log Status VI and verify that the code modules work properly.

    ☐ Navigate to the `Status Log.txt` that the Test - Log Status VI created.

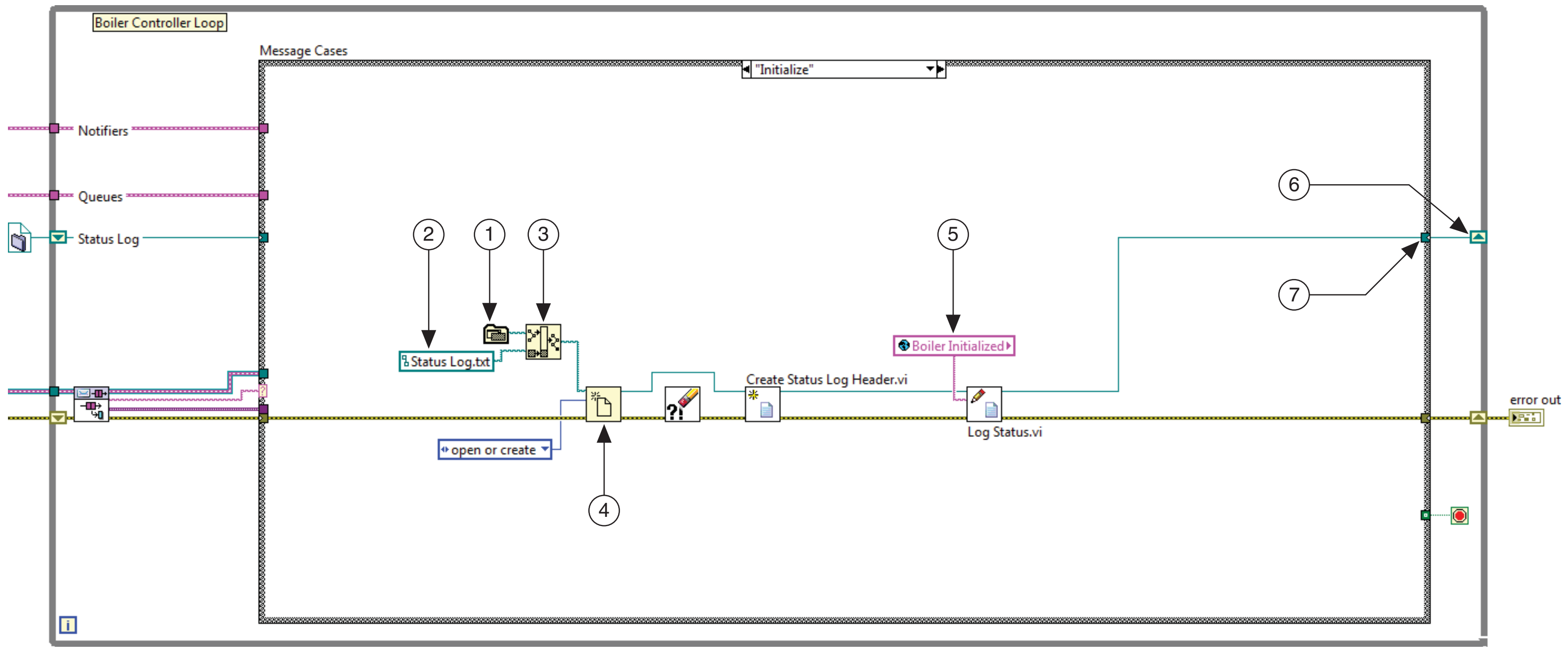
    ☐ View the contents of the log file in Excel.

6. Save and close Test - Log Status VI.

Create a Log Entry After the Boiler Controller Initializes

1. Modify the Initialize case of the Boiler Controller VI as shown in Figure 6-4.

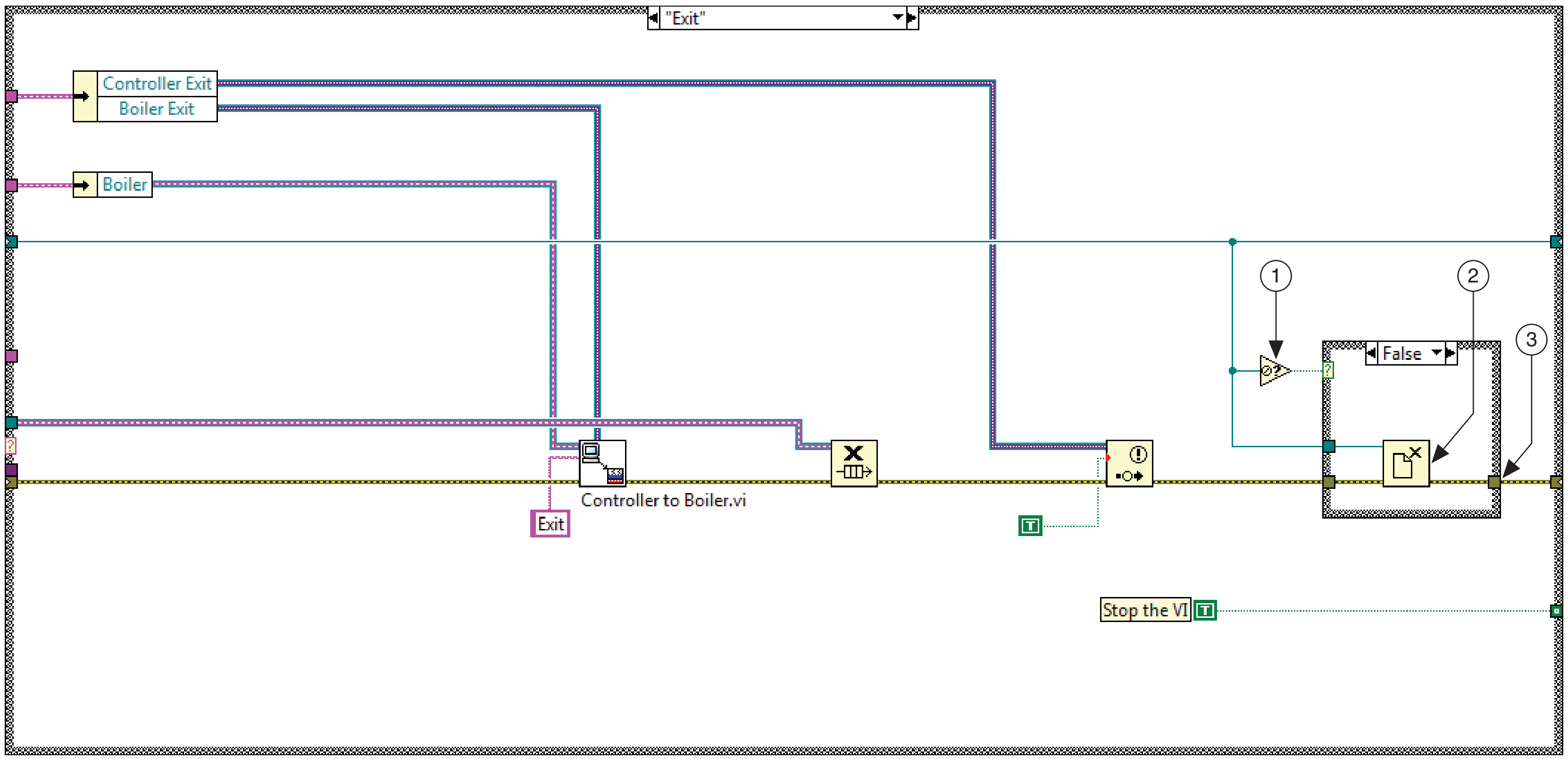
Figure 6-4. Adding Logging to the Initialize Case of the Boiler Controller VI



1	**Application Directory**—Specifies to use the directory of the VI or executable.
2	**Path Constant**—Place a Path constant on the block diagram and enter `Status Log.txt`.
3	**Build Path**—Build the path to the `Status Log.txt` file.
4	**Open/Create/Replace File**—Right-click the **operation (0:open)** input and select **Create»Constant** from the shortcut menu.
5	**Boiler System Globals**—Drag from the **Project Explorer** window and select the appropriate string. Right-click the global and select **Change To Read**.
6	**Shift Register**—After wiring the **Log File Refnum out** output of the Log Status VI to the While loop, right-click the tunnel and select **Replace with Shift Register** from the shortcut menu. Wire the shift register on the left side of the While loop to the Case structure.
7	Right-click the tunnel and select **Linked Input Tunnel»Create & Wire Unwired Cases** from the shortcut menu.

2. Modify the Boiler Controller Exit case to close the log file when the application shuts down, as shown in Figure 6-5.

Figure 6-5. Closing the Log File in the Exit Case



1 **Not A Number/Path/Refnum?**—Makes sure the log file refnum is valid. The file refnum would be invalid only if the log file could not be created or opened.

2 **Close File**—Closes `Status Log.txt`.

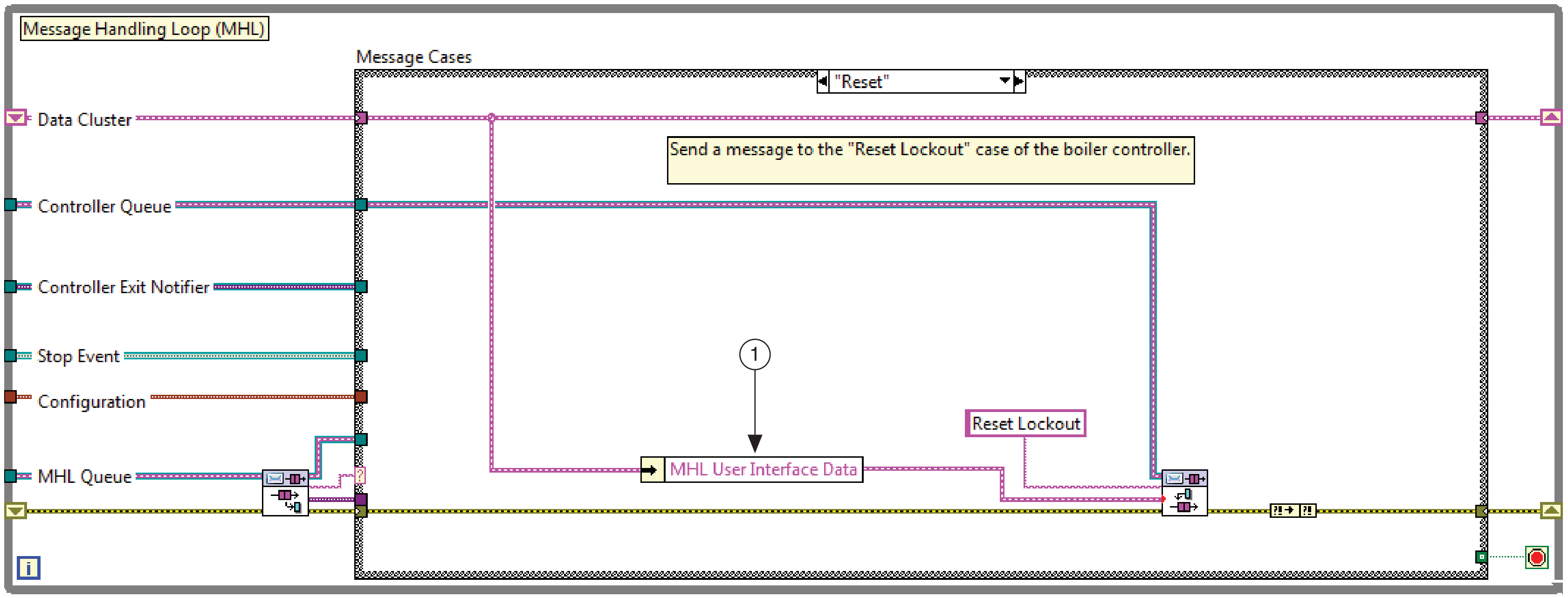
3 Right-click the tunnel and select **Linked Input Tunnel»Create & Wire Unwired Cases** from the shortcut menu.

3. Save Boiler Controller VI.

Implement Functionality for the Reset Button

1. Open **Main.vi** from the **Project Explorer** window and update the Reset case of the MHL as shown in Figure 6-6.

Figure 6-6. Adding Message Data to the Reset Lockout Message

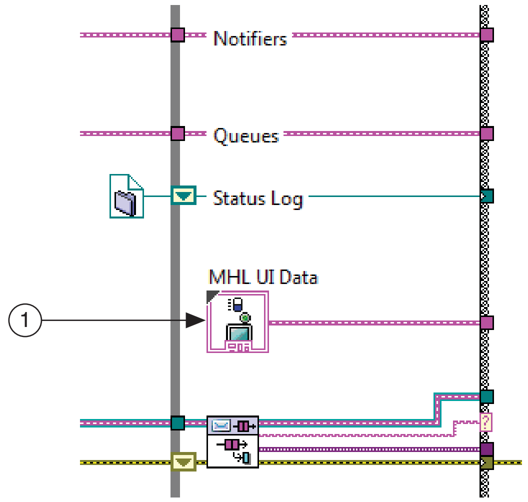


1 **Unbundle By Name**—Unbundle the MHL User Interface Data from the MHL Data wire.

2. Save Main VI.

3. Open the Boiler Controller VI and place an MHL UI Data constant between the While Loop and the Case structure as shown in Figure 6-7.

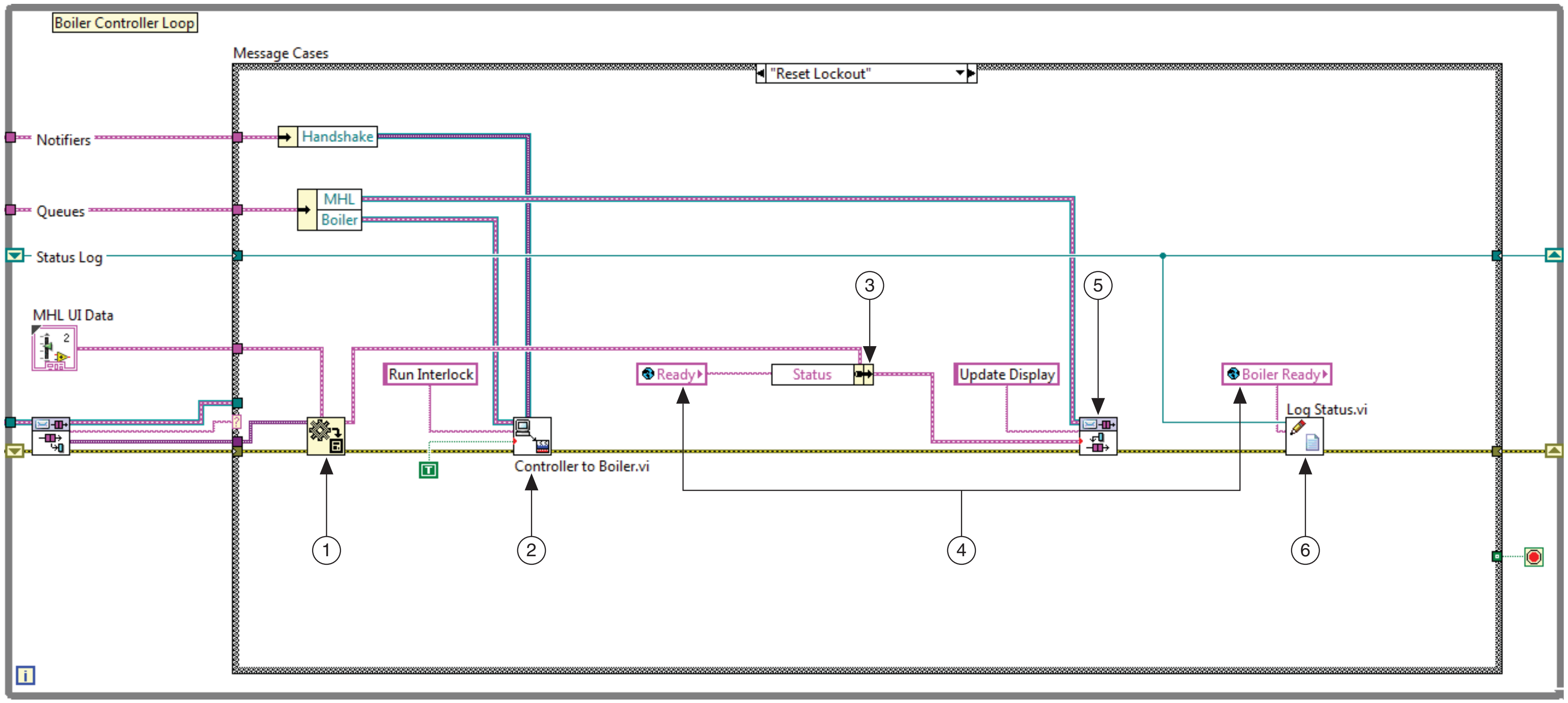
Figure 6-7. Adding MHL UI Data to the Boiler Controller Loop



1 **MHL UI Data constant**—Drag **MHL UI Data.ctl** from the **Project Explorer** window.

4. Update the Reset Lockout case of the Boiler Controller VI as shown in Figure 6-8.

Figure 6-8. Implementing Reset Functionality to the Reset Lockout Case



1 **Variant to Data**—Converts variant data from the Dequeue Message VI to a MHL UI Data type.

2 **Controller to Boiler**—Drag **Controller to Boiler.vi** from the **Support VIs»Notifiers** folder in the **Project Explorer** window. This VI sends a message to the boiler to initiate the run interlock process.

3 **Bundle By Name**—Updates the MHL UI Data.

4 **Boiler System Globals**—Drag from the **Project Explorer** window and select the appropriate string. Right-click the global and select **Change To Read**.

5 **Enqueue Message**—Sends a message to the MHL to update the status of the boiler controller.

6 **Log Status**—Create a log entry when the reset lockout operation is complete.

Disable Controls Depending on the Current State of the Boiler

1. Add files that your teammates developed to the project. Your teammates developed the following two files:

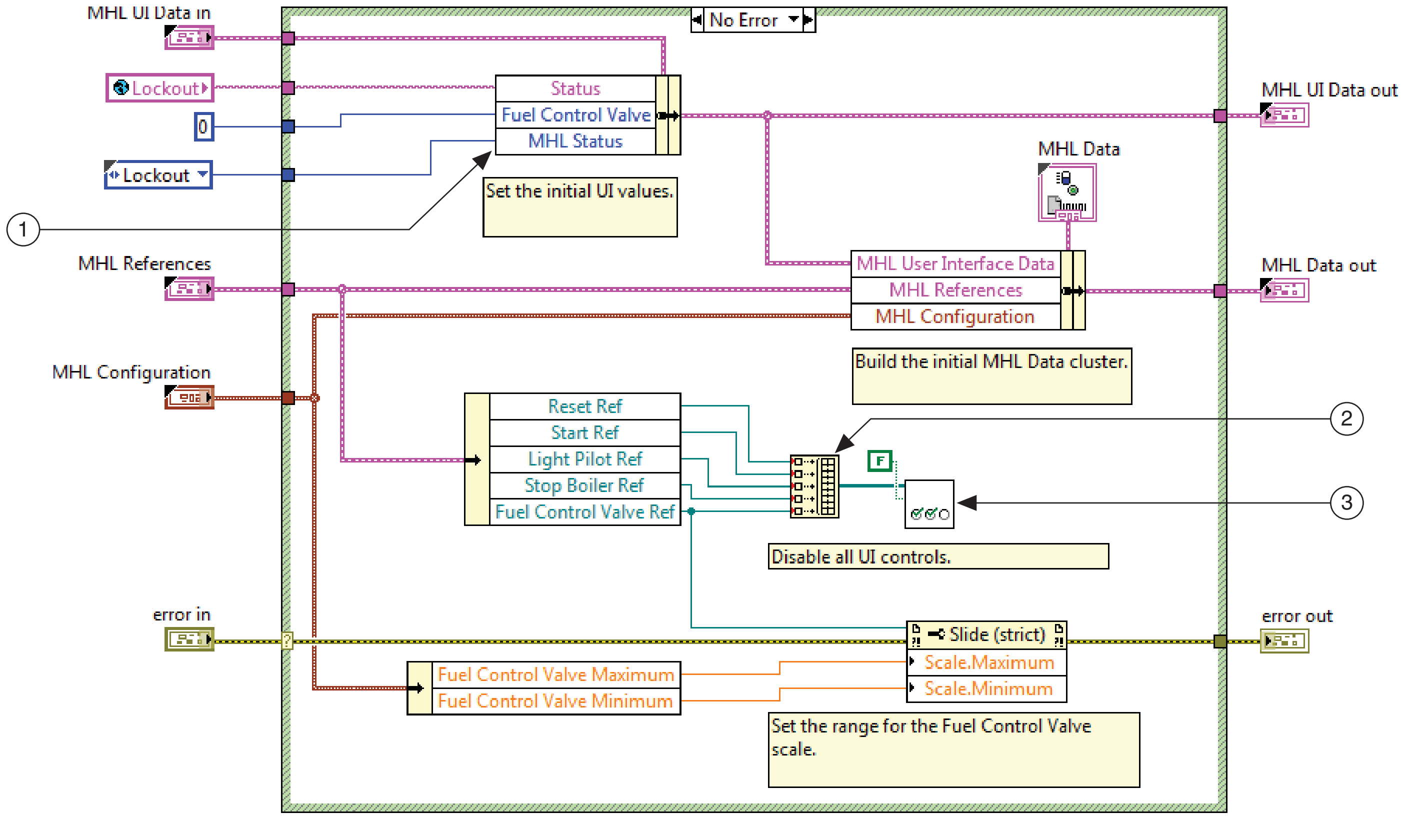
   - `Set Enable State on Multiple Controls.vi`—The VI enables and disables controls using an array of references to those controls.
   - `MHL Status.ctl`—This enum is necessary to make decisions based on the current state of the boiler.

   ☐ Copy `Set Enable State on Multiple Controls.vi` from `<Exercises>\LabVIEW Core 3\External\Support VIs` to `<Exercises>\LabVIEW Core 3\Course Project\support`.

   ☐ Add `Set Enable State on Multiple Controls.vi` to the **Support VIs** folder in the **Project Explorer**.

2. Open and modify the Initialize Panel VI to set the initial state of the boiler and disable all controls by default, as shown in Figure 6-9.

Figure 6-9. Updating the Initialize Panel VI to Disable All Controls



1 Expand the Bundle By Name function to display the **MHL Status** terminal. Right-click the terminal select **Create»Constant** from the shortcut menu. The MHL Status sets the initial state of the boiler.

2 **Build Array**—Builds an array of all the UI control references from the Unbundle By Name function.

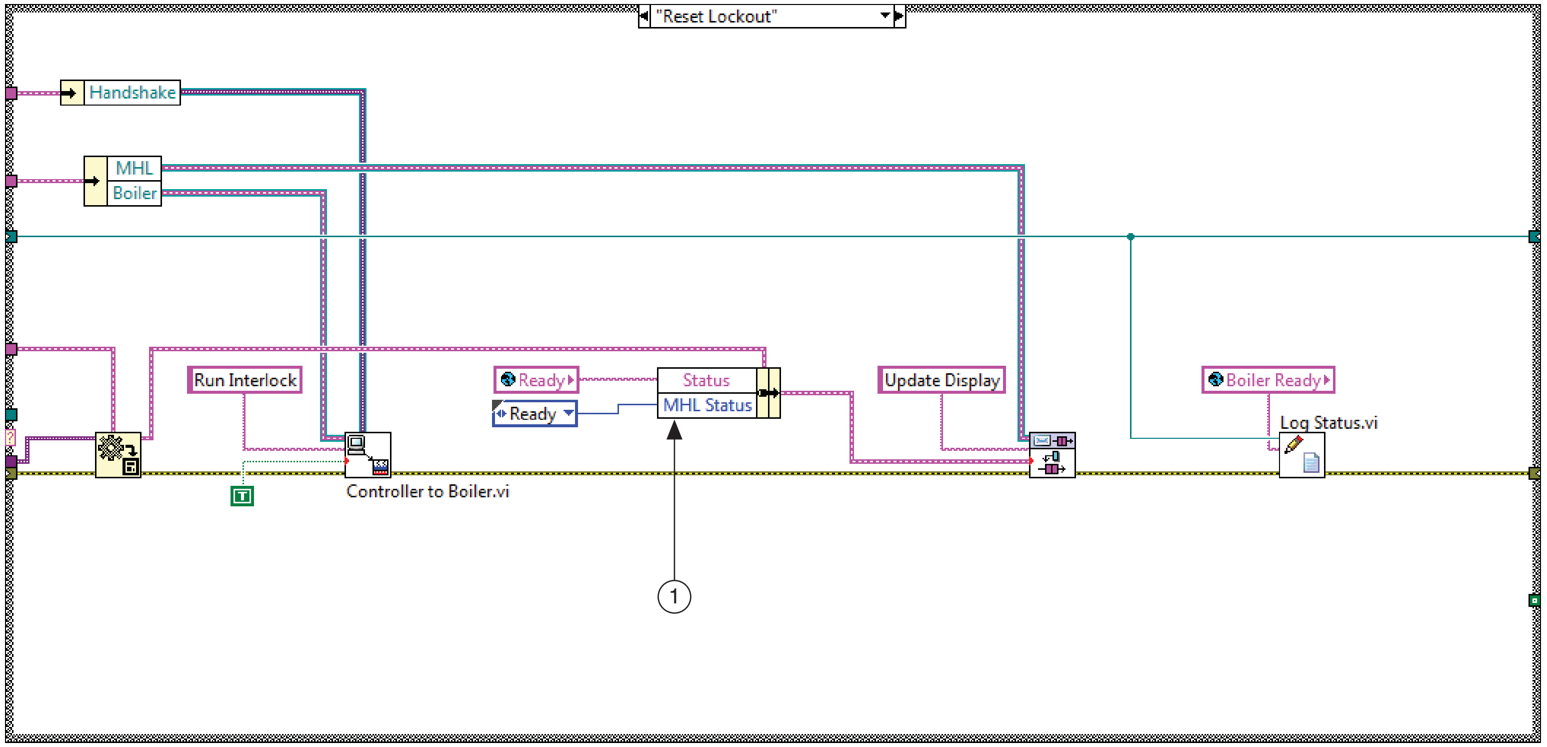
3 **Set Enable State on Multiple Controls**—Drag this VI from the **Project Explorer** window. Wire a False constant to the **Enable?** input to disable all controls.

3. Save and close the Initialize Panel VI.

4. Open the Boiler Controller VI block diagram.

5. Modify the Reset Lockout case of the Boiler Controller VI to set the MHL Status to **Ready** as shown in Figure 6-10.

Figure 6-10. Adding MHL Status to the Reset Lockout Case



1 Right-click the **MHL Status** terminal and select **Create»Constant** from the shortcut menu.

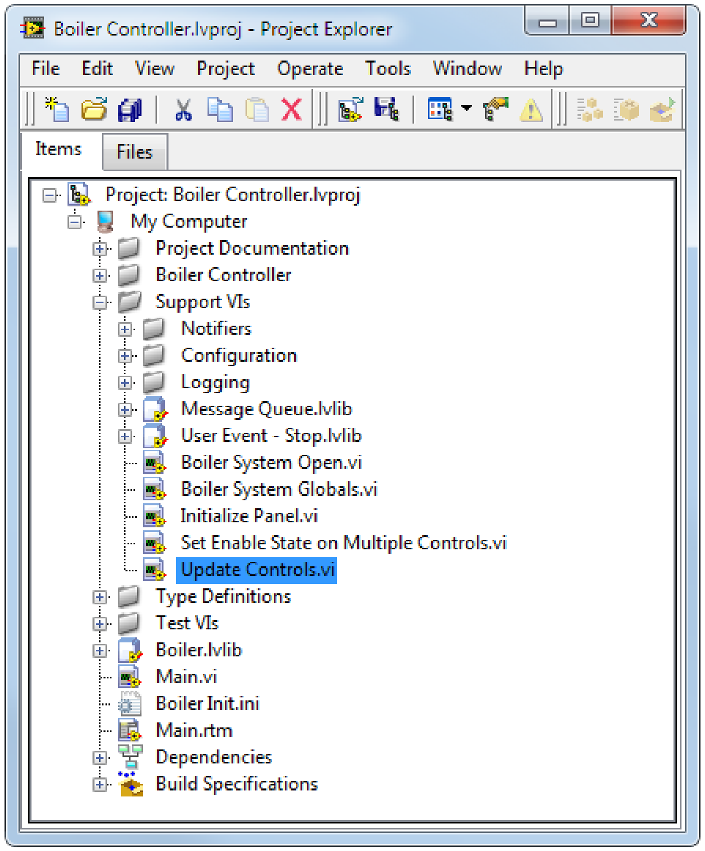
6. Save the Boiler Controller VI.

7. Add files to the project to update the front panel controls.

   ☐ Copy `Update Controls.vi` from `<Exercises>\LabVIEW Core 3\External\Support VIs` to `<Exercises>\LabVIEW Core 3\Course Project\support`.

   ☐ Add `Update Controls.vi` to the **Support VIs** folder in the **Project Explorer** window, as shown in Figure 6-11.

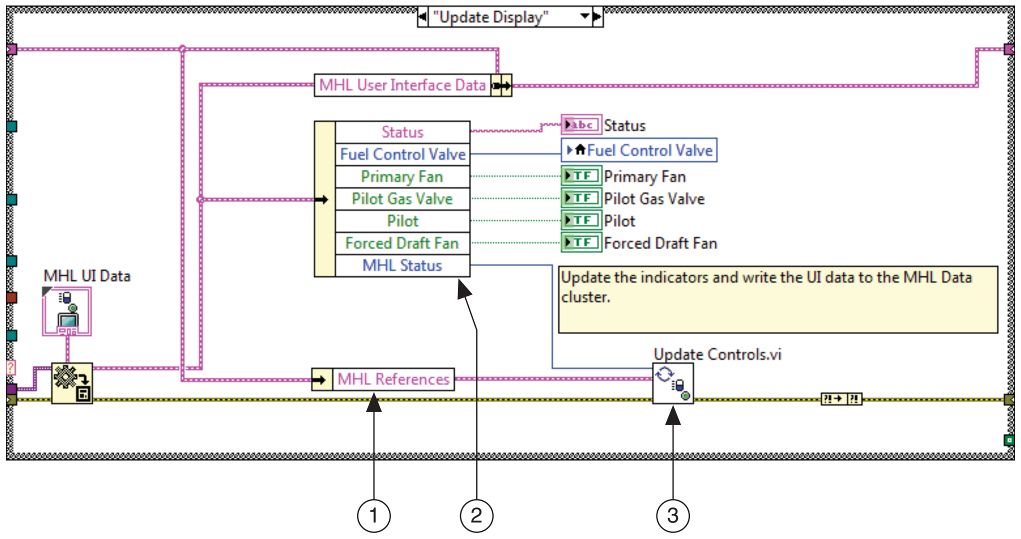
Figure 6-11. Adding Update Controls VI to the Project


Boiler Controller.lvproj - Project Explorer
File Edit View Project Operate Tools Window Help
Items Files

```
Project: Boiler Controller.lvproj
  My Computer
    Project Documentation
    Boiler Controller
    Support VIs
      Notifiers
      Configuration
      Logging
      Message Queue.lvlib
      User Event - Stop.lvlib
      Boiler System Open.vi
      Boiler System Globals.vi
      Initialize Panel.vi
      Set Enable State on Multiple Controls.vi
      Update Controls.vi
    Type Definitions
    Test VIs
    Boiler.lvlib
    Main.vi
    Boiler Init.ini
    Main.rtm
    Dependencies
    Build Specifications
```

8. Modify the Update Display case of the MHL, as shown in Figure 6-12, to include code that enables and disables controls based on the current state of the system.

Figure 6-12. Enabling and Disabling Controls in the Update Display Case



1	**Unbundle By Name**—Right-click and choose **Select Item»MHL References»All Elements** from the menu.
2	Expand the Unbundle By Name function to display the **MHL Status** terminal.
3	**Update Controls**—Wire the MHL Status terminal to the Update Controls VI.

9. Save the Main VI.

Test the Application

1. Test the Main VI.

   ☐ Run the VI.

   ☐ Verify the following:

   – The **Boiler Controller - Initialize** dialog box does not appear.

   – All controls except for **Reset** and **Emergency Stop** are disabled.

- ☐ Click the **Reset** button and verify the following.
  - – A dialog box does not appear.
  - – The **Reset** button is disabled and the **Start** button is enabled.
  - – The **Run Interlock** indicator on the Boiler VI front panel is illuminated. The indicator illuminates because the Controller to Boiler VI sent a `Run Interlock` message to the Boiler VI.
- ☐ Click the **Start** button. The **Boiler Controller - Pre-Purge** dialog box still appears because you have not implemented any other functionality yet.
- ☐ Click the **Emergency Stop** button.
- ☐ Open `Status Log.txt` and verify that the file contains two entries: `Boiler Initialized` and `Boiler Ready`.

2. Test the executable.

- ☐ Build the executable from the build specification.
- ☐ Run `Boiler Controller.exe`.
- ☐ Verify the behavior from step 1.

End of Exercise 6-1

Exercise 6-2 Integrate and Test Modules from the Top-Down

Goal
Perform top-down integration testing for the Start button functionality.

Scenario
In the sprint planning meeting for this iteration, the product owner chose to implement the following user story:

- As a boiler operator, I want to click a button to execute the pre-purge process, which runs the primary fan for a pre-set period of time to purge gases from the combustion chamber prior to lighting the pilot, so that the system does not explode.

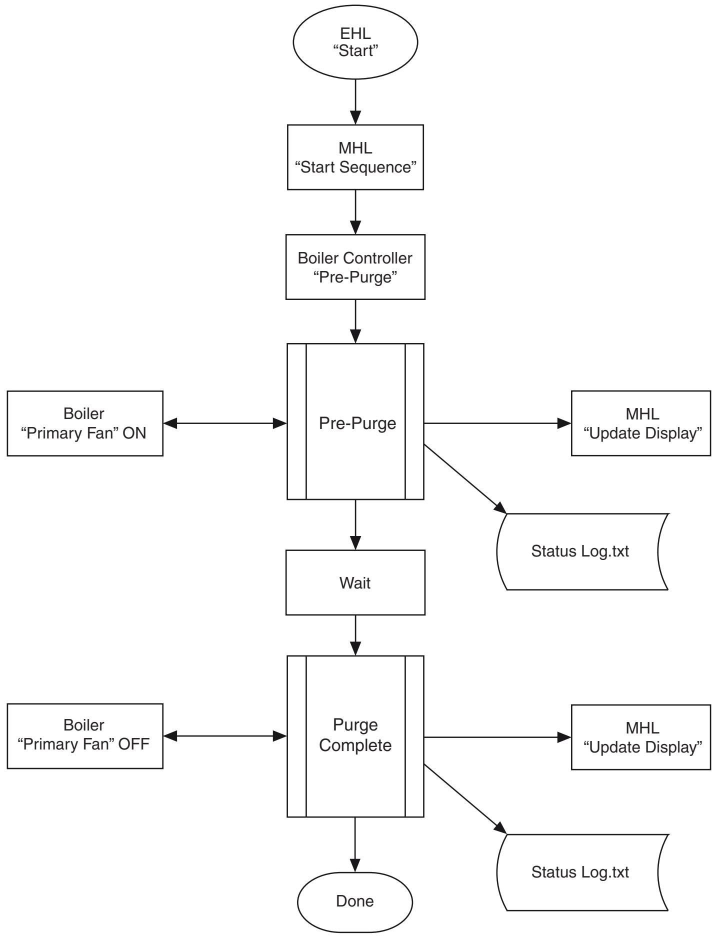
To implement this user story, the team identifies the following tasks:

1. Create code modules to execute the state functionality.
2. Create code to test each module.
3. Integrate and test each code module with the Boiler Controller system.

Design
The team decides on the following strategies for approaching each task:

1. Create code modules to execute the state functionality.
    - Several of your teammates will work with an expert on this type of boiler to implement code modules to handle the bulk of the communication between the boiler controller and the boiler.
    - They will provide you with a project library that contains all of the code modules that you will need to test and integrate with the boiler controller.
2. Integrate each code module with the Boiler Controller system.
    - The team has decided to use top-down integration testing to identify any control or communication issues that may occur early on.
    - Your role will be to create stub code that you will replace as you test and integrate the code modules.

Figure 6-13. Implementing the Pre-Purge Process



Implementation

Create Code Modules to Execute the State Functionality

1. Your teammate provides you with a project library, `Controller.lvlib`.

   - ☐ Move this library and its files from `<Exercises>\LabVIEW Core 3\External\Controller` to `<Exercises>\LabVIEW Core 3\Course Project\Controller`.

2. Add the project library to the top level of the Boiler Controller project.

3. Add `Boiler Controller.vi` to `Controller.lvlib`.

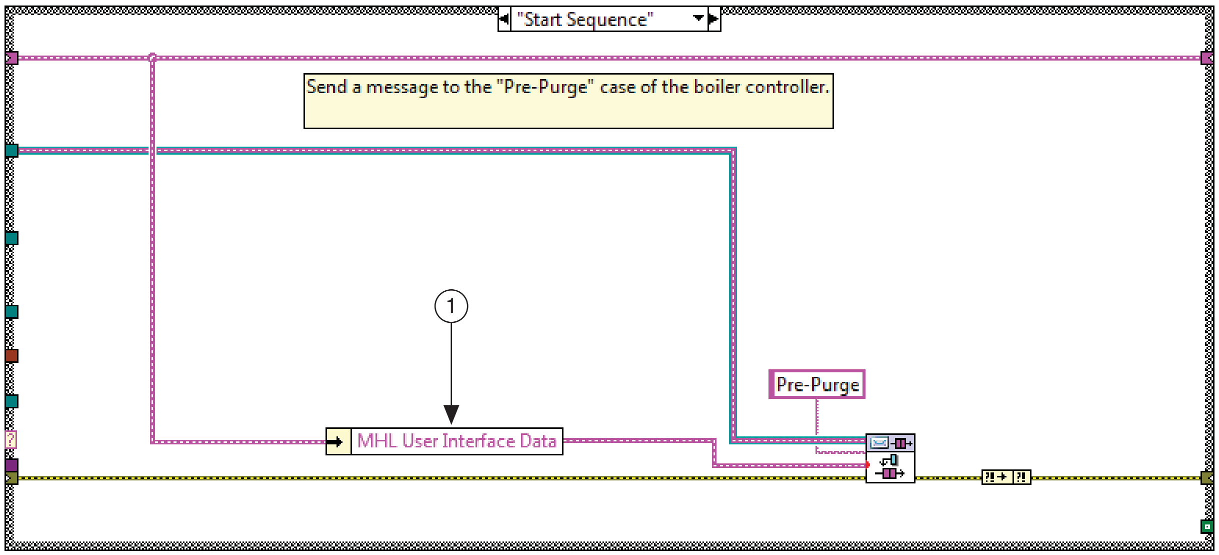
4. Remove the empty Boiler Controller virtual folder from the project.

5. Save `Main.vi`, the project, and the project library.

Integrate and Test Each Code Module with the Boiler Controller System

1. Modify the Start Sequence case of the MHL in `Main.vi` as shown in Figure 6-14.

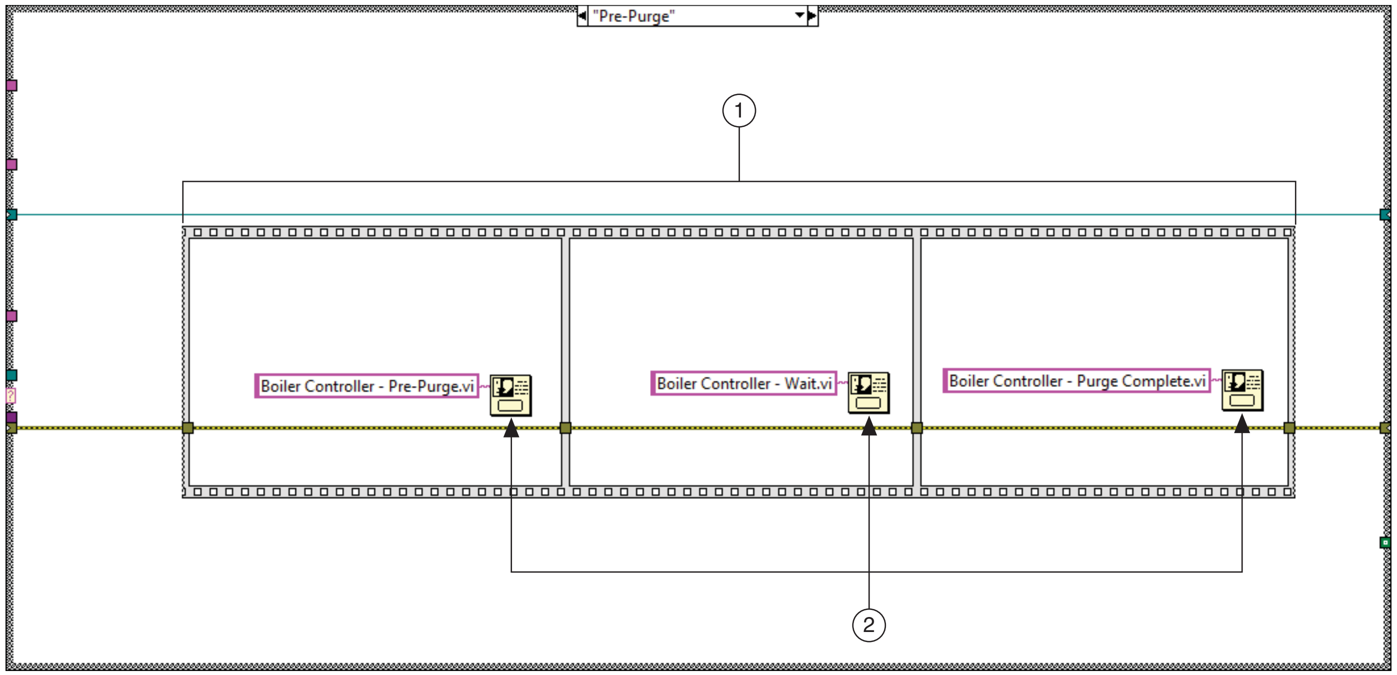
Figure 6-14. Main VI Message Handling Loop Start Sequence Case


"Start Sequence"
Send a message to the "Pre-Purge" case of the boiler controller.
1
Pre-Purge
MHL User Interface Data

1 Pass the MHL User Interface Data as the message data for the Pre-Purge message.

2. Create stub code for the Pre-Purge case of `Boiler Controller.vi`, as shown in Figure 6-15, to test the functionality.

Figure 6-15. Boiler Controller VI Pre-Purge Case With Stub Code


"Pre-Purge"
1
Boiler Controller - Pre-Purge.vi
Boiler Controller - Wait.vi
Boiler Controller - Purge Complete.vi
2

1 Flat Sequence Structure—Add a three-frame sequence structure to the Pre-Purge case of the Boiler Controller VI.

- ☐ Right-click the edge of the sequence structure and select Add Frame After to add a frame.

2 One Button Dialog function—Add a One Button Dialog function to each frame and wire the messages as shown.

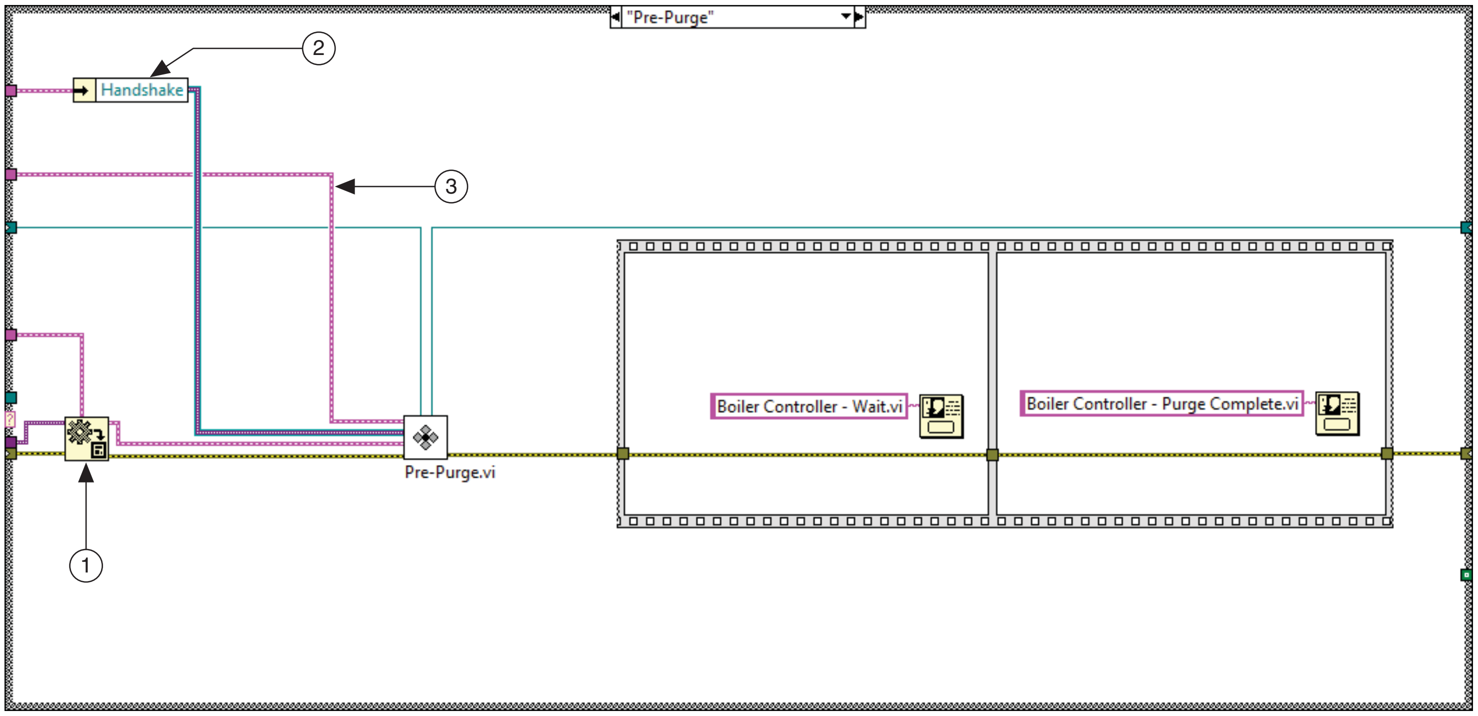
Note The sequence structure frames are placeholders until you are ready to integrate each module. You delete these frames as you integrate each code module into `Boiler Controller.vi`.

3. Run `Main.vi`.

   ☐ Click the **Start** button. Each of the three dialog boxes launch in order. This shows that the top-level code is reaching the stub code that you created.

4. Integrate Pre-Purge VI in the Boiler Controller VI, as shown in Figure 6-16.

Figure 6-16. Boiler Controller VI with Pre-Purge VI Integrated


"Pre-Purge"
Handshake
Pre-Purge.vi
Boiler Controller - Wait.vi
Boiler Controller - Purge Complete.vi
1
2
3

1 Variant to Data function

2 Unbundle by Name—Unbundle the Handshake notifier

3 Message Queues

   ☐ Save the Boiler Controller VI.

5. Test the Pre-Purge VI.

   ☐ Save and run `Main.vi`.

   – Verify that previous functionality still works.

   ☐ Click **Start** and verify the following changes.

   – **Status** displays **`Pre-Purge`**.

   – Primary Fan turns on for both `Main.vi` and `Boiler.vi`.

   – The Boiler Controller - Wait.vi dialog box appears.

   – Click **OK**.

   – The Boiler Controller - Purge Complete.vi dialog box appears.

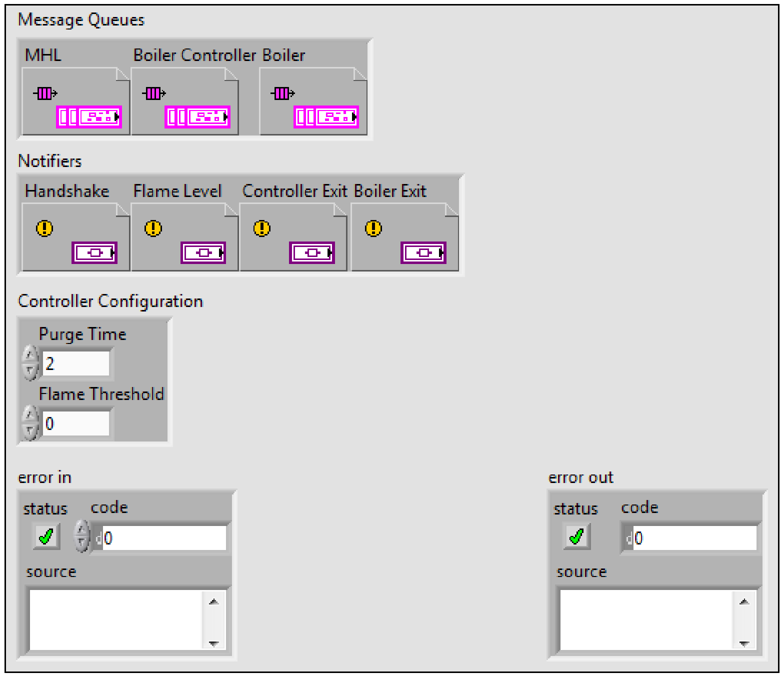
   – Click **OK**.

   ☐ Click **Emergency Stop**.

   ☐ Open `Status Log.txt` and notice the new **`Start Pre-Purge`** entry.

6. Add the Controller Configuration type def to the Boiler Controller front panel as shown in Figure 6-17.

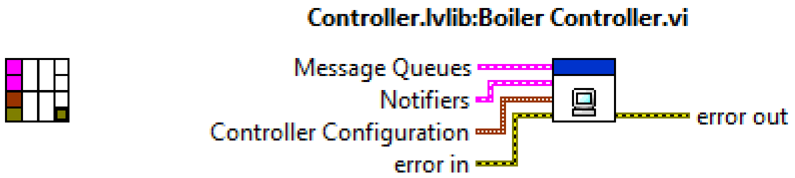
Figure 6-17. Boiler Controller Front Panel with Controller Configuration



1 Controller Configuration Type Def—Add this control from the Controller.lvlib.

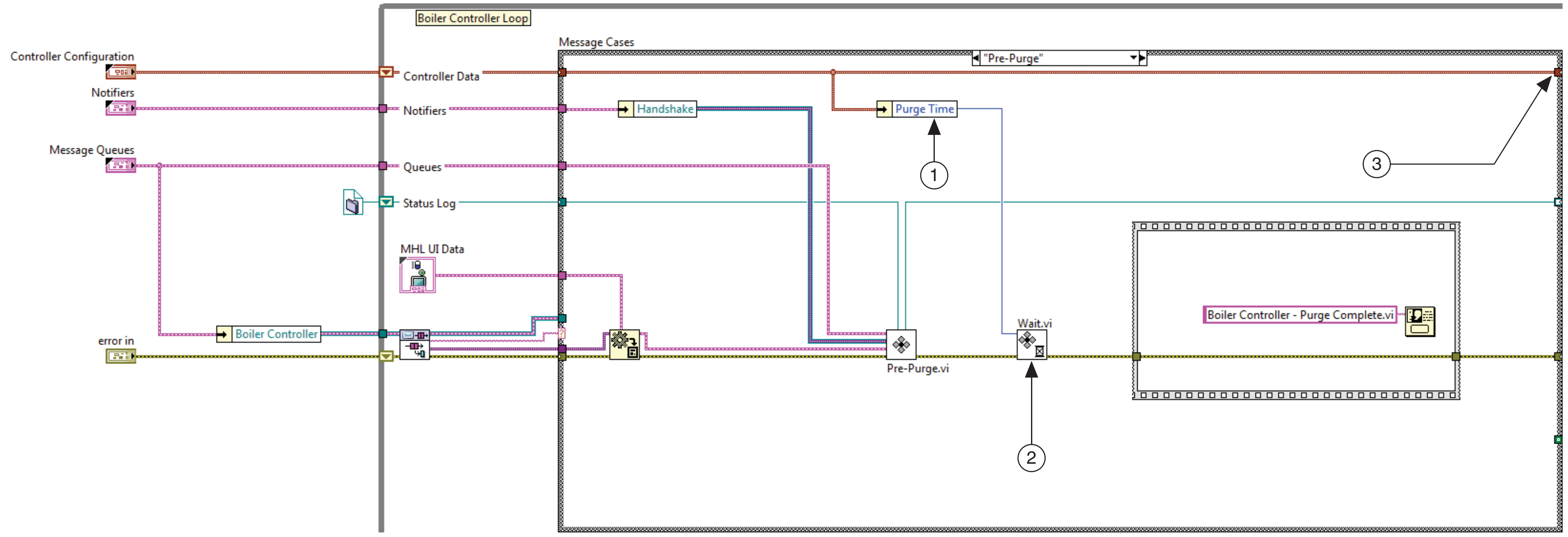
7. Modify the connector pane to include the Controller Configuration as shown in Figure 6-18.

Figure 6-18. Boiler Controller VI Icon and Connector Pane



8. Integrate the Wait VI as shown in Figure 6-19.

Figure 6-19. Boiler Controller VI with Wait VI Integrated



1 Purge Time—Extract Purge Time from the Controller Configuration and wire it to the Wait VI.

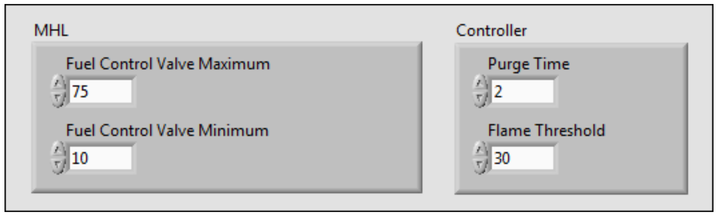
2 Wait VI—This VI uses the Purge Time to determine how long to wait.

3 Wire unwired cases—Right-click the tunnel and select **Linked Input Tunnel»Create & Wire Unwired Cases**. Select the Controller Data tunnel on the left side of the Case structure as the linked tunnel.

9. Initialize the Purge Time value of Controller Configuration.ctl

 ☐ Verify the default values for the Write Configuration Settings File VI, as shown in Figure 6-20.

Figure 6-20. Write Configuration Settings File Default Values



Fuel Control Valve Maximum: 75
Fuel Control Valve Minimum: 10
Purge Time: 2
Flame Threshold: 30

 ☐ Run the Write Configuration Settings File VI to overwrite Boiler Init.ini with the values you specified.

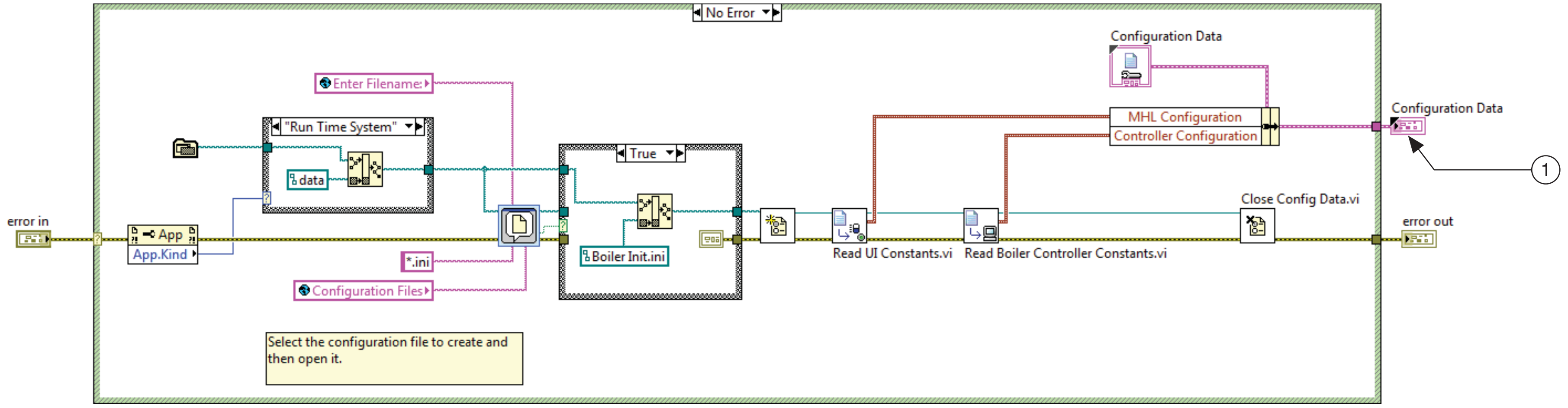
10. Modify `Read Configuration Data.vi` to read the Purge Time from the INI file.

 ☐ Navigate to `<Exercises>\LabVIEW Core 3\Course Project\support\Configuration`.

 ☐ Add `Read Boiler Controller Constants.vi` and `Configuration Data.ctl` to the **Support VIs»Configuration** folder in the **Project Explorer** window.

☐ Open `Read Configuration Data.vi` and modify it to call `Read Boiler Controller Constants.vi`, as shown in Figure 6-21.

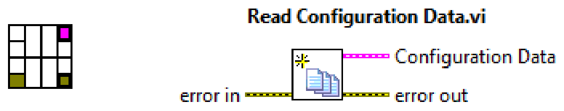
Figure 6-21. Read Configuration Data VI Calling Read Boiler Controller Constants VI



1 Delete the MHL Configuration indicator on the front panel and replace it with the Configuration Data type def control and complete the block diagram as shown.

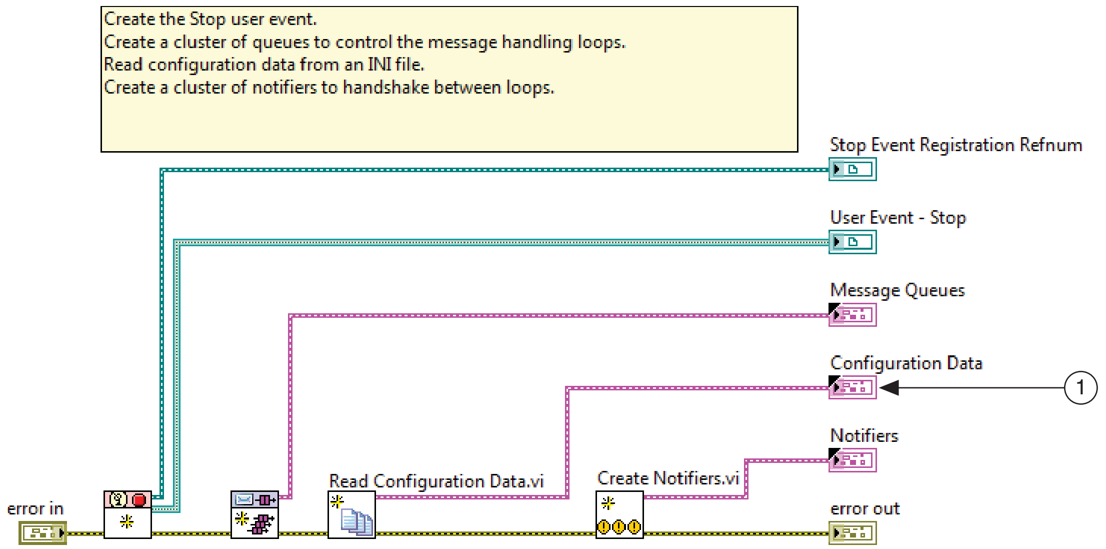
☐ Modify the `Read Configuration Data.vi` connections to assign the Configuration Data indicator as shown in Figure 6-22.

Figure 6-22. Read Configuration Data VI Connections with Configuration Data



☐ Modify `Boiler System Open.vi` to use the new output of `Read Boiler Controller Constants.vi`, as shown in Figure 6-23.

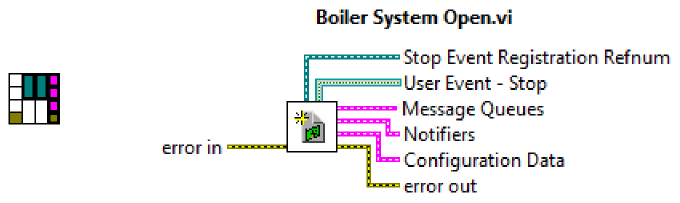
Figure 6-23. Boiler System Open VI


Create the Stop user event.
Create a cluster of queues to control the message handling loops.
Read configuration data from an INI file.
Create a cluster of notifiers to handshake between loops.
Stop Event Registration Refnum
User Event - Stop
Message Queues
Configuration Data
1
Notifiers
error in
Read Configuration Data.vi
Create Notifiers.vi
error out

1 Replace the MHL Configuration type def on the front panel with the Configuration Data type def and complete the wiring as shown.

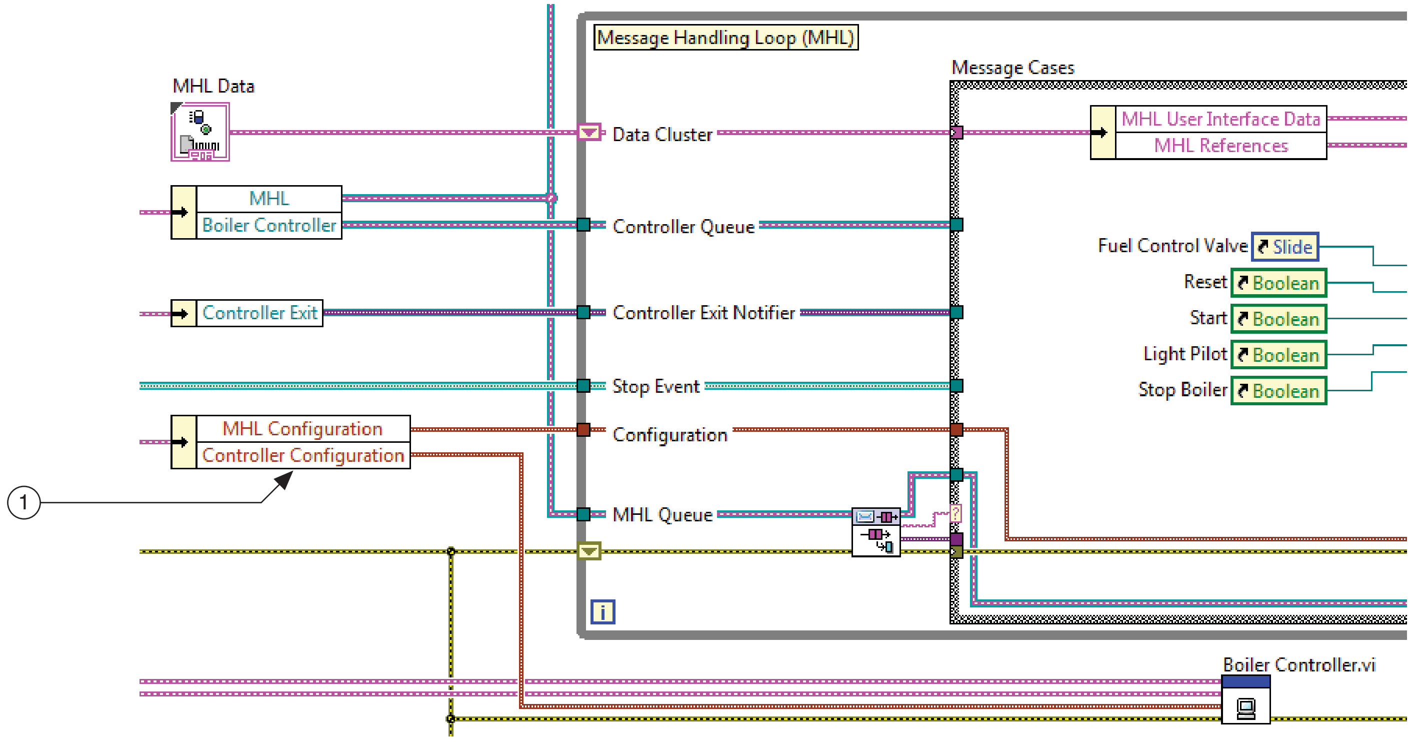
☐ Modify the `Boiler System Open.vi` connections to assign the Configuration Data indicator as shown in Figure 6-24.

Figure 6-24. Boiler System Open VI Connections with Configuration Data Indicator



☐ Modify `Main.vi`, as shown in Figure 6-25.

Figure 6-25. Main VI with MHL Configuration and Controller Configuration Passed to their Respective Loops



1 Unbundle MHL Configuration and Controller Configuration and pass them to their respective loops.

11. Test `Wait.vi`.

☐ Save and run `Main.vi`.

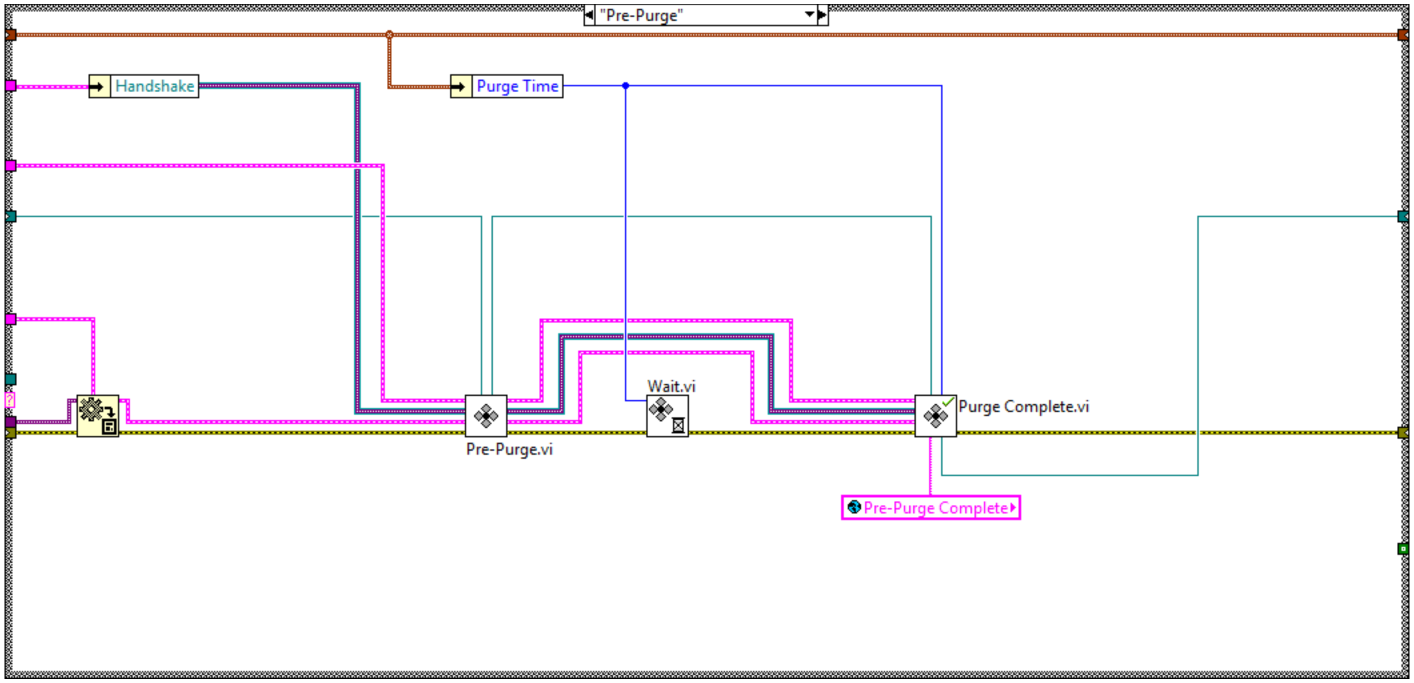
☐ Verify that previous functionality still works.

☐ When you click Start, there is now a 2 second delay before the Boiler Controller - Purge Complete.vi dialog box appears. Click **OK**.

☐ Click **Emergency Stop**.

12. Integrate `Purge Complete.vi`, as shown in Figure 6-26.

Figure 6-26. Boiler Controller VI with Purge Complete VI Integrated



13. Test `Purge Complete.vi`.

- ☐ Run `Main.vi`.
- ☐ Verify that previous functionality still works.
- ☐ When you click start, after the 2-second delay, the **Primary Fan** turns off and the **Status** displays **`Pre-Purge Complete`**.
- ☐ Click **Emergency Stop**.
- ☐ Open `Status Log.txt` and note the new **`Pre-Purge Complete`** entry.

Test the Application

You have already tested the functionality of these code modules and their interaction with the rest of the system through top-down integration testing.

1. Test the build specification

   - ☐ Build the executable from the build specification.
   - ☐ Run `Boiler Controller.exe`.
   - ☐ Verify the behavior that you tested as you integrated each module.

End of Exercise 6-2

Exercise 6-3 Integrate and Test Modules from the Bottom-Up

Goal

Perform bottom-up integration testing for the Light Pilot button functionality.

Scenario

In the sprint planning meeting for this iteration, the product owner chose to implement the following user story:

- As a boiler operator, I want to click a button to start the flow of gas, ignite the pilot, prove the pilot flame, and start the boiler so that these steps always occur in the proper order to ensure safe operation.

To implement this user story, the team identifies the following tasks:

1. Create code to test each module.
2. Integrate and test each code module with the Boiler Controller system.

Design

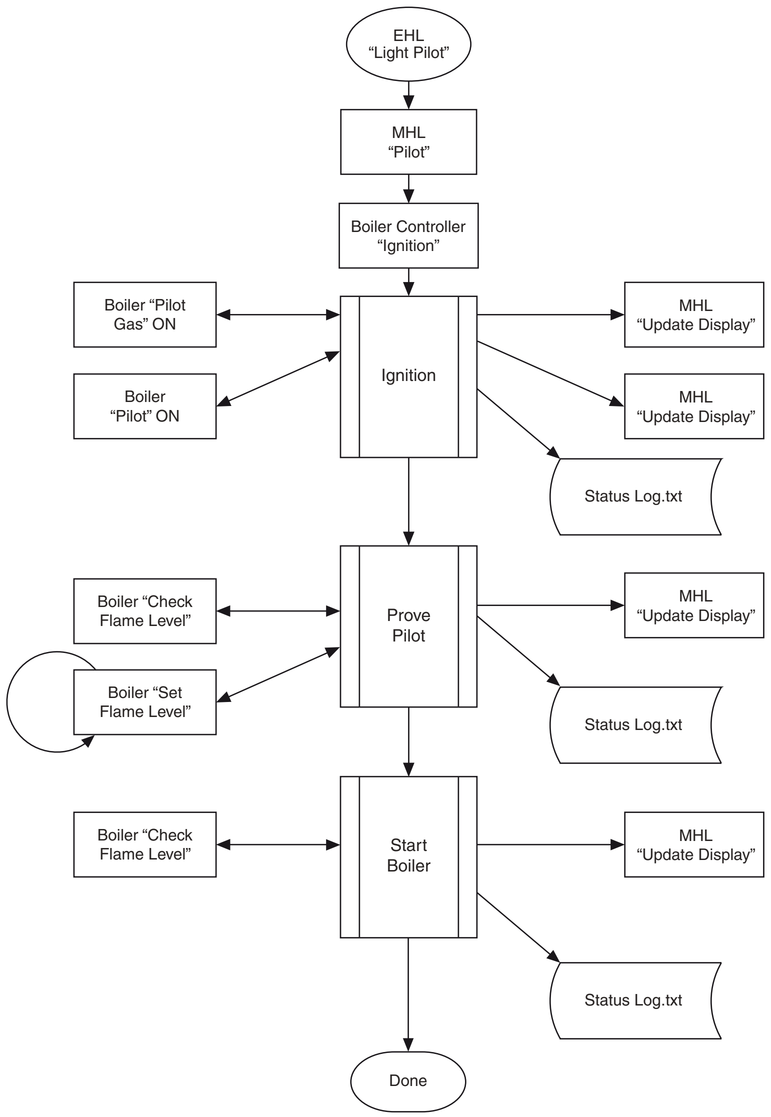
The team decides to use bottom-up integration testing to test and integrate the code modules for the Light Pilot functionality. This method of testing allows you to identify any module functional problems earlier in the test/integration process. However, it will take a bit longer to find out if the top-level application control methods are appropriate for these modules.

The team decides on the following strategies for approaching each task:

1. Create code to test each module.
    - `Boiler Controller.lvlib` includes the modules that you will need to test for this functionality (`Ignition.vi`, `Prove Pilot.vi`, and `Start Boiler.vi`).
    - You will develop a test harness for `Ignition.vi`.
    - Teammates will develop test harnesses for `Prove Pilot.vi` and `Start Boiler.vi`.
    - Test each module prior to integrating it with the rest of the system.

2. Integrate each code module with the Boiler Controller system.
   - Since all three pieces have been tested individually, it is now time to test how the interact with each other.
   - You will integrate all three modules into the Ignition case of `Boiler Controller.vi`.
   - You will integrate this functionality with `Main.vi`.

Figure 6-27. Implementing the Light Pilot Functionality



Implementation

Create Code to Test Each Module

1. Move `Test Harness.vit` from `<Exercises>\LabVIEW Core 3\External\Test VIs` to `<Exercises>\LabVIEW Core 3 Course Project\Test VIs`. Add it to the `Test VIs` virtual folder in the Project Explorer window of the Boiler Controller.

2. Create code to test `Ignition.vi`.

   ☐ Create `Test - Ignition.vi`.

   – From Windows Explorer, open `Test Harness.vit`.

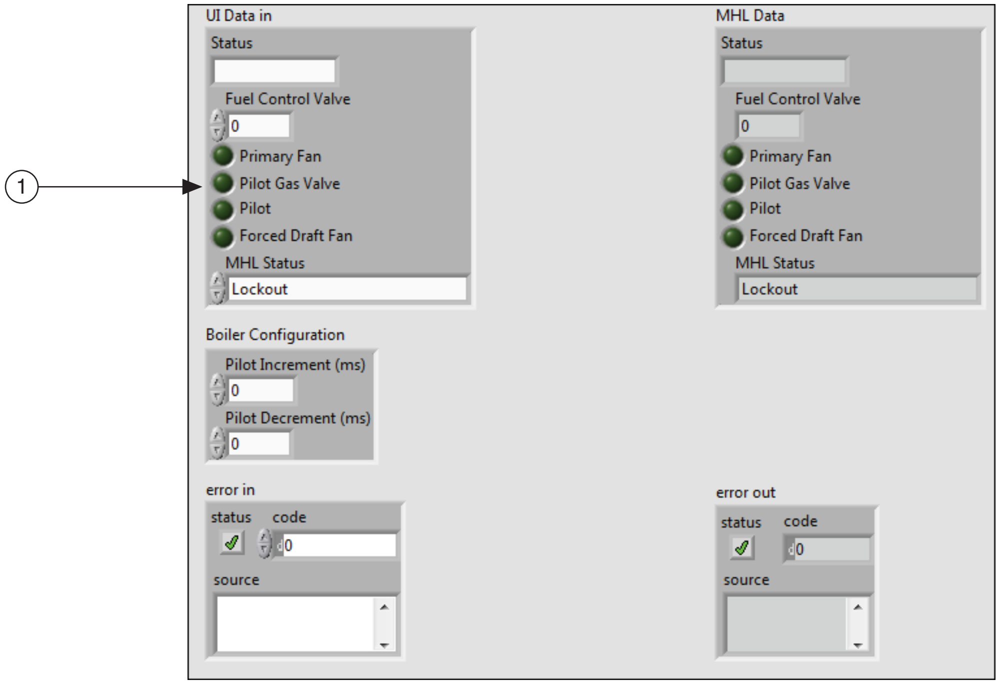
Note By opening the VI template file in Windows Explorer, LabVIEW forces you to save the template after you open it. Opening from the project, does not force you to save. However, the file must be part of the project or the linking does not work.

   – Save this VI as `Test - Ignition.vi` in the `<Exercises>\LabVIEW Core 3\Course Project\Controller\Controller Tests` directory.

– Create a `Test VIs` virtual folder in the `Controller.lvlib` and add `Test - Ignition.vi` to this folder.

– Modify the VI as shown in Figure 6-28 and Figure 6-29.

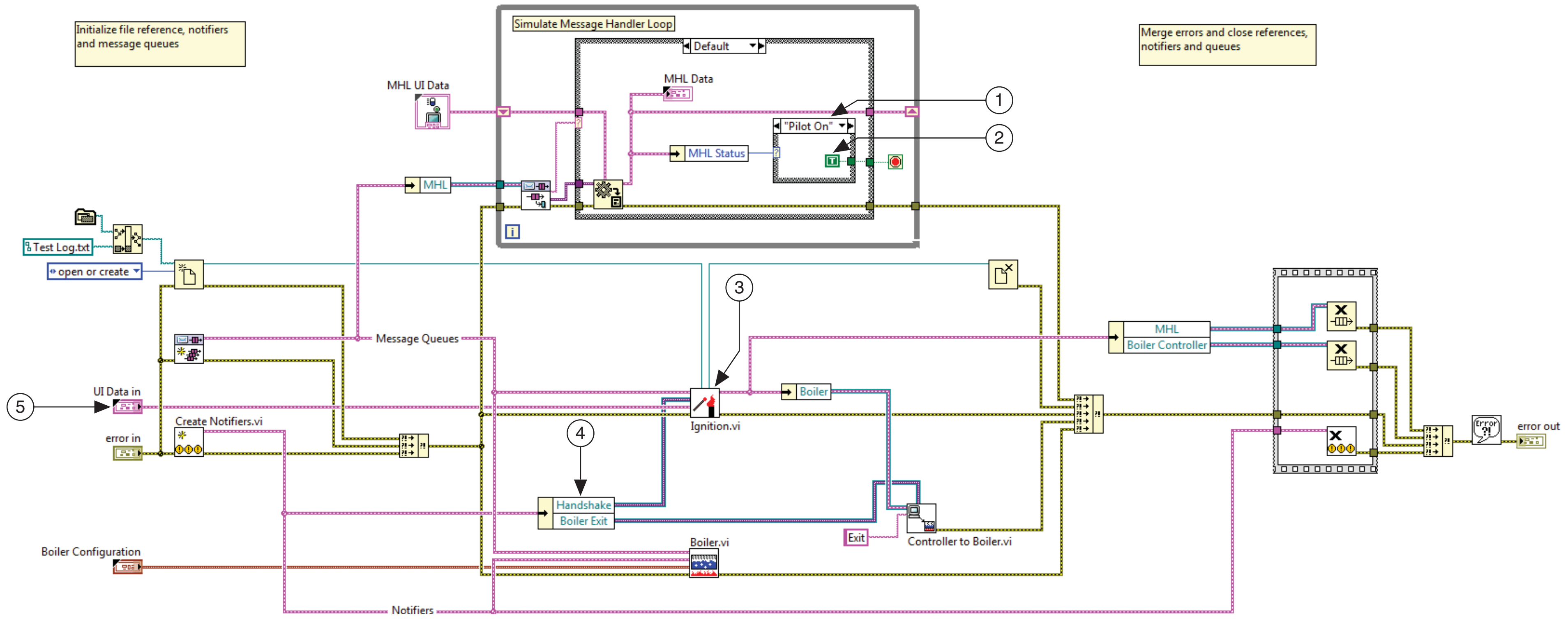
Figure 6-28. Test - Ignition VI Front Panel



1 UI Data in control

- ☐ Create a copy of the MHL Data indicator.
- ☐ Right-click the copy and select **Change to Control**.
- ☐ Rename the control.

Figure 6-29. Test - Ignition VI Block Diagram



1 Create the Pilot On case.

2 Change the value of the constant in the Default Case to False.

3 Ignition VI

4 Expand the Unbundle by Name and select Handshake

5 UI Data in Control

3. Execute the test.

   ☐ Open `Boiler.vi`.

   Note You must open `Boiler.vi` first, because that VI is configured to show its front panel when it runs and to close when it is finished and for this test, you want to observe changes to the `Boiler.vi` front panel.

   ☐ Run `Test - Ignition.vi` and observe the following:

   – The **Status** and **MHL Status** display **`Pilot On`**

   – The **Pilot Gas Valve** and **Pilot** LEDs turn on for both the Test - Ignition VI and Boiler VI.

4. Create code to test `Prove Pilot.vi`.

   ☐ Create `Test - Prove Pilot.vi`.

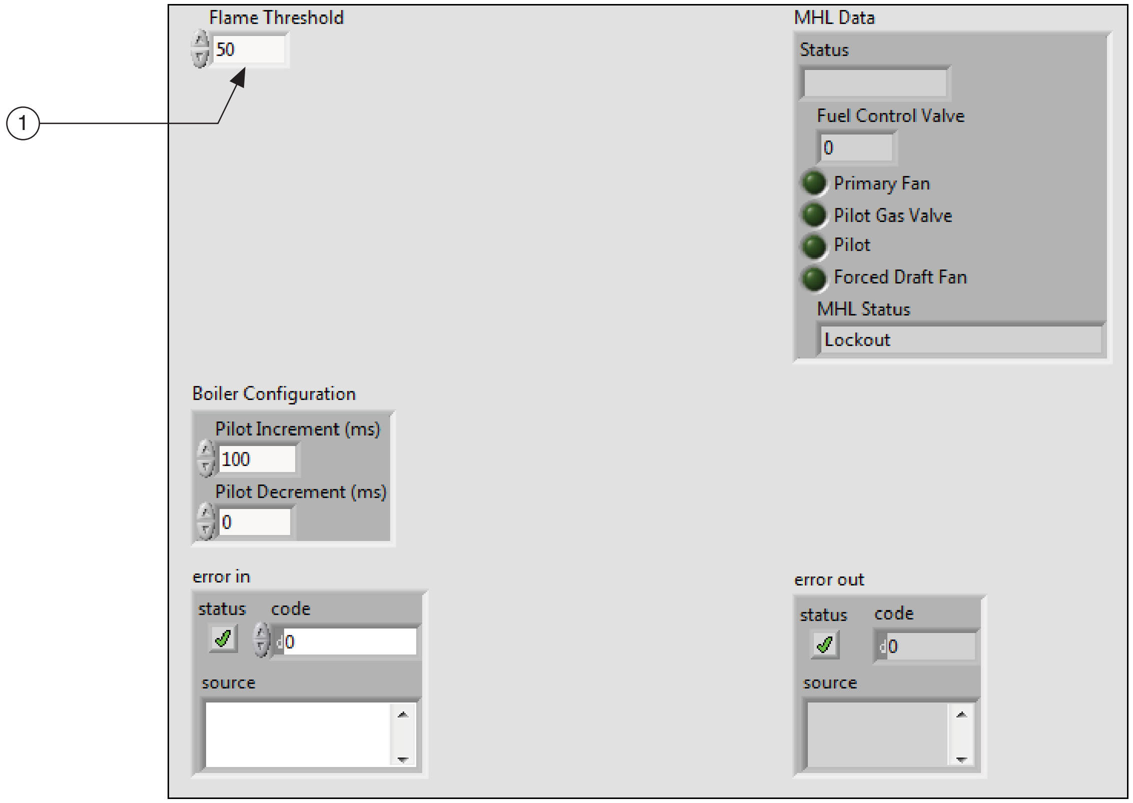
   – From Windows Explorer, open `Test Harness.vit`.

   – Save this VI as `Test - Prove Pilot.vi` in the `<Exercises>\LabVIEW Core 3\Course Project\Controller\Controller Tests` directory.

– Add the VI to the `Test VIs` virtual folder in `Controller.lvlib`.

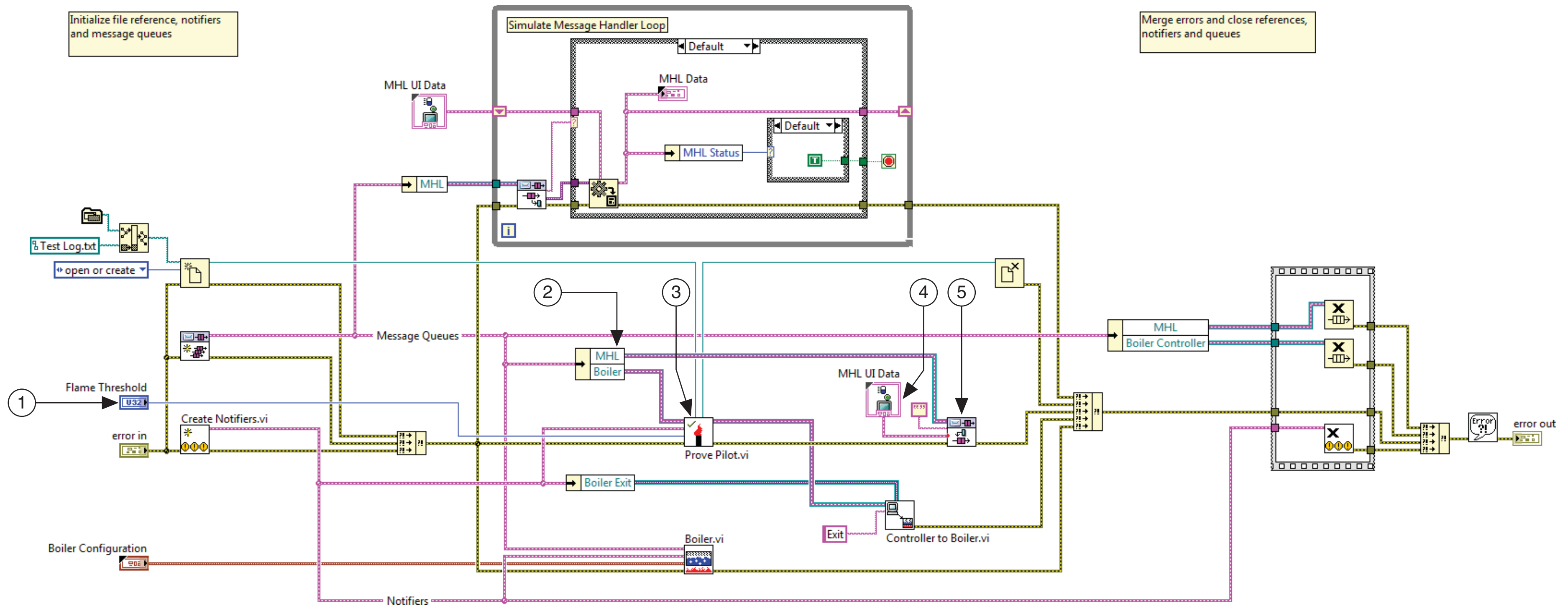
– Modify the `Test - Prove Pilot.vi` as shown in Figure 6-30 and Figure 6-31

Figure 6-30. Test - Prove Pilot VI Front Panel



1 Numeric Control—Create the `Flame Threshold` control and set the representation to U32.

Figure 6-31. Test - Prove Pilot VI Block Diagram



1 Flame Threshold control

2 Unbundle the MHL and Boiler message queues

3 Prove Pilot VI

4 MHL UI Data type def constant

5 Enqueue Message VI

5. Execute the test

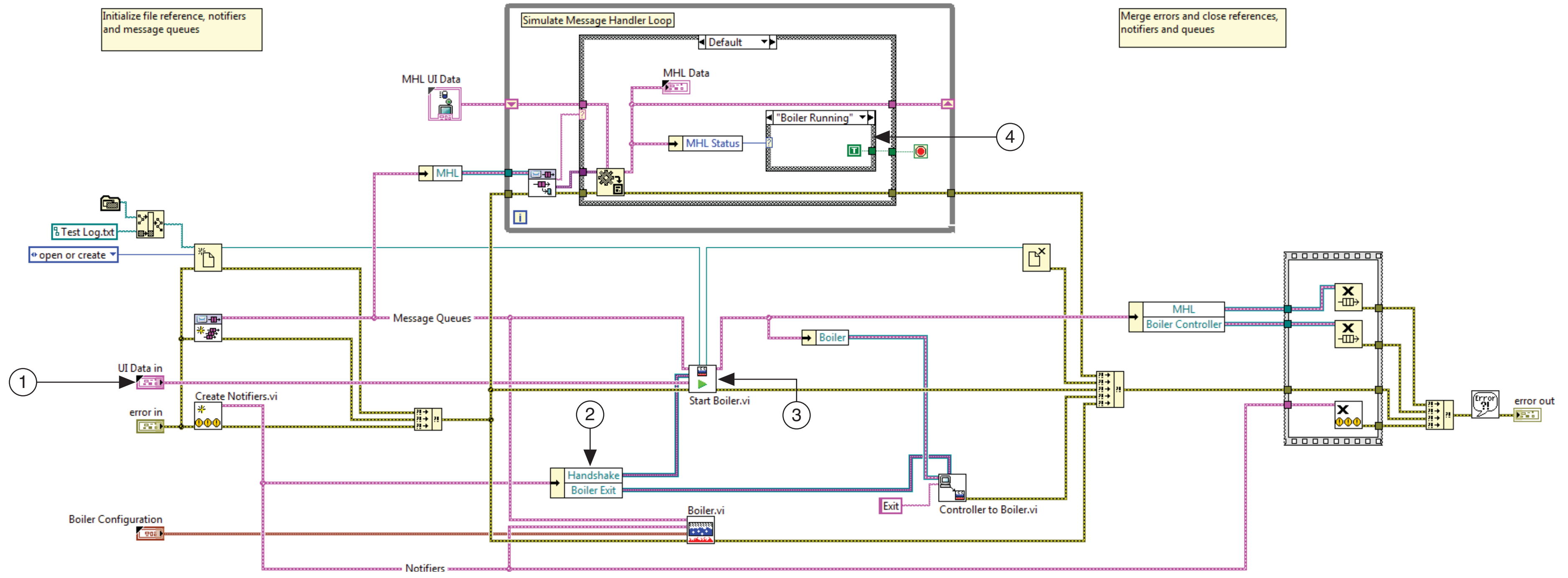
   - ☐ Open `Boiler.vi`.
   - ☐ Set the following values in `Test - Prove Pilot.vi`.
     - – Set Flame Threshold to `50`
     - – Set Pilot Increment to `100`
   - ☐ Run `Test - Prove Pilot.vi`.
   - ☐ Observe the Pilot Flame Level on the `Boiler.vi` front panel increment from 0 to 50 in five seconds.

6. Create code to test `Start Boiler.vi`.

   - ☐ Create `Test - Start Boiler.vi`.
     - – From Windows Explorer, open `Test Harness.vit`.
     - – Save this VI as `Test - Start Boiler.vi` in the `<Exercises>\LabVIEW Core 3\Course Project\Controller\Controller Tests` directory.
   - ☐ Add the `Test - Start Boiler.vi` to the `Test VIs` virtual folder in `Controller.lvlib`.

☐ Modify the VI as shown in Figure 6-32.

Figure 6-32. Test - Start Boiler VI Block Diagram



1 Create the UI Data in control similar to the one you created for the Test - Ignition.vi.

2 Expand the Unbundle by Name and select Handshake

3 Start Boiler VI

4 Create the Boiler Running case as shown and change the Boolean constant in the Default case to False.

7. Execute the test.

   ☐ Open `Boiler.vi`.

   ☐ Run `Test - Start Boiler.vi` and observe the following values.

   – **Status** displays `Boiler Running`.

   – The **Fuel Control Valve** indicator on the Test - Start Boiler VI and the **Fuel Level (%)** indicator on the Boiler VI both indicate a value of **`10`**.

   – The **Forced Draft Fan** LED turns on for both the test VI and `Boiler.vi`.

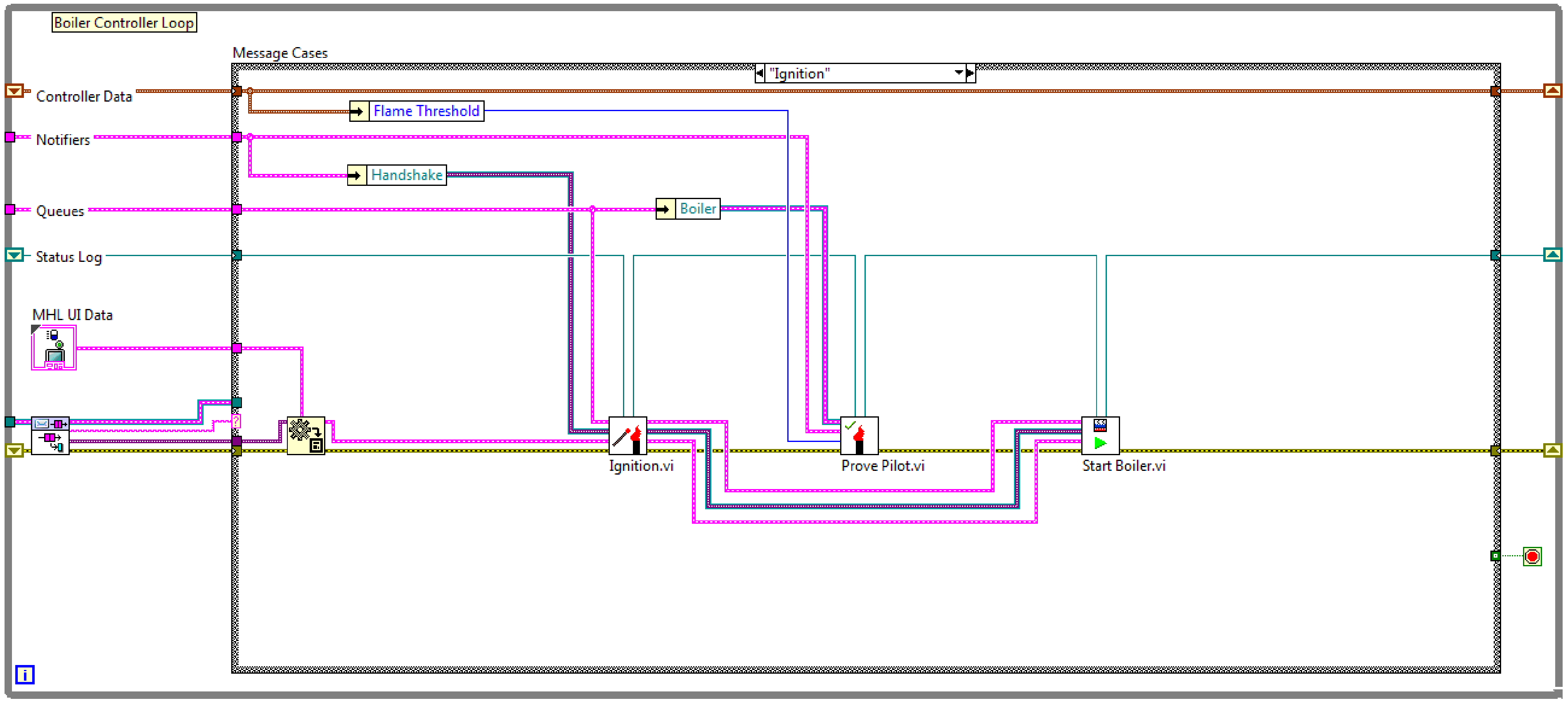
   – **MHL status** displays `Boiler Running`.

Integrate Each Module with the Boiler Controller System

Now that you have verified that each code module functions as expected, integrate the modules into `Boiler Controller.vi`.

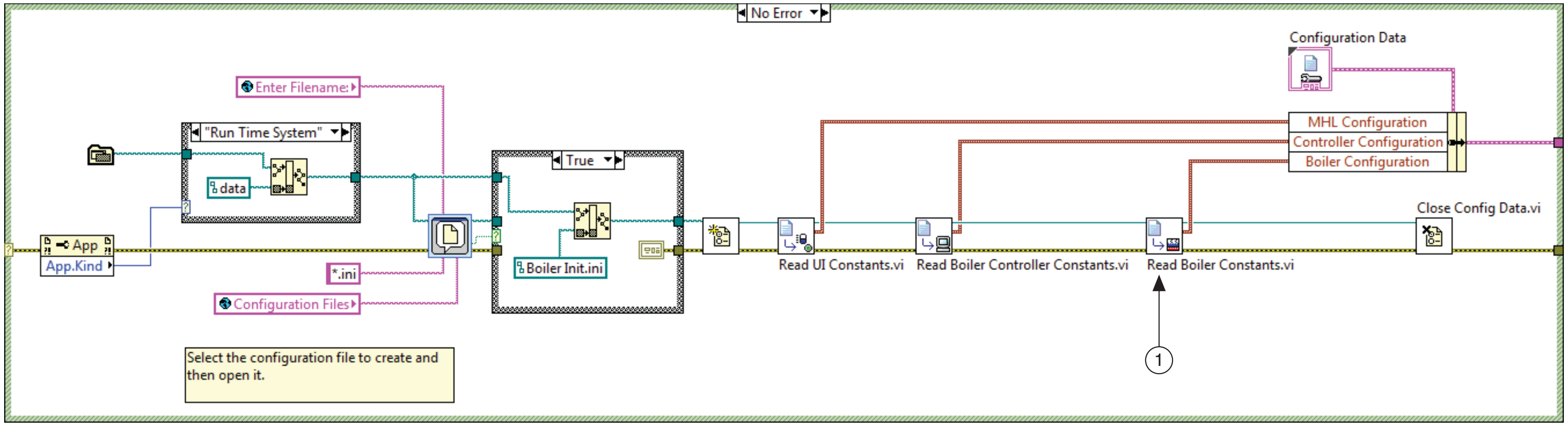
1. Open `Boiler Controller.vi` and modify the Ignition case, as shown in Figure 6-33.

Figure 6-33. Boiler Controller VI Ignition Case



2. Modify `Read Configuration Data.vi` to call `Read Boiler Constants.vi` as shown in Figure 6-34.

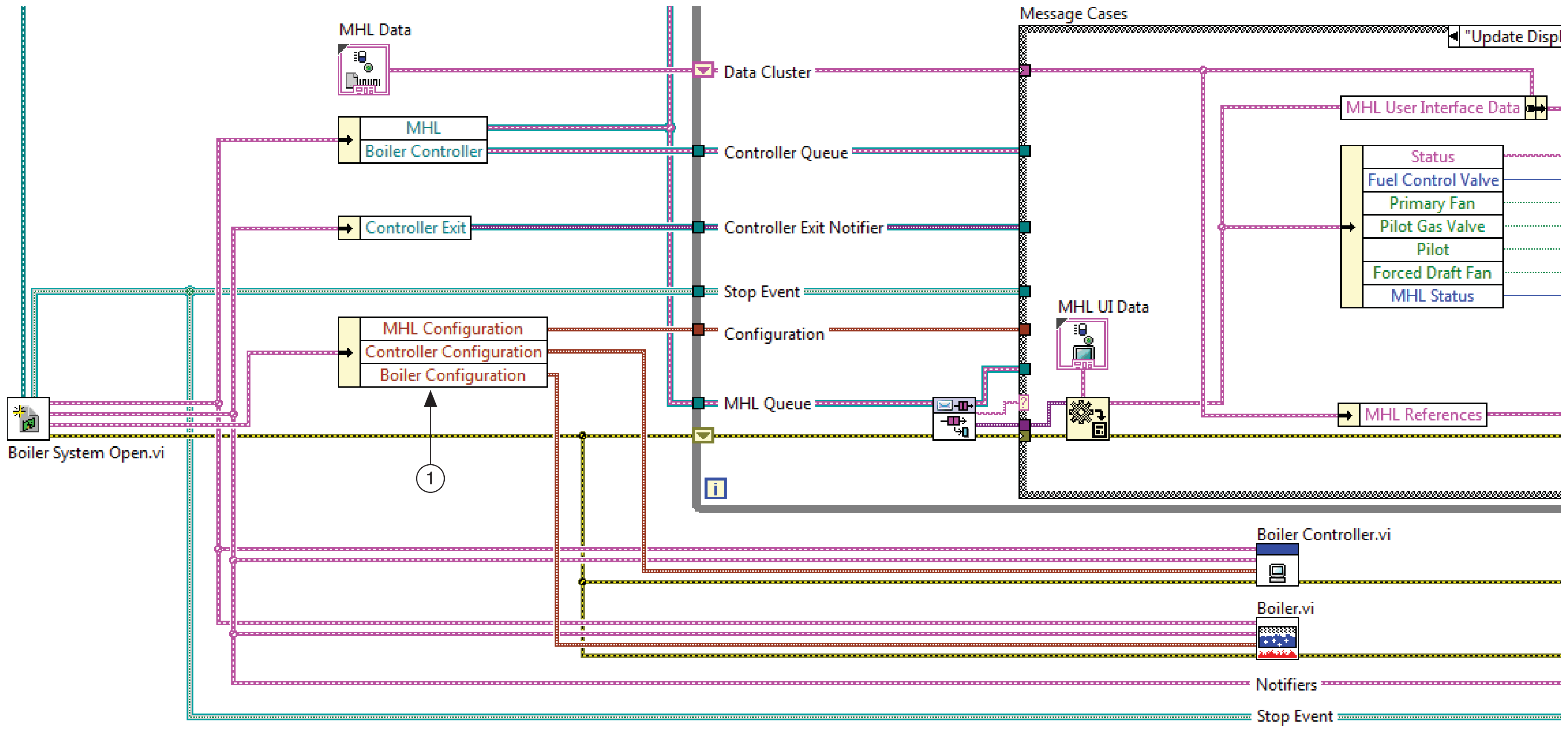
Figure 6-34. Read Configuration Data VI Calling Read Boiler Constants VI



1 Add `Read Boiler Constants.vi` from the `<Exercises>\LabVIEW Core 3\Course Project\Support\Configuration` directory to the **Support VIs»Configuration** directory in the Project Explorer window.

☐ Modify `Main.vi`, as shown in Figure 6-35 to extract Boiler Configuration and pass it to `Boiler.vi`.

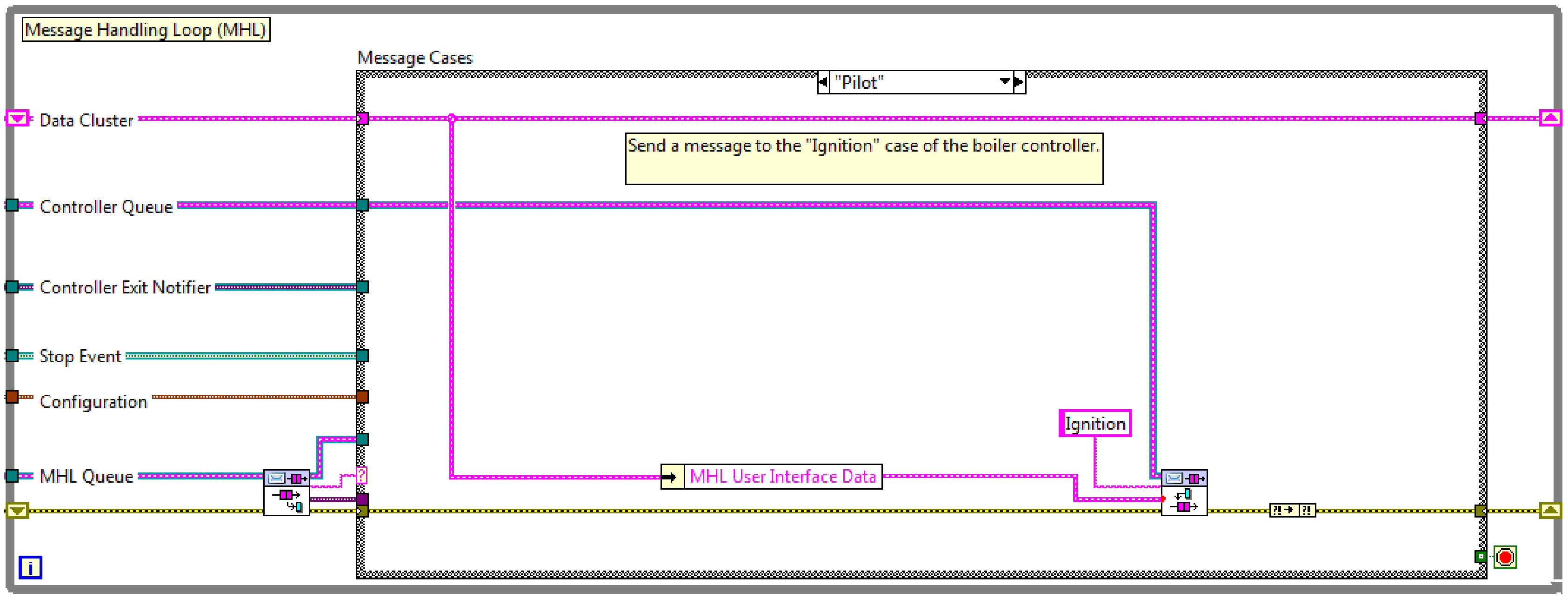
Figure 6-35. Main VI Passing Boiler Configuration to Boiler VI



1 Unbundle Boiler Configuration and pass it to the Boiler VI

3. Modify the Pilot case of the Message Handling Loop in `Main.vi` to pass MHL User Interface Data as the message data for the Ignition message to the Boiler Controller as shown in Figure 6-36.

Figure 6-36. Main VI Pilot Case Passing MHL User Interface Data



Test the Application

1. Test the VI.

☐ Run the VI. Verify that all previously-implemented functionality still works.

☐ When you click **Light Pilot**, the following should occur.

– Light Pilot is disabled.

– Status = Pilot On

- Pilot Gas Valve and Pilot LEDs turn on for both the test `Main.vi` and `Boiler.vi`.
- On `Boiler.vi`, the Pilot Flame Level increments from 0 to 100 in 10 seconds.

☐ When the Pilot Flame Level reaches 30%, observe the following changes.

- Status changes to Boiler Running.
- Fuel Control Valve and Stop Boiler are both enabled.
- Fuel Control Valve on `Main.vi` and Fuel Level (%) on Boiler.vi both = 10.
- Forced Draft Fan LED turns on for both `Main.vi` and `Boiler.vi`.

☐ Click Emergency Stop.

☐ Open `Status Log.txt` and review the following new entries.

- Pilot On
- Pilot Proved
- Forced Draft Fan On

2. Test the build specification

☐ Build the executable from the build specification.

☐ Run `Boiler Controller.exe`.

☐ Verify the behavior from Step 1.

End of Exercise 6-3

Exercise 6-4 Implement Fuel Control Valve Functionality

Goal

Implement functionality surrounding the fuel control valve.

Scenario

The product owner gathered feedback from the customer representative and a boiler expert after the previous sprint. The boiler expert pointed out that when the boiler enters the Boiler Running state, the pilot fuel valve should be closed, turning off the pilot light. The customer pointed out that the **Simulate Failure** button should be enabled and monitored once the boiler is running and there should be a status log entry when the boiler starts running. This entry should record the fuel control valve level when the boiler starts running.

In the sprint planning meeting for this iteration, the product owner chose to implement the above feedback and the following user story:

- As a boiler operator, I want to adjust the rate at which fuel enters the boiler furnace, so that I can adjust the temperature of the boiler and the fuel level is always within safe levels.

To implement the user feedback and this user story, the team identifies the following tasks:

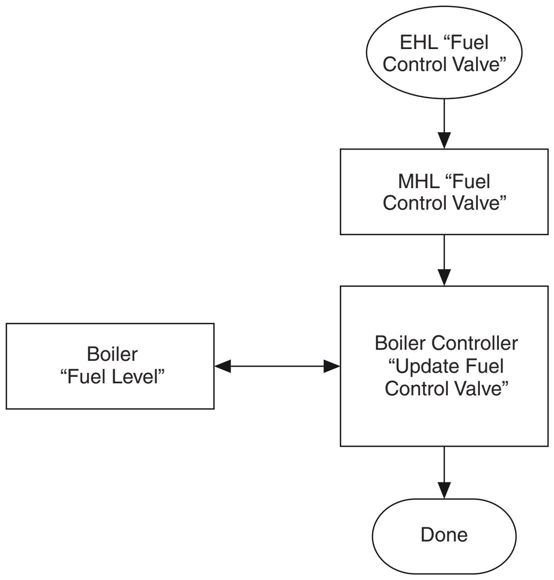
1. Create a new state in the boiler controller to handle the steps involved in running the boiler.
2. When the user updates the fuel control valve level, send the new value to the boiler.

Design

The team decides on the following strategies for approaching each task:

1. Create a new state in the boiler controller to handle the steps of running the boiler.
    - The pieces to implement this functionality already exist. The Run Boiler case, which executes immediately after the Ignition case, just needs to call the functionality.
2. When the user updates the fuel control valve level, send the new value to the boiler.
    - Use the handshaking notifier to send the fuel control valve value to the boiler.

Figure 6-37. Fuel Control Valve Functionality



Implementation

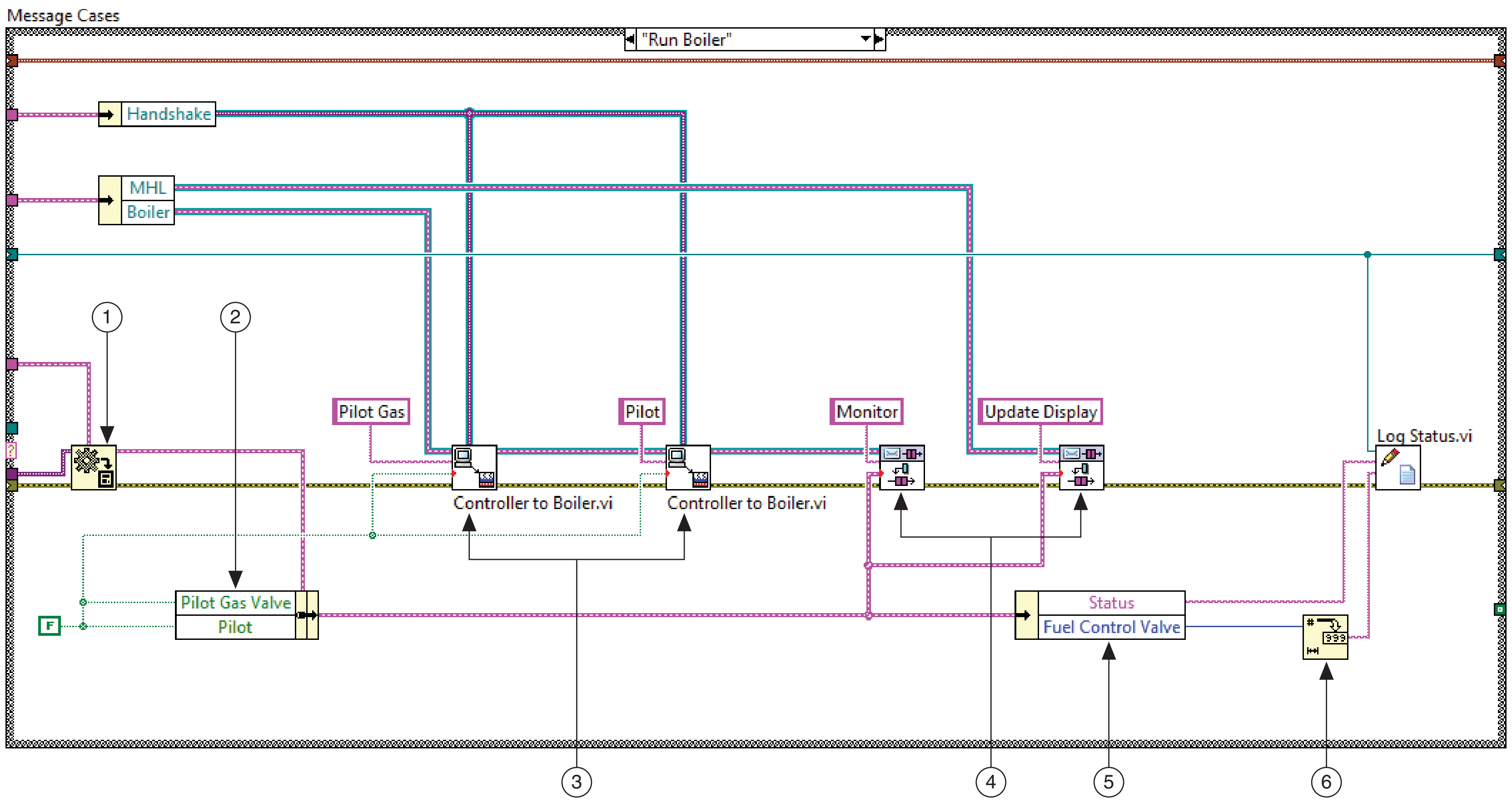
Create a New State in the Boiler Controller VI to Implement Boiler Running States

Note Because the Run Boiler VI directly implements the new functionality and each piece is relatively small, you test the functionality when the case is finished.

1. Open the Boiler Controller VI block diagram.
2. Create a new case after the Ignition case and name it `Run Boiler`.

3. Modify the case as shown in Figure 6-38.

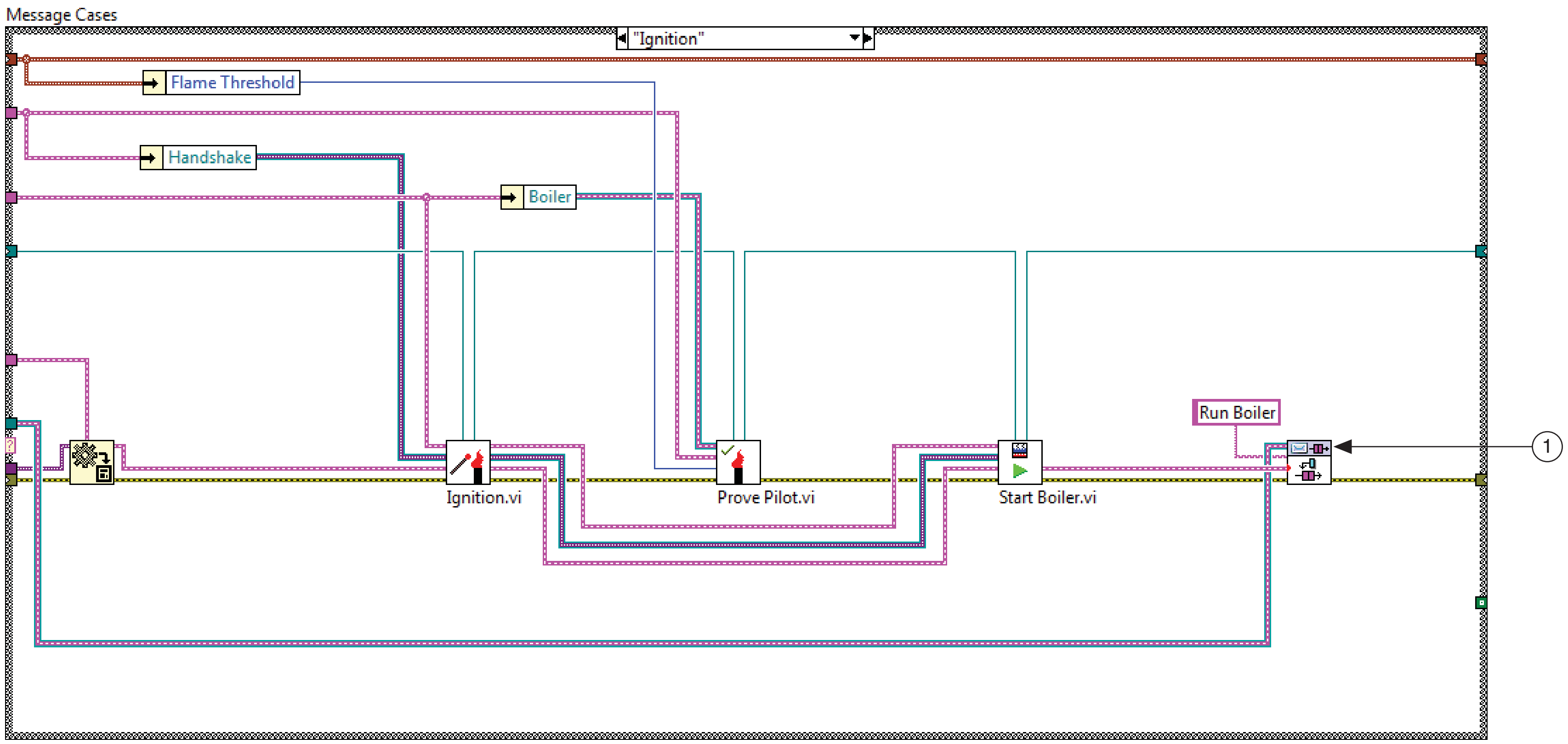
Figure 6-38. The Run Boiler Case of the Boiler Controller VI



1 **Variant to Data**—Converts variant data from the Dequeue Message VI to a MHL UI Data type.
2 **Bundle By Name**—Sets the Pilot Gas Valve and Pilot indicators to off, indicating that the pilot gas valve is closed and the pilot light is off.
3 **Controller to Boiler**—This VI sends a message to the boiler to turn off the gas and pilot light.
4 **Enqueue Message**—The first instance sends a message to the Boiler.vi and the second sends a message to the MHL to update the status of Boiler Controller.vi.
5 **Unbundle By Name**—Extracts the Status and Fuel Control Valve values.
6 **Number To Decimal String**—Converts the fuel control valve data to a string so the Log Status VI can log the information.

4. Modify the Ignition case to call the Run Boiler case as shown in Figure 6-39.

Figure 6-39. Updating the Ignition Case to Call the Run Boiler Case



1 **Enqueue Message**—Sends the `Run Boiler` message to the Boiler Controller queue.

5. Save the Boiler Controller VI.

6. Run the Main VI.

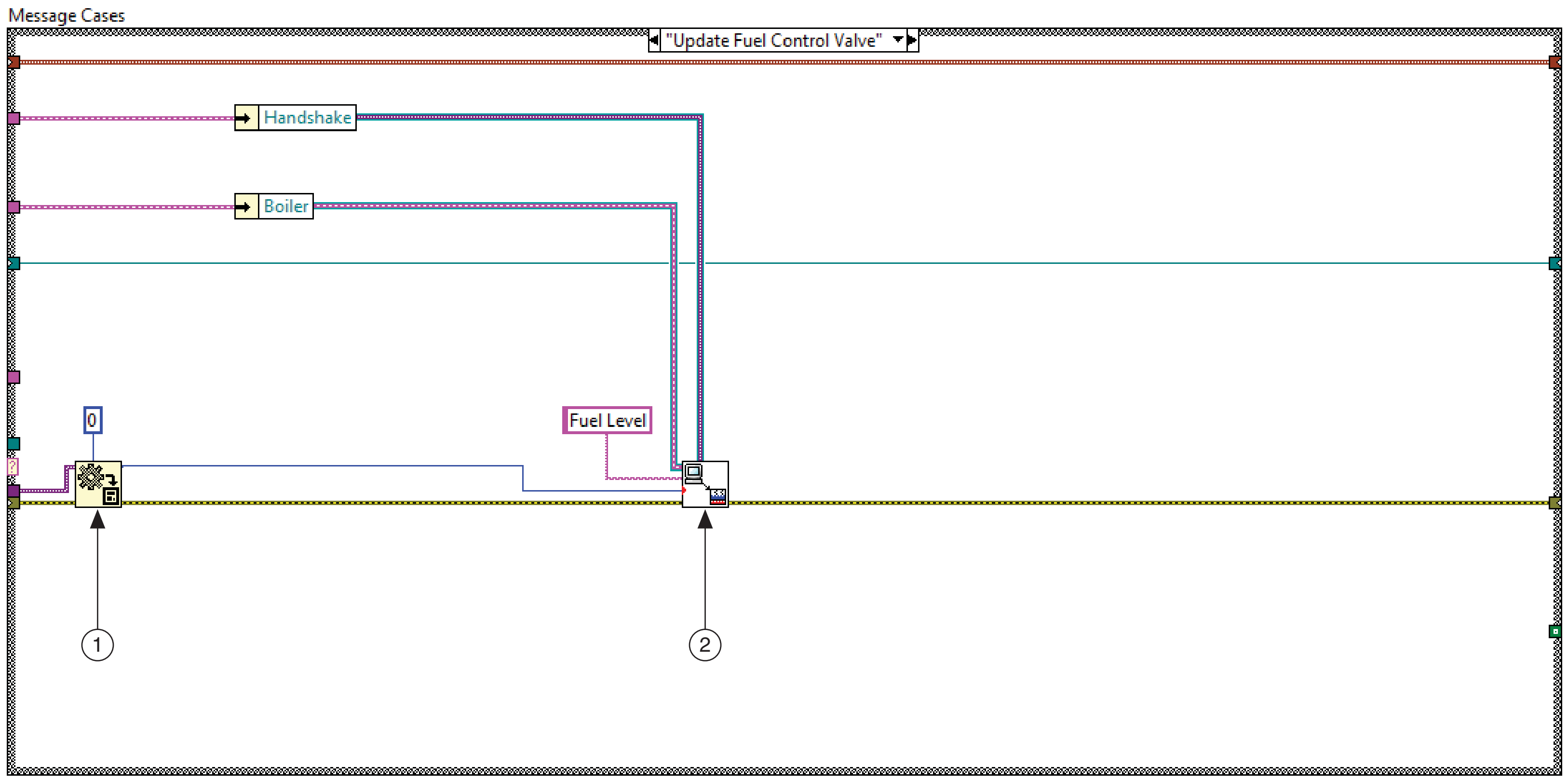
7. Verify the following:

- ☐ Previously implemented functionality still works.
- ☐ When the pilot flame reaches the flame threshold level, the **Pilot Gas Valve** and **Pilot** LEDs turn off and the **Simulate Failure** button is enabled on the Boiler VI.
- ☐ Clicking the **Simulate Failure** button displays the **Boiler Controller - Shutdown** dialog box. Simulating failure results in a boiler shutdown, in the same way as clicking **Stop Boiler** button.
- ☐ Click the **Emergency Stop** button.
- ☐ Open `Status Log.txt` and verify that the file contains the entry: `Boiler Running`.

Send New Fuel Control Valve Levels to the Boiler

1. Open the Boiler Controller VI block diagram and modify the Update Fuel Control Valve case, as shown in Figure 6-40.

Figure 6-40. Modifying the Update Fuel Control Valve Case

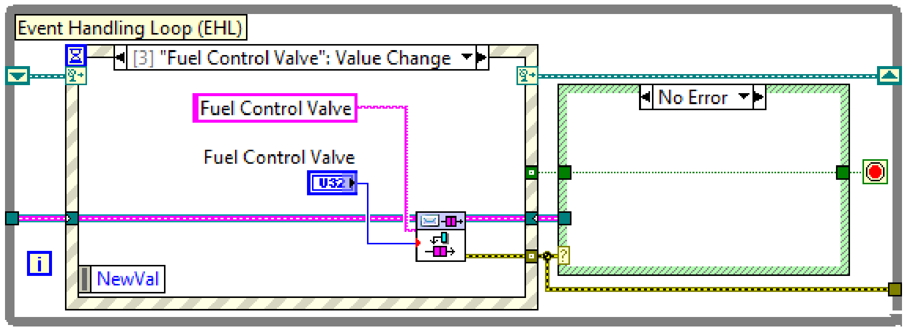


1	**Variant to Data**—Wire a Numeric constant of `0` to the **type** input of the Variant to Data function.
2	**Controller to Boiler**—This VI sends a message to the boiler about the fuel level.

2. Save the Boiler Controller VI.

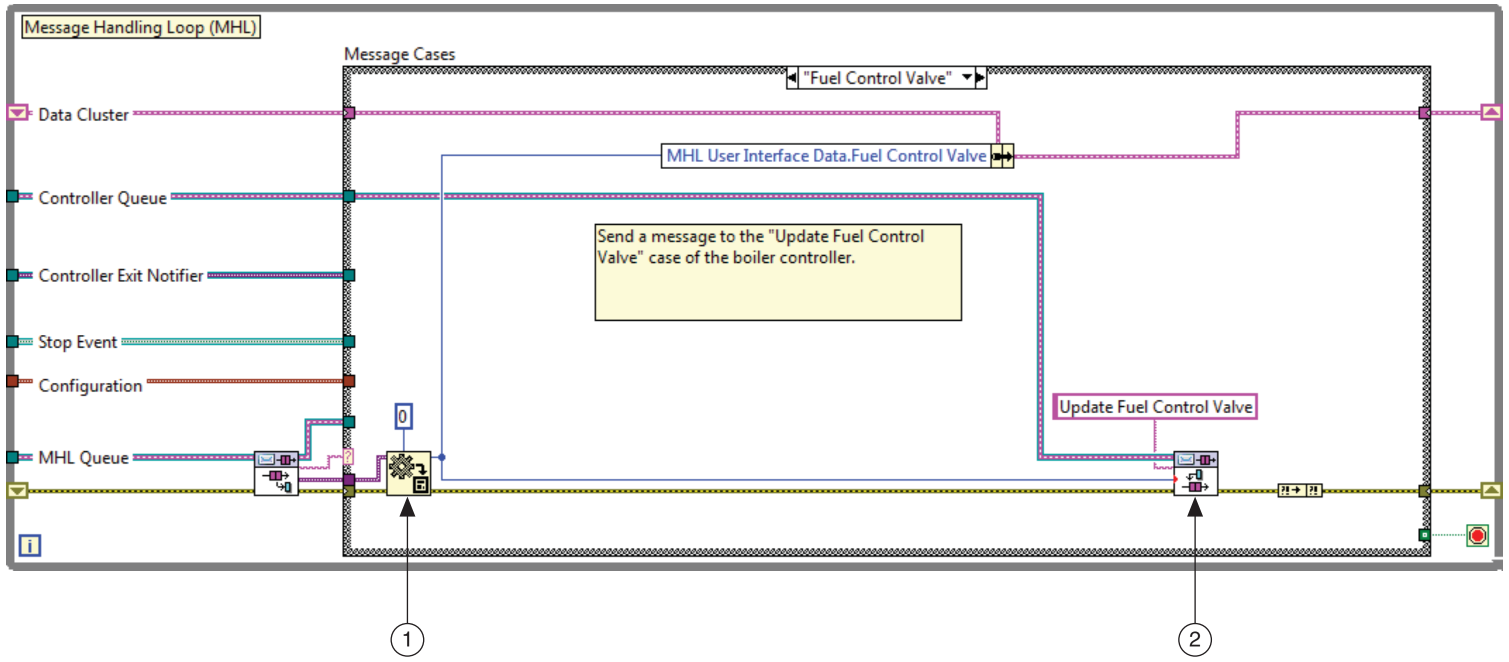
3. Open the Main VI block diagram and modify the Fuel Control Valve: Value Change event in the EHL to pass the **Fuel Control Valve** control value as message data to the MHL, as shown in Figure 6-41.

Figure 6-41. Updating the Fuel Control Valve Event in the EHL of the Main VI


Event Handling Loop (EHL)
[3] "Fuel Control Valve": Value Change
Fuel Control Valve
Fuel Control Valve
U32
NewVal
No Error

4. Modify the Fuel Control Valve case of the MHL as shown in Figure 6-42.

Figure 6-42. Modifying the Fuel Control Valve Case of the MHL


Message Handling Loop (MHL)
Message Cases
"Fuel Control Valve"
Data Cluster
MHL User Interface Data.Fuel Control Valve
Controller Queue
Send a message to the "Update Fuel Control Valve" case of the boiler controller.
Controller Exit Notifier
Stop Event
Configuration
Update Fuel Control Valve
0
MHL Queue
1
2

1 **Variant to Data**—Wire a Numeric constant of 0 to the **type** input of the Variant to Data function. Right-click the constant and select **Representation»U32** from the shortcut menu. U32 is the same representation as the **Fuel Control Valve** numeric control.

2 **Enqueue Message**—This VI sends the value of the fuel control valve to the boiler controller.

5. Save the Main VI.

6. Run the Main VI.

7. Verify the following:

   - ☐ Previously implemented functionality still works.
   - ☐ When the pilot flame reaches the flame threshold level, the **Fuel Control Valve** control is active and the value is 10.
   - ☐ Changing the value of the **Fuel Control Valve** control updates to the **Fuel Level (%)** indicator on the Boiler VI.
   - ☐ Click the **Emergency Stop** button.

Test the Application

You have already tested the Main VI for each task as part of completing that task, so you need to test only the executable.

1. Test the executable.

   - ☐ Build the executable from the build specification.
   - ☐ Run `Boiler Controller.exe`.
   - ☐ Verify the behavior of the new functionality.

End of Exercise 6-4

Exercise 6-5 Integrate and Sandwich Test a Module

Goal

Perform sandwich integration testing for the Stop Boiler button functionality.

Scenario

In the sprint planning meeting for this iteration, the product owner chose to implement the following user story:

- As a boiler operator, when the boiler is running, I want to click a button to shut down the boiler without exiting the system, so that the system always follows the boiler shutdown procedure and the boiler safely shuts down.

To implement this user story, the team identifies the following tasks:

1. Test each code module needed to test this functionality.
2. Integrate and test each code module with the Boiler Controller system.

Design

The team decides on the following strategies for approaching each task:

1. Test each code module needed to test this functionality.
    - Two code modules implement this functionality: `Shutdown Controller.vi` and `Purge Complete.vi`. You already tested and used `Purge Complete.vi` when you implemented the **Start** button functionality. You need only test `Shutdown Controller.vi` in this sprint.
    - Use sandwich testing to test the Shutdown Controller module. Sandwich testing requires a bit more overhead, but it provides all of the benefits of top-down testing (the ability to test the top-level control early in the process) and bottom-up testing (the ability to ensure module functionality early).
    - Sandwich testing, like bottom-up testing, requires that you develop a test VI for the Shutdown Controller module.
    - Sandwich testing also requires that you create stub code in the calling VI to ensure that the top-level control works properly.
2. Integrate and test the Shutdown Controller VI and Purge Complete VI code modules with the Boiler Controller system.
    - You modify the Shutdown case of the Boiler Controller VI to integrate this code.

Implementation

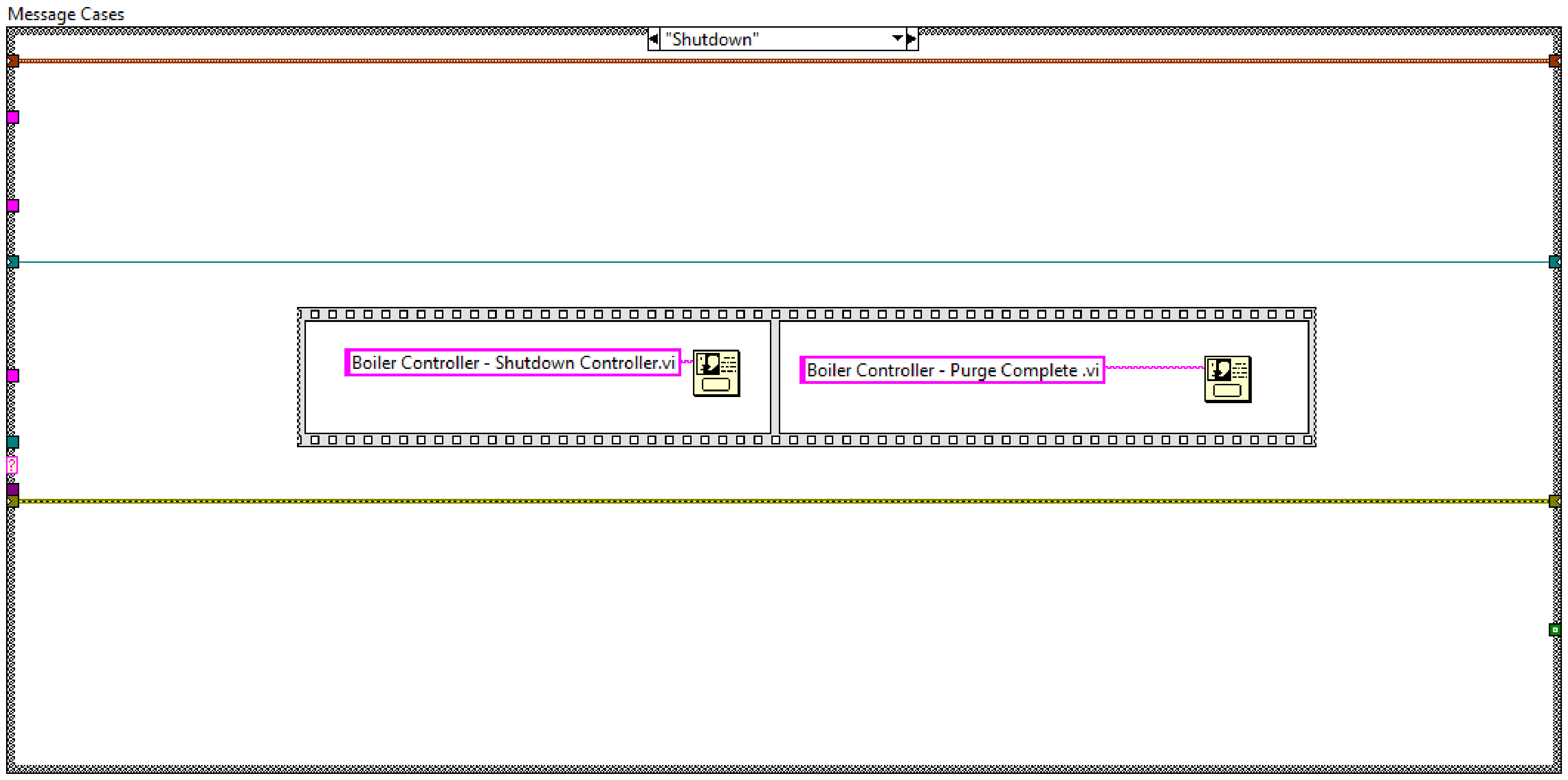
Test Each Code Module Needed

1. Create stub code for the Shutdown case of the Boiler Controller VI.

    ☐ Open the Boiler Controller VI block diagram.

    ☐ Modify the Shutdown case with a Flat Sequence Structure as shown in Figure 6-43.

Figure 6-43. Stub Code in the Shutdown Case


Message Cases
"Shutdown"
Boiler Controller - Shutdown Controller.vi
Boiler Controller - Purge Complete .vi

Note You delete these frames as you integrate each code module (Shutdown Controller VI and Purge Complete VI) into the Boiler Controller VI. These frames serve as placeholders until you are ready to integrate each module.

☐ Save the Boiler Controller VI.

☐ Run the Main VI.

☐ Verify that when you click the **Stop Boiler** button, the two dialogs boxes launch in order. This ensures that the top-level code, Main VI, is reaching the stub code.

☐ Click the **Emergency Stop** button.

2. Create a test VI for the Shutdown Controller VI.

   ☐ In the **Project Explorer** window, open **Boiler Controller»Controller.lvlib»Test VIs»Test - Prove Pilot.vi**.

   – Select **File»Save As**.

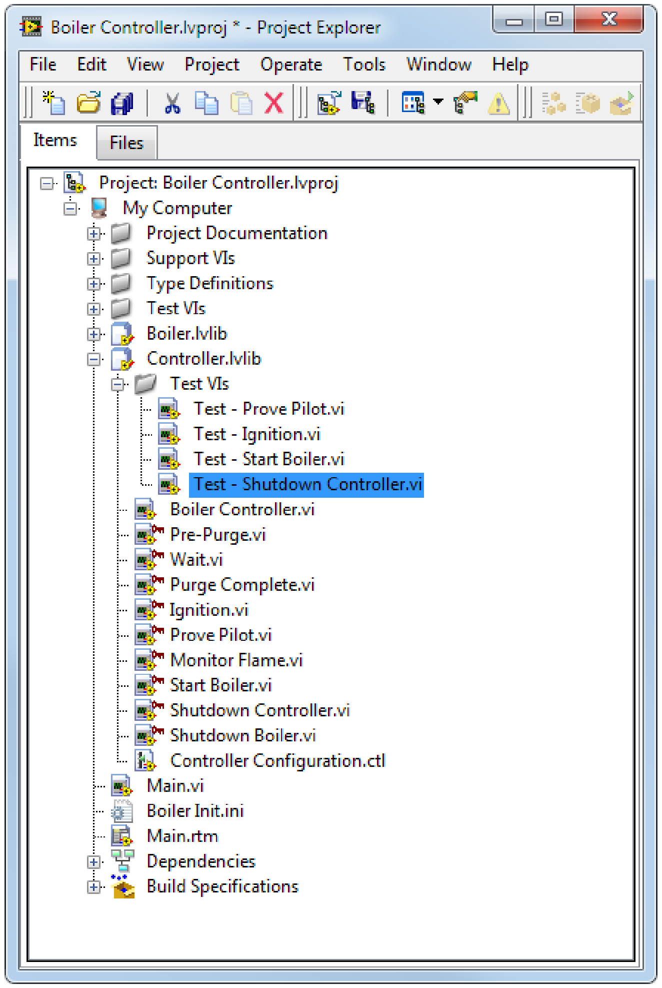
   – Select **Create unopened disk copy** and click **Continue**.

   – Save as `<Exercises>\Course Project\Controller\Controller Tests\Test - Shutdown Controller.vi`.

   ☐ Close `Test - Prove Pilot.vi`.

☐ Add **Test - Shutdown Controller.vi** to **Controller.lvlib** by dragging the VI to the **Test VIs** folder in the **Project Explorer** window as shown in Figure 6-44.

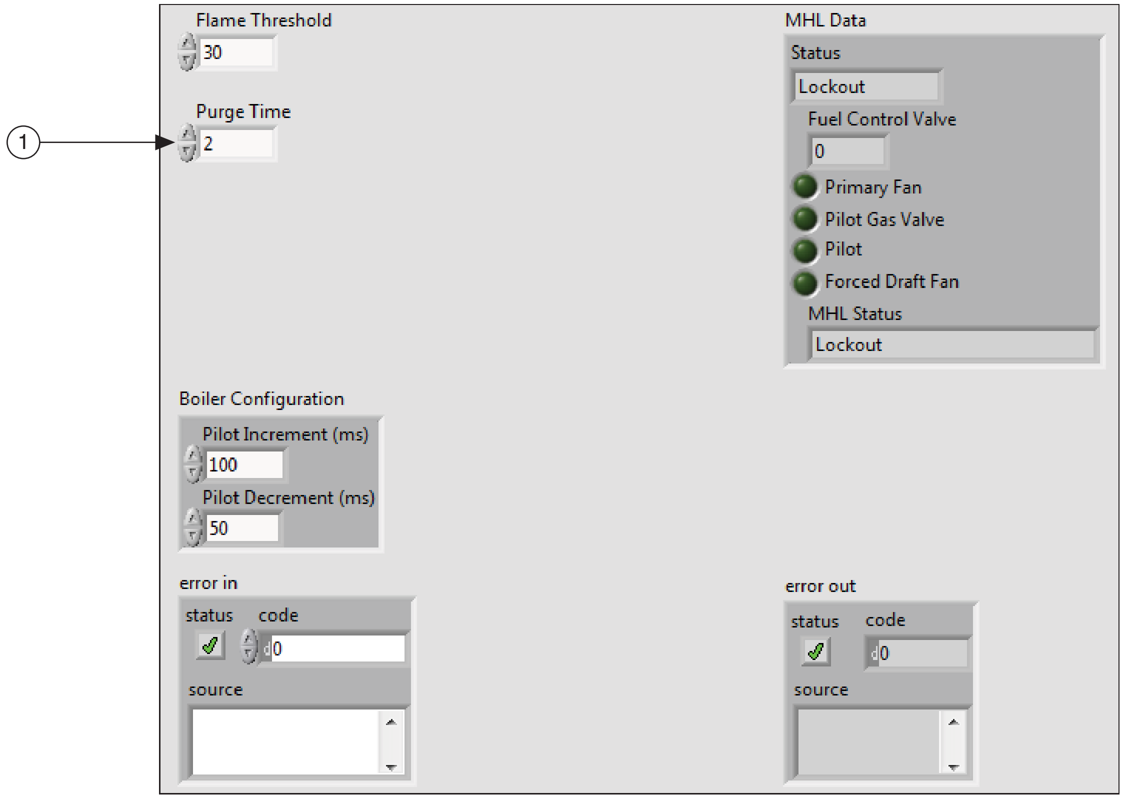
Figure 6-44. Project with Test - Shutdown Controller VI


Boiler Controller.lvproj * - Project Explorer
File Edit View Project Operate Tools Window Help
Items Files

```
Project: Boiler Controller.lvproj
  My Computer
    Project Documentation
    Support VIs
    Type Definitions
    Test VIs
    Boiler.lvlib
    Controller.lvlib
      Test VIs
        Test - Prove Pilot.vi
        Test - Ignition.vi
        Test - Start Boiler.vi
        Test - Shutdown Controller.vi
      Boiler Controller.vi
      Pre-Purge.vi
      Wait.vi
      Purge Complete.vi
      Ignition.vi
      Prove Pilot.vi
      Monitor Flame.vi
      Start Boiler.vi
      Shutdown Controller.vi
      Shutdown Boiler.vi
      Controller Configuration.ctl
    Main.vi
    Boiler Init.ini
    Main.rtm
    Dependencies
    Build Specifications
```

3. Update the front panel of the Test - Shutdown Controller VI as shown in Figure 6-45.

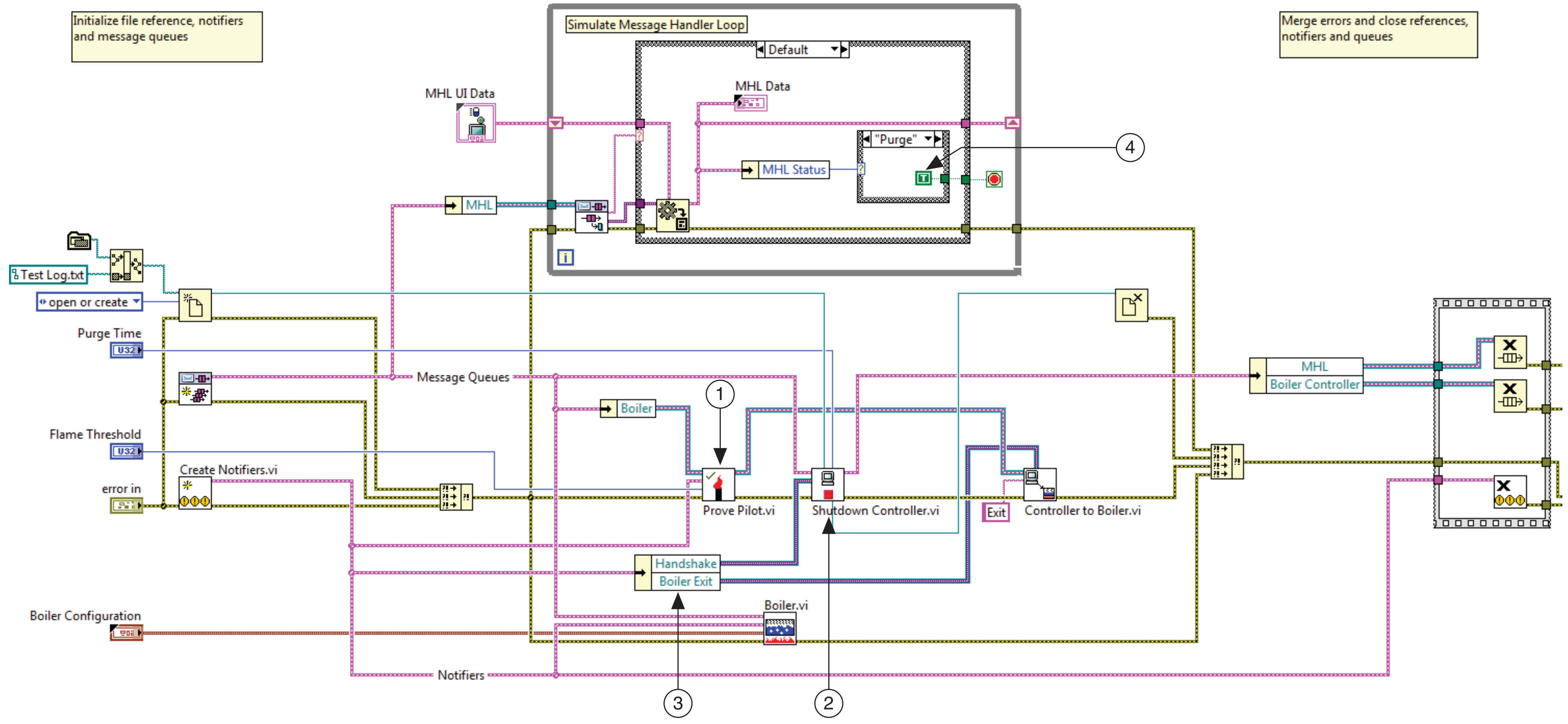
Figure 6-45. Test - Shutdown Controller VI Front Panel



1 **Numeric control**—Right-click the numeric control and select **Representation»U32** from the shortcut menu.

4. Update the block diagram of the Test - Shutdown Controller VI as shown in Figure 6-46.

Figure 6-46. Test - Shutdown Controller VI Block Diagram



1 **Prove Pilot**—Re-wire the connections to the Prove Pilot VI. The Shutdown Controller VI requires that the Pilot Flame Level (%) on the boiler be a non-zero value. The Prove Pilot VI sets this value.

2 **Shutdown Controller**—Wire the **Purge Time** control terminal to the Shutdown Controller VI.

3 Expand the Unbundle By Name function and wire the **Handshake** terminal to the **Handshake Notifier in** input of the Shutdown Controller VI.

4 Create the Purge case as shown and change Boolean constant in the Default case to False.

5. Save the Test - Shutdown Controller VI.

6. Test the Shutdown Controller VI with the Test - Shutdown Controller VI.

   ☐ Open the Boiler VI so you can observe changes to the front panel when you run the test. You must open the Boiler VI before running the test because it is configured to appear when run and close when finished running.

   ☐ Make the following settings on the Test - Shutdown Controller VI:

Control	Value
Flame Threshold	40
Purge Time	2
Pilot Increment (ms)	100
Pilot Decrement (ms)	50

   ☐ Run the Test - Shutdown Controller VI.

   ☐ Verify the following:

   – On the Boiler VI front panel, the **Pilot Flame Level (%)** increments from 0 to 40 in 4 seconds (100 ms per increment).

   – On the Test - Shutdown Controller VI, the **MHL Status** indicator changes to **`Shutdown`**.

   – On the Boiler VI front panel, the **Pilot Flame Level (%)** decrements from 40 to 0 in 2 seconds (50 ms per increment).

   – On the Test - Shutdown Controller VI, the following actions occur:
     The **MHL Status** indicator changes to **`Purge`**.
     The **Primary Fan** LED illuminates.

   – After 2 seconds (the purge time set) the VI stops.

   ☐ Close the Test - Shutdown Controller VI and the Boiler VI.

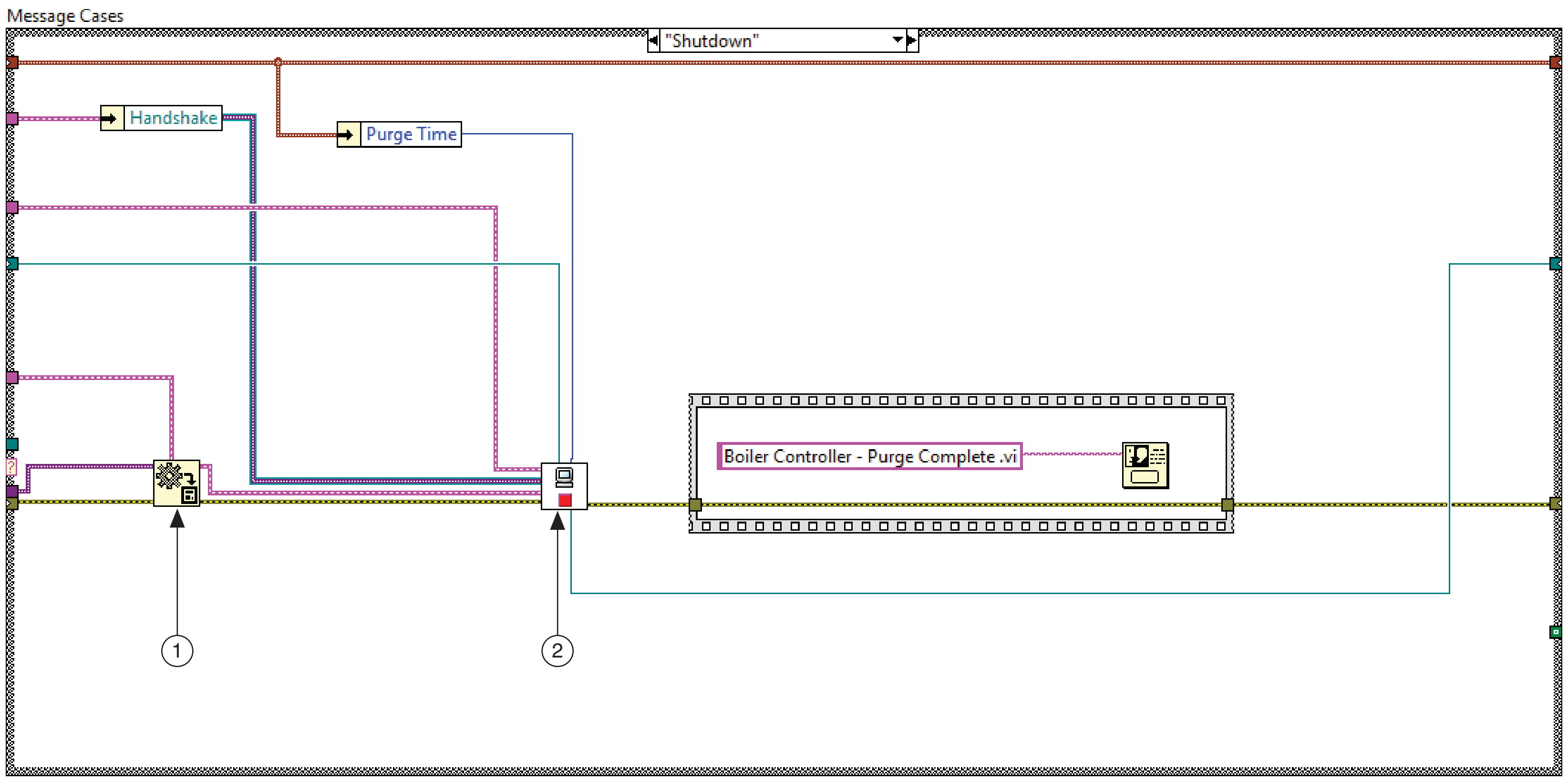
Integrate and Test the Shutdown Controller VI and Purge Complete VI into the Boiler Controller System

1. Integrate the Shutdown Controller VI into the application.

   ☐ Open the Boiler Controller VI.

   ☐ Modify the Shutdown case of the Boiler Controller VI to include the Shutdown Controller VI as shown in Figure 6-47.

Figure 6-47. Adding the Shutdown Controller VI to the Shutdown Case

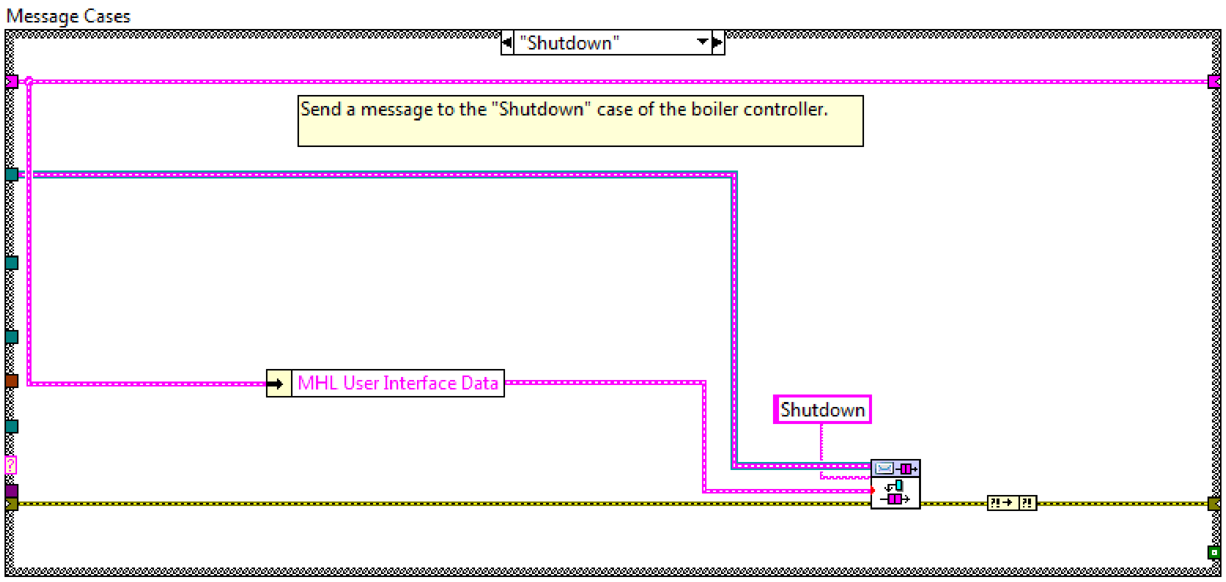


1	**Variant to Data**—Wire the Variant to Data function as shown.
2	**Shutdown Controller**—Wire the Shutdown Controller VI as shown.

☐ Save the Boiler Controller VI.

☐ Modify the Shutdown case of the Main VI to pass MHL User Interface Data as the message data for the Shutdown message as shown in Figure 6-48.

Figure 6-48. Modifying the Main VI Shutdown Case to Use MHL UI Data


Message Cases
"Shutdown"
Send a message to the "Shutdown" case of the boiler controller.
MHL User Interface Data
Shutdown

☐ Save the Main VI.

2. Test the integration of the Shutdown Controller VI in the application.

☐ Run the Main VI and verify that previous functionality still works.

☐ Click the **Stop Boiler** button or the **Simulate Failure** button and verify the following:

– **Fuel Control Valve**, **Stop Boiler**, and **Simulate Failure** buttons are disabled.

– On the Main VI, the **Status** indicator changes to **`Shutdown`**.

– On the Boiler VI front panel, the **Pilot Flame Level (%)** indicator decrements to 0 in about 2 seconds.

- The **Forced Draft Fan** LED turns off.
- On the Main VI, the **Status** indicator changes to **`Purge`**.
- The **Primary Fan** LED turns on.
- The **Boiler Controller - Purge Complete.vi** dialog box appears. Click **OK**.

☐ Click the **Emergency Stop** button.

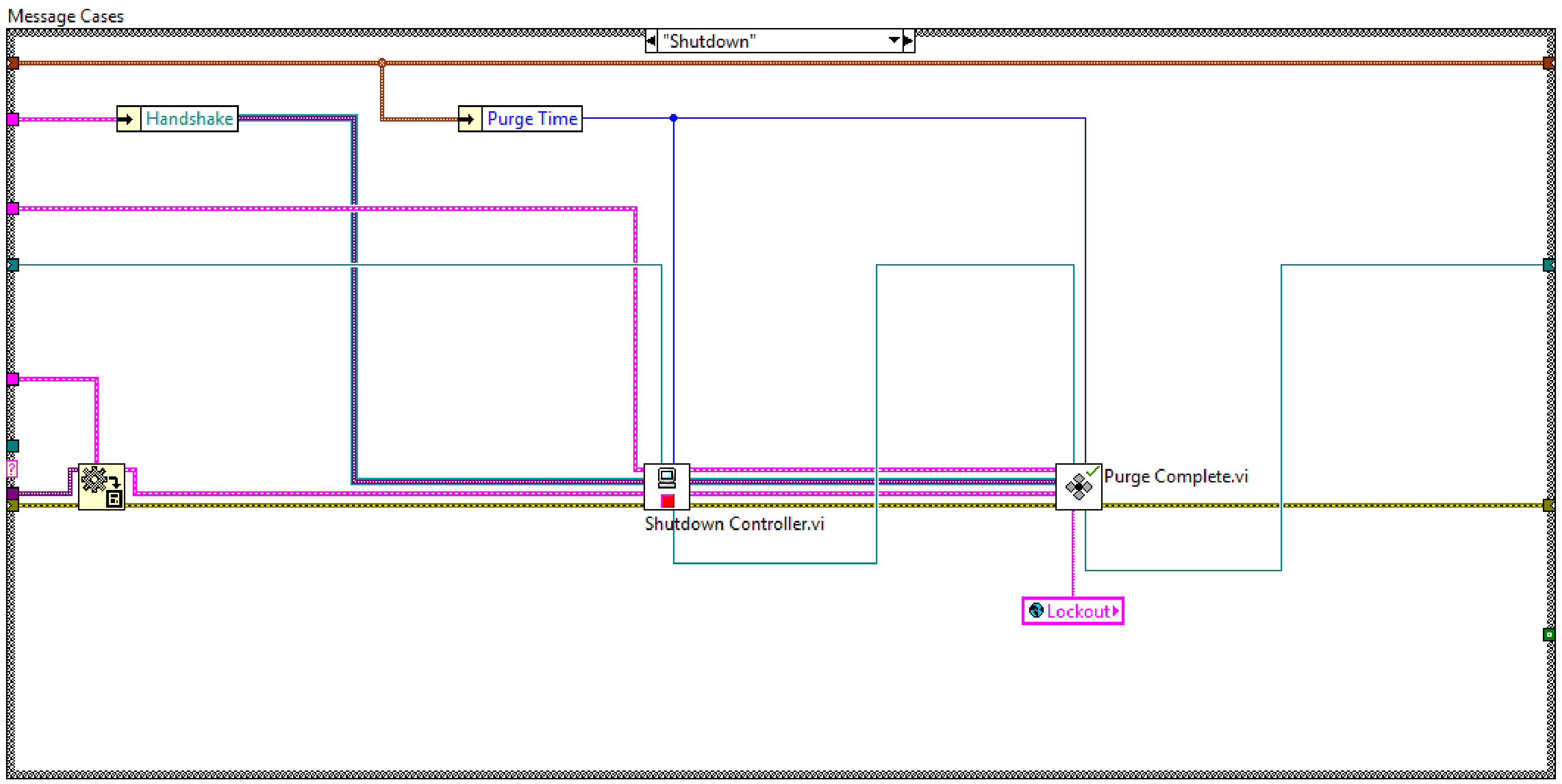
☐ Open `Status Log.txt` and note the **`Start Shutdown Purge`** entry.

3. Integrate the Purge Complete VI into the application.

Note You do not need to test this VI at the module level because you tested and integrated it during Exercise 6-2 when integrating the Start functionality.

☐ Modify the Shutdown case of the Boiler Controller VI to include the Purge Complete VI as shown in Figure 6-49.

Figure 6-49. Adding the Purge Complete VI to the Shutdown Case



☐ Save and close the Boiler Controller VI.

☐ Run `Main.vi`.

- ☐ Verify that previous functionality still works.
- ☐ Click the **Stop Boiler** button.
    - – Notice that after the purge cycle finishes, the primary fan turns off, **Reset** is enabled, and **Status** indicates **`Lockout`**. You can now begin the entire boiler startup cycle again.
- ☐ Click the **Emergency Stop** button.
- ☐ Open `Status Log.txt` and note the new **`Shutdown Purge Complete`** entry.
- ☐ Save and close the Main VI.

Test the Application

You already tested the Main VI for each task as part of completing that task, so you need to test only the executable.

1. Test the executable.
    - ☐ Build the executable from the build specification.
    - ☐ Run `Boiler Controller.exe`.
    - ☐ Verify the behavior of the new functionality.
    - ☐ Click the **Emergency Stop** button to close the application.

End of Exercise 6-5

A

Additional Information and Resources

This appendix contains additional information about National Instruments technical support options and LabVIEW resources.

National Instruments Technical Support Options

Log in to your National Instruments `ni.com` User Profile to get personalized access to your services. Visit the following sections of `ni.com` for technical support and professional services:

- **Support**—Technical support at `ni.com/support` includes the following resources:
  - **Self-Help Technical Resources**—For answers and solutions, visit `ni.com/support` for software drivers and updates, a searchable KnowledgeBase, product manuals, step-by-step troubleshooting wizards, thousands of example programs, tutorials, application notes, instrument drivers, and so on. Registered users also receive access to the NI Discussion Forums at `ni.com/forums`. NI Applications Engineers make sure every question submitted online receives an answer.
  - **Standard Service Program Membership**—This program entitles members to direct access to NI Applications Engineers via phone and email for one-to-one technical support, as well as exclusive access to self-paced online training modules at `ni.com/self-paced-training`. All customers automatically receive a one-year membership in the Standard Service Program (SSP) with the purchase of most software products and bundles including NI Developer Suite. NI also offers flexible extended contract options that guarantee your SSP benefits are available without interruption for as long as you need them. Visit `ni.com/ssp` for more information.

    For information about other technical support options in your area, visit `ni.com/services` or contact your local office at `ni.com/contact`.
- **System Integration**—If you have time constraints, limited in-house technical resources, or other project challenges, National Instruments Alliance Partner members can help. The NI Alliance Partners joins system integrators, consultants, and hardware vendors to provide comprehensive service and expertise to customers. The program ensures qualified, specialized assistance for application and system development. To learn more, call your local NI office or visit `ni.com/alliance`.

You also can visit the Worldwide Offices section of `ni.com/niglobal` to access the branch office Web sites, which provide up-to-date contact information, support phone numbers, email addresses, and current events.

Other National Instruments Training Courses

National Instruments offers several training courses for LabVIEW users. These courses continue the training you received here and expand it to other areas. Visit `ni.com/training` to purchase course materials or sign up for instructor-led, hands-on courses at locations around the world.

National Instruments Certification

Earning an NI certification acknowledges your expertise in working with NI products and technologies. The measurement and automation industry, your employer, clients, and peers recognize your NI certification credential as a symbol of the skills and knowledge you have gained through experience. Visit `ni.com/training` for more information about the NI certification program.