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

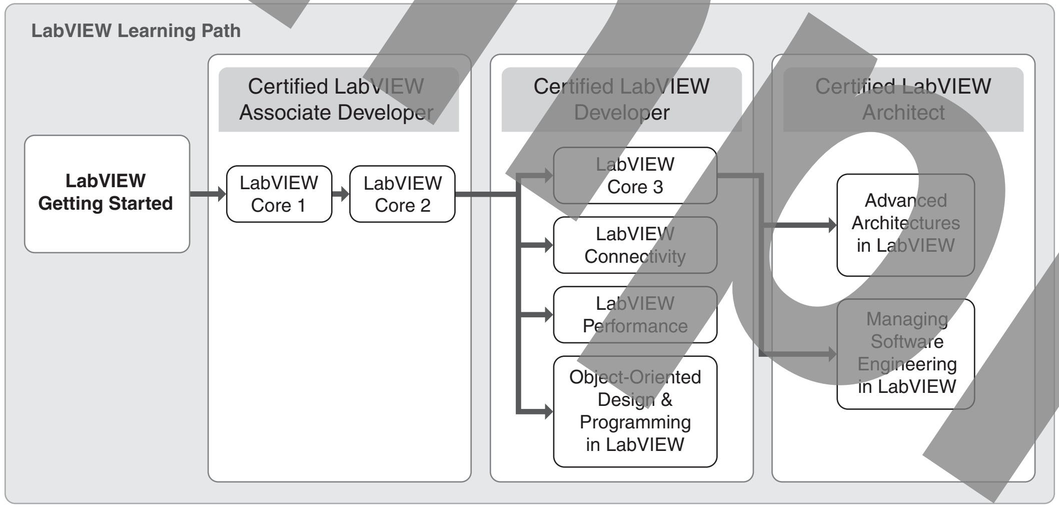
Thank you for purchasing the *LabVIEW Core 3* course kit. This course manual and the accompanying software are used in the three-day, hands-on *LabVIEW Core 3* course.

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.

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:

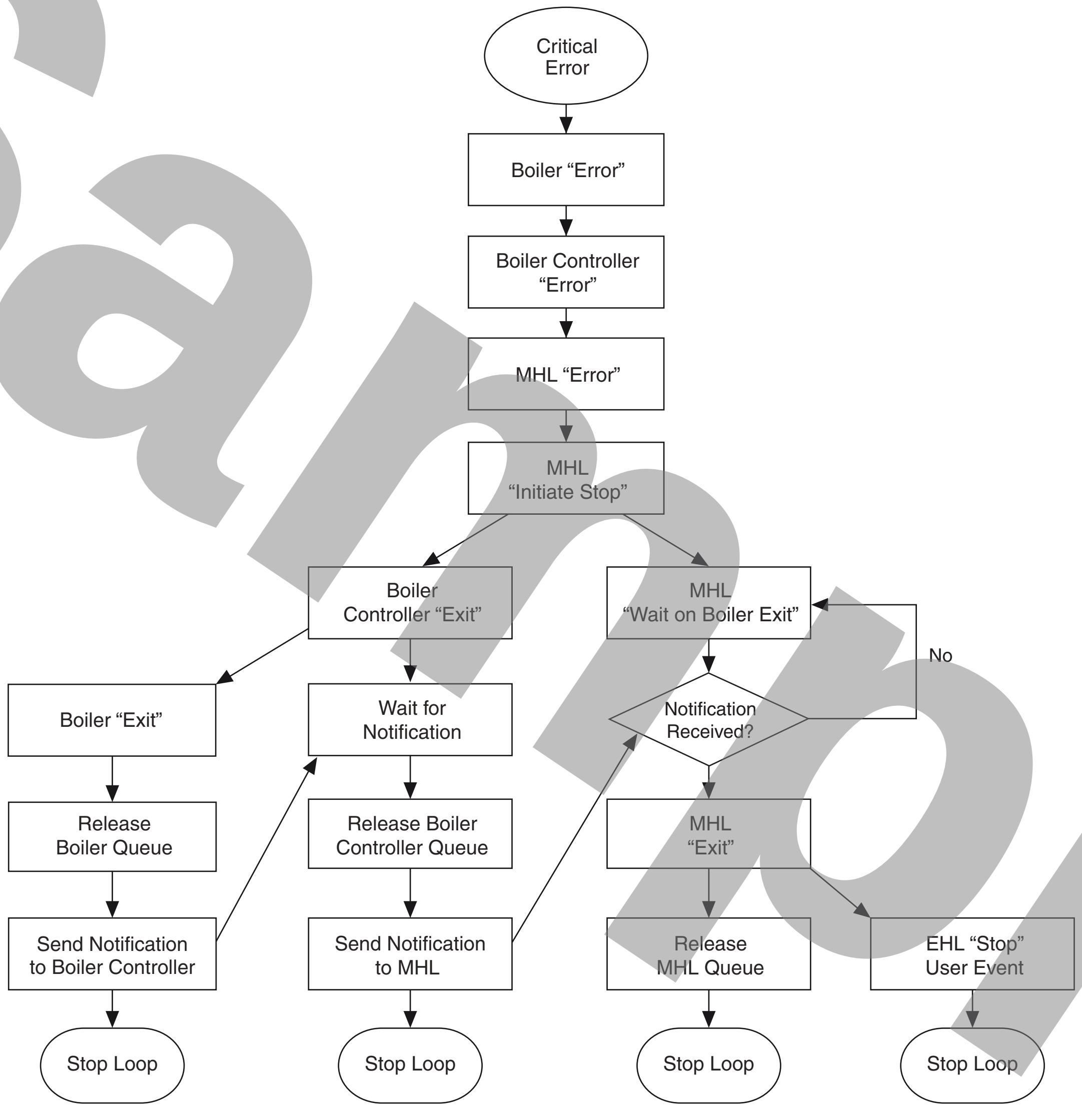
1. Modify `Read Configuration Data.vi` to handle cancellation of the Open File dialog.
2. Modify `Read UI Constants.vi` to handle selection of an invalid INI file.
3. Modify the Default case of the Message Handling Loop and Boiler Controller to handle typos when sending messages.
4. Modify the Message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

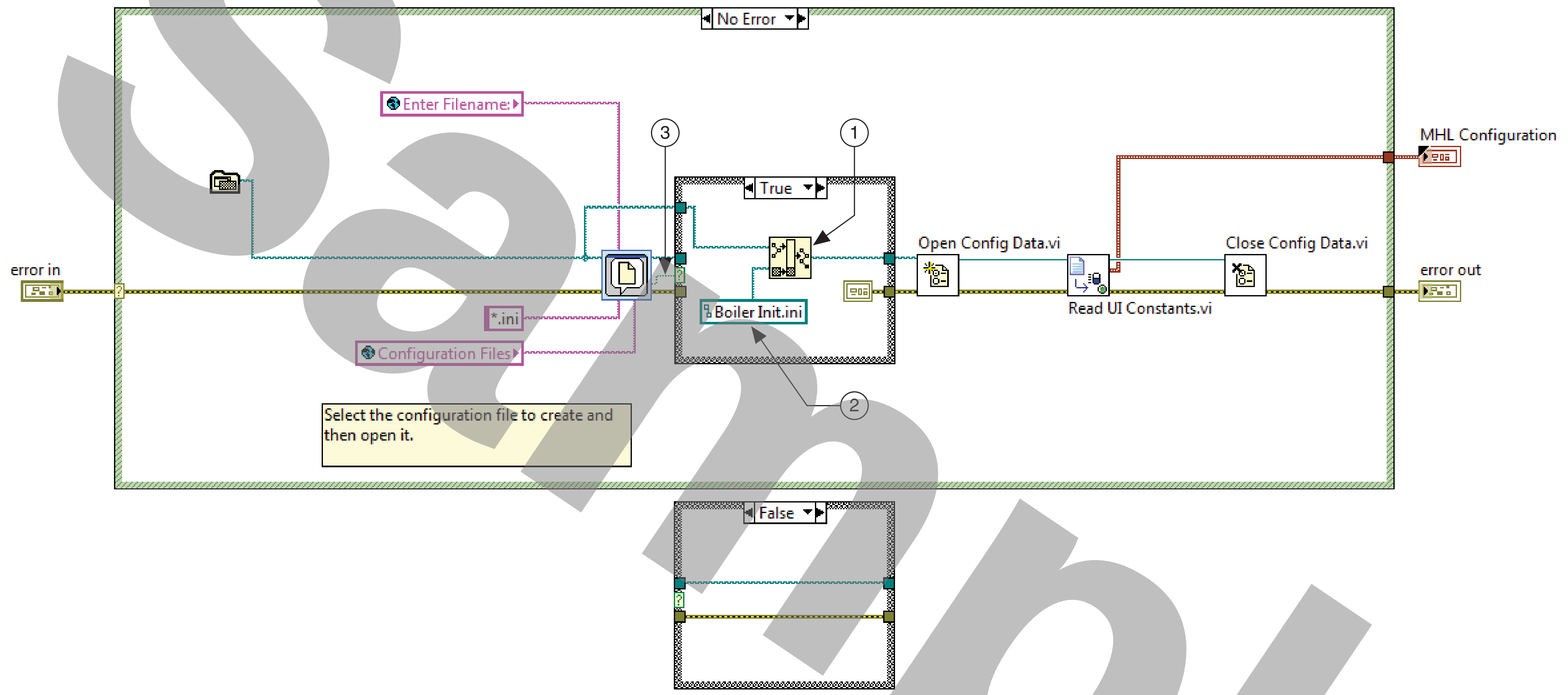
Non-Critical Errors

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.

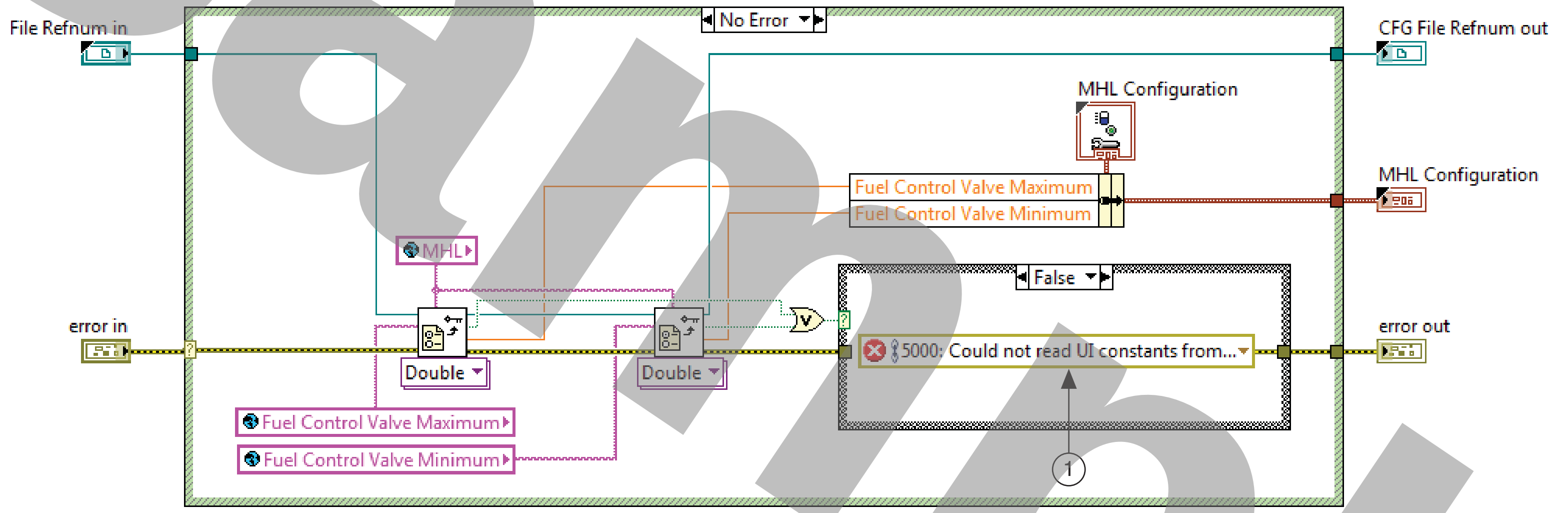
Critical Errors

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



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

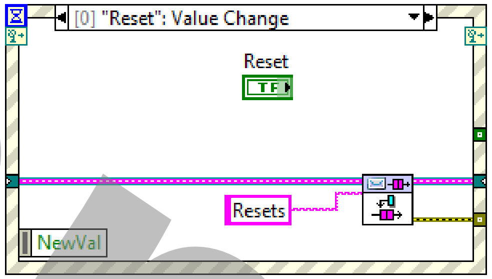
1 Error Ring—Generate a custom error to indicate that the INI file is not valid. This error shuts down the system.

   ☐ 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.

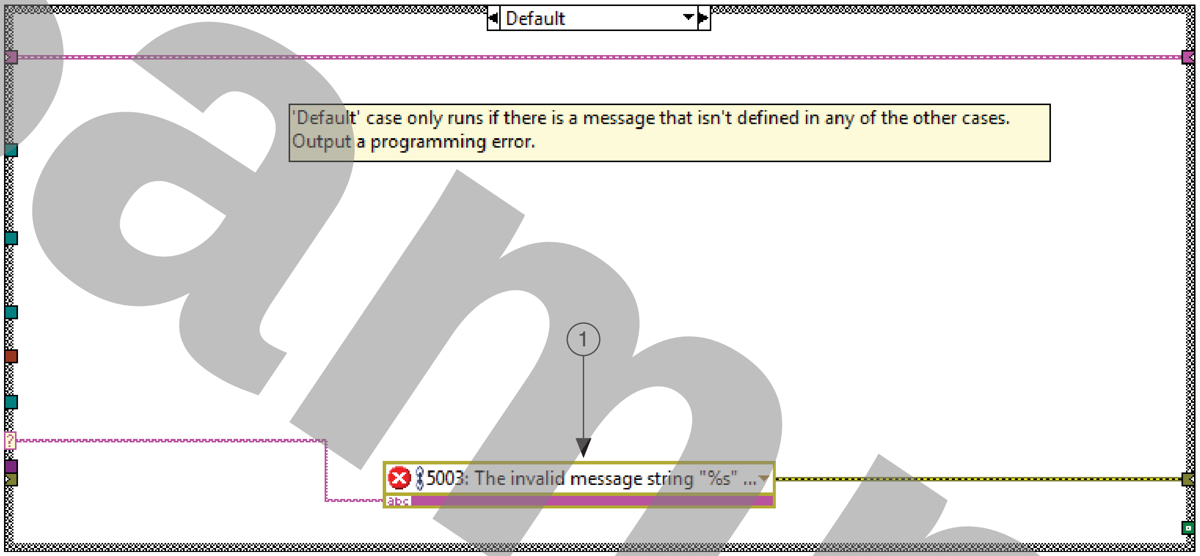
Figure 5-4. Event Handling Loop Reset: Value Change Case with Typo



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.

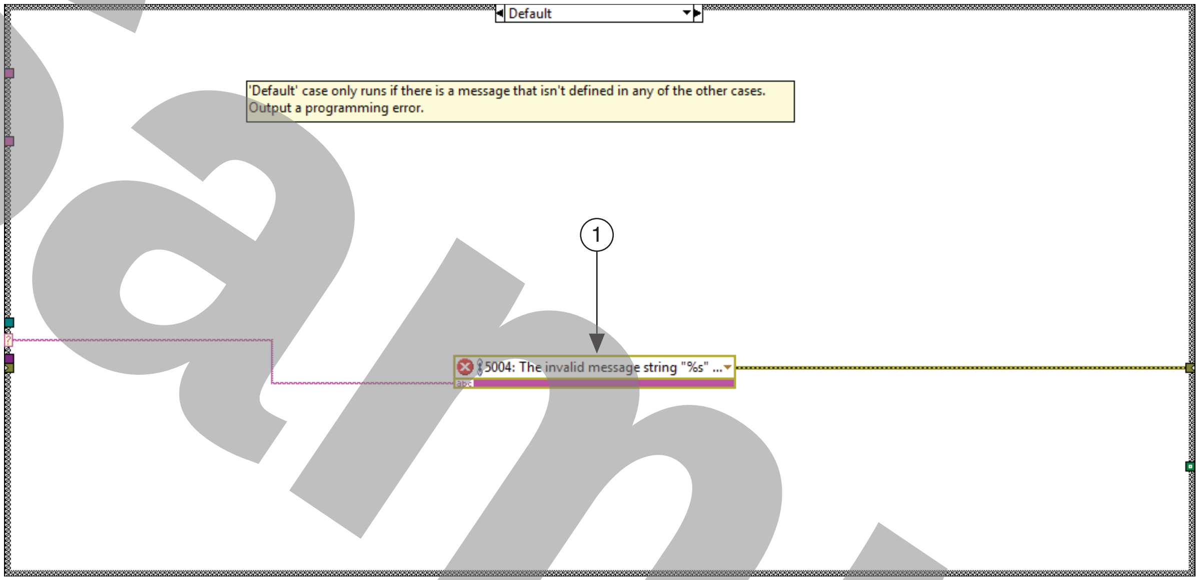
Figure 5-5. Message Handling Loop Default Case with Error Ring


Default
'Default' case only runs if there is a message that isn't defined in any of the other cases. Output a programming error.
1
5003: The invalid message string "%s" ...
abc

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' 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 `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

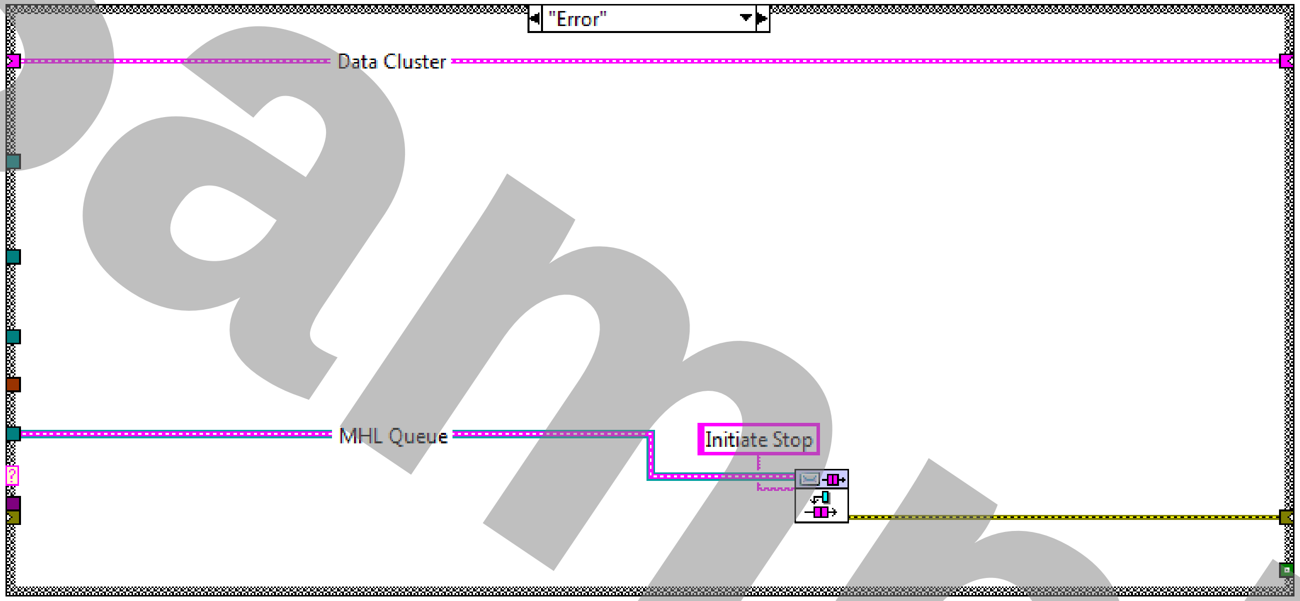
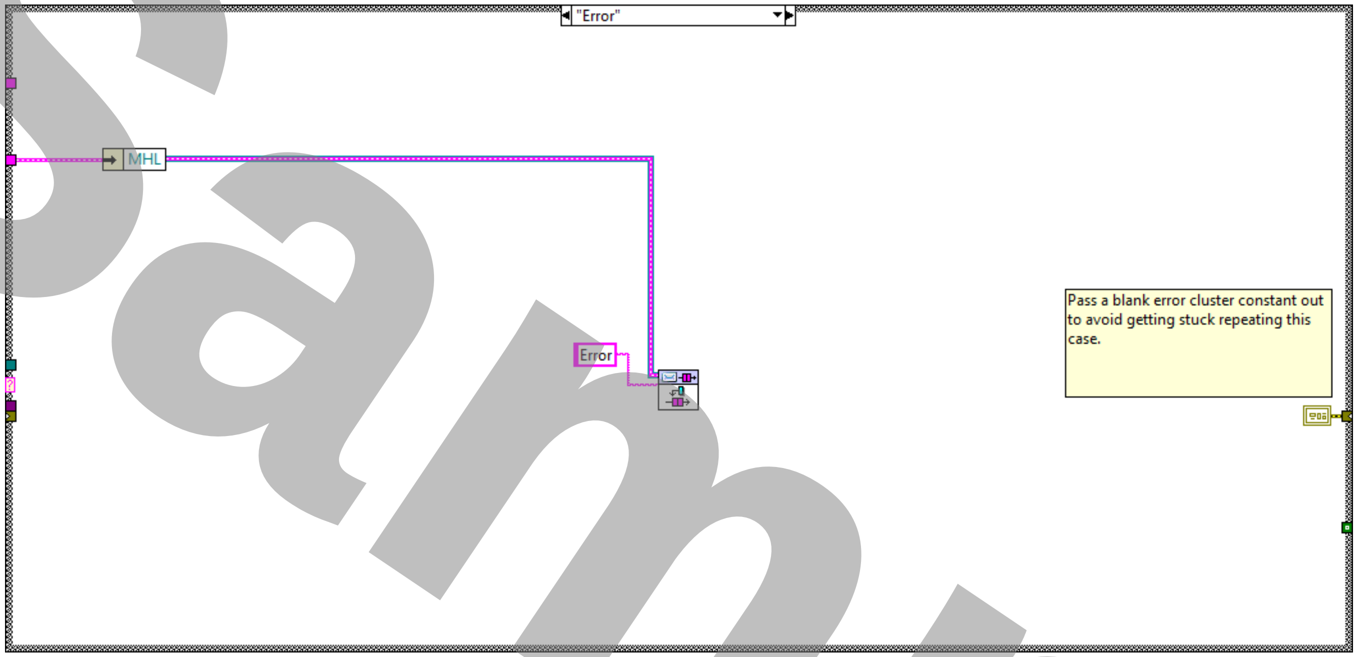


Figure 5-8. Boiler Controller Loop Error Case

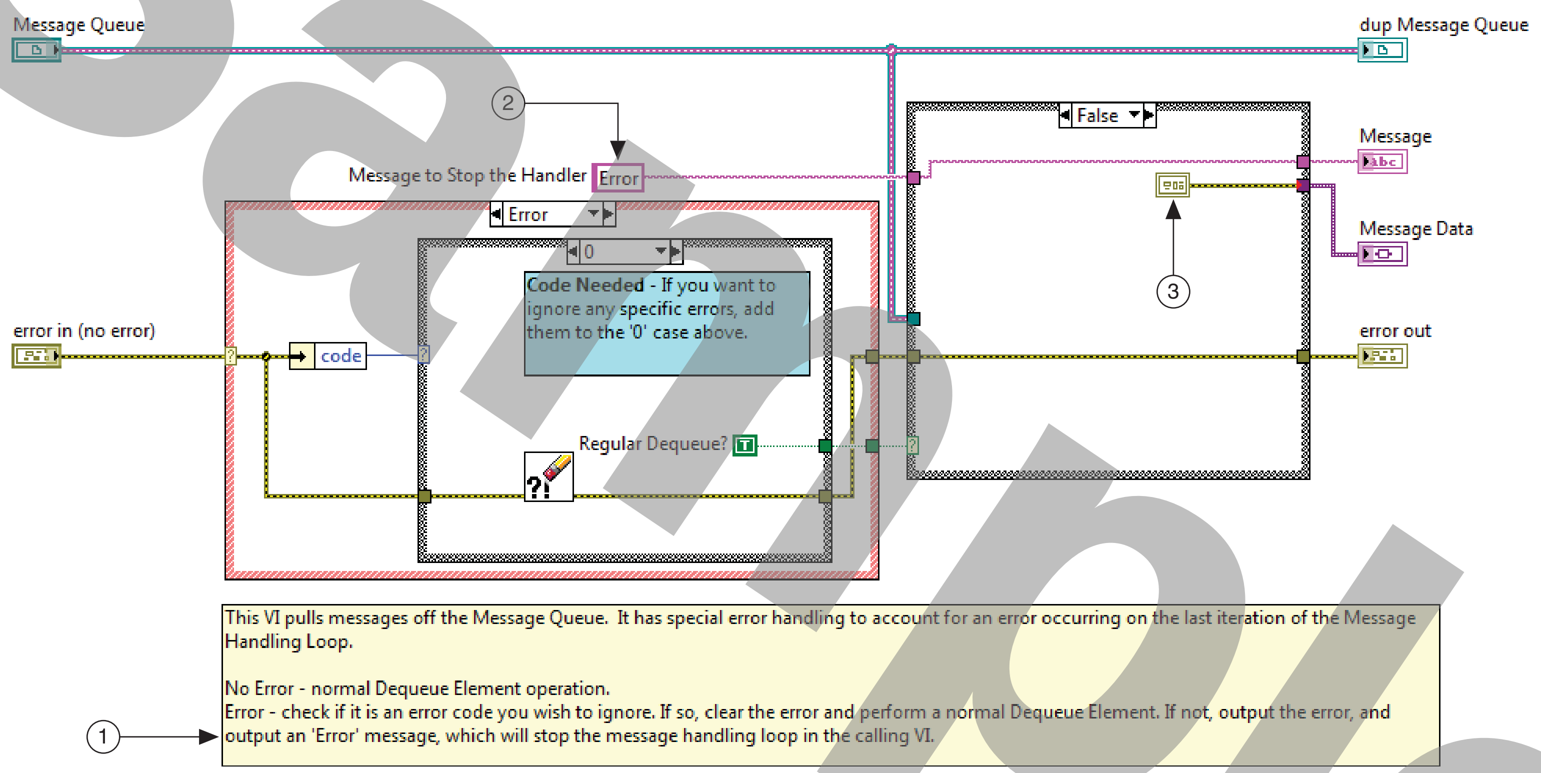


Pass a blank error cluster constant out to avoid getting stuck repeating this case.

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.)

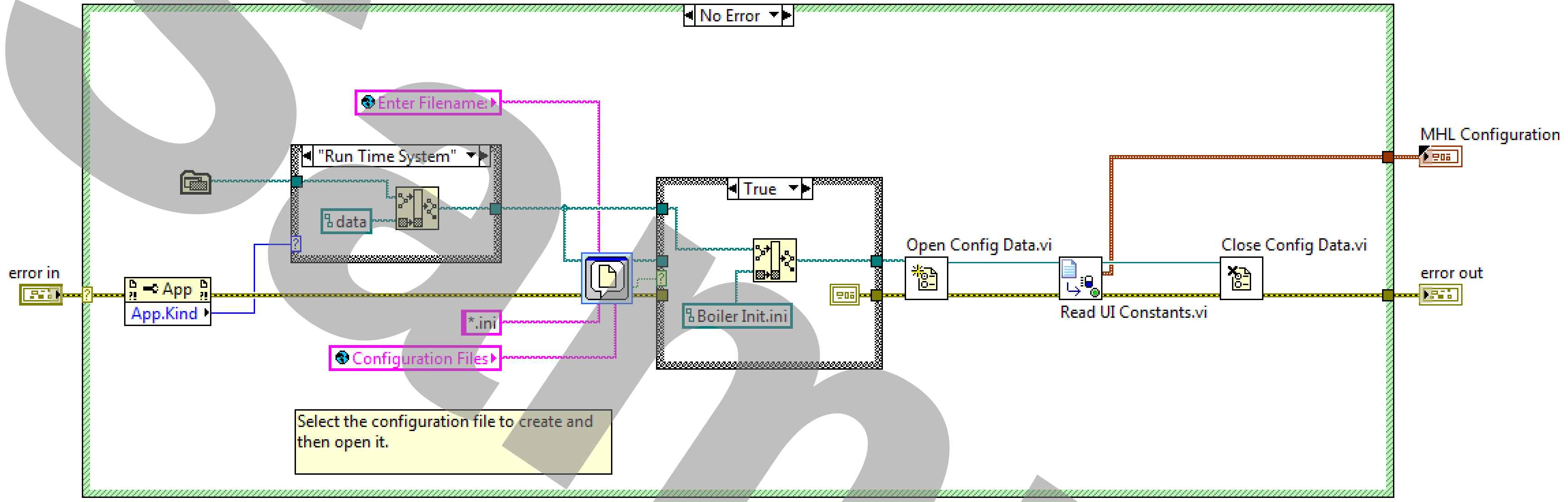
- ☐ The Error case of the Message Handling Loop sends you to the Initiate Stop case, as if you clicked Emergency Stop.

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.

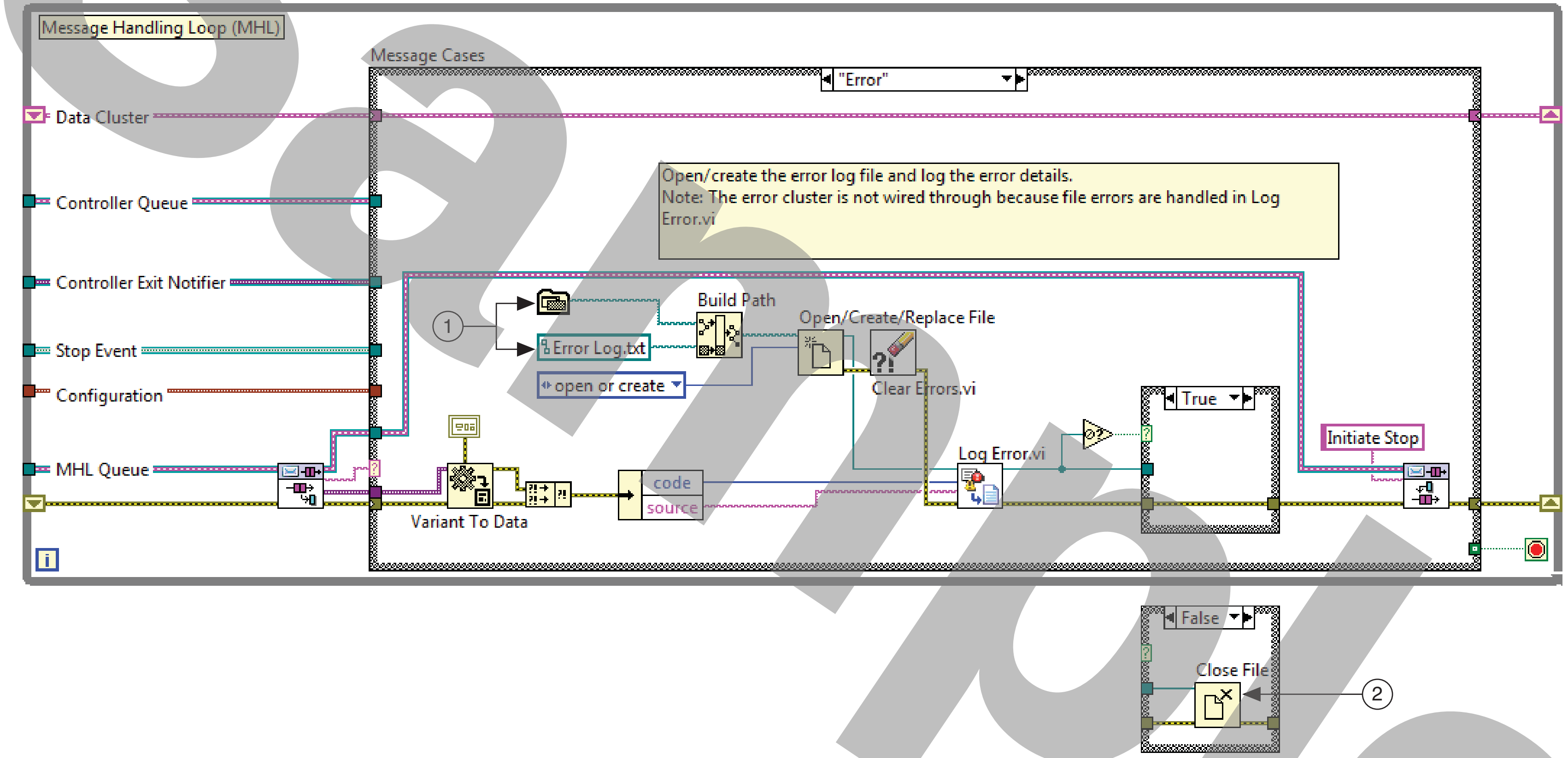
Implementation

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



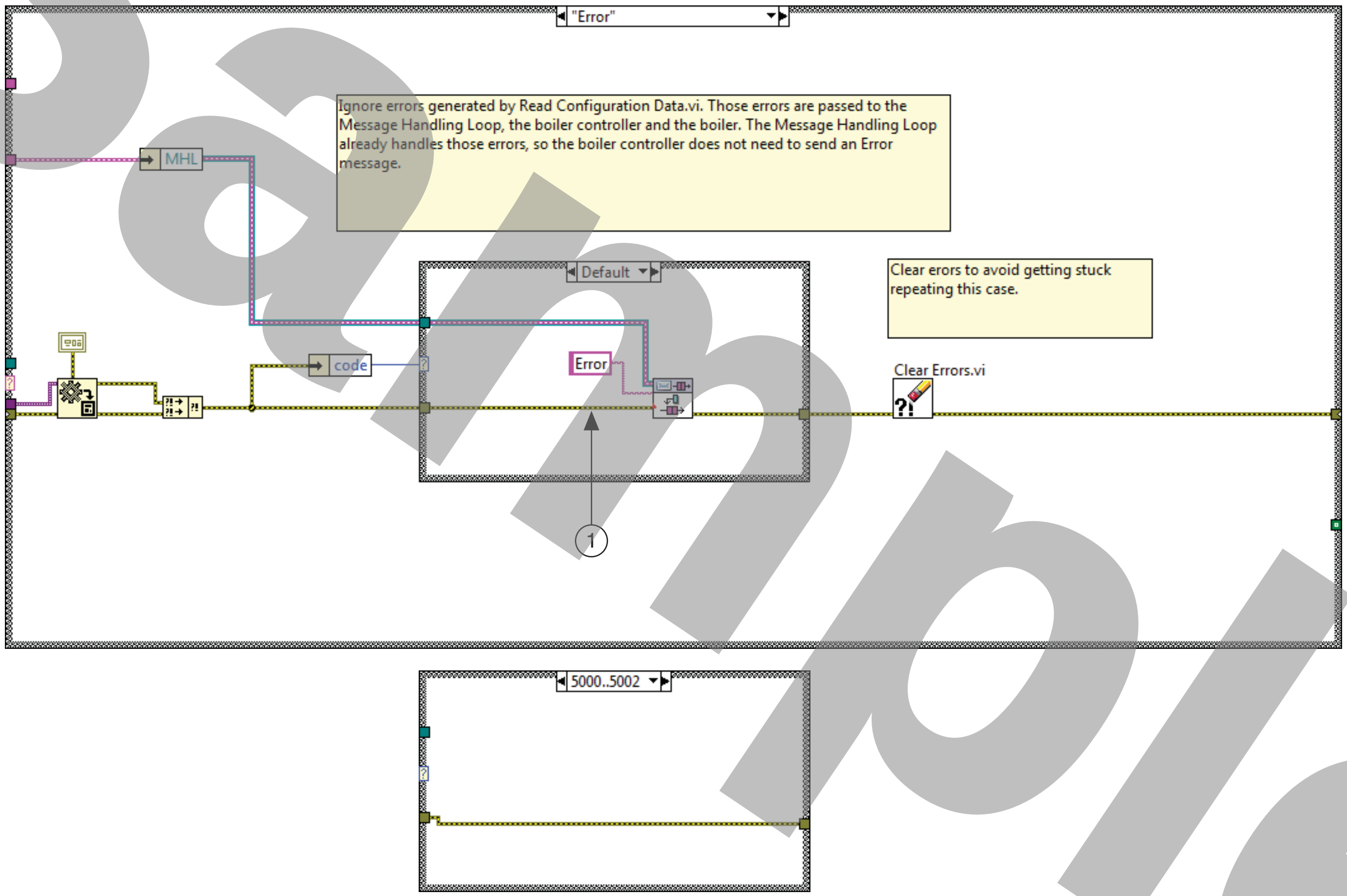
1 Create the error log file in the same directory as the main application.

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

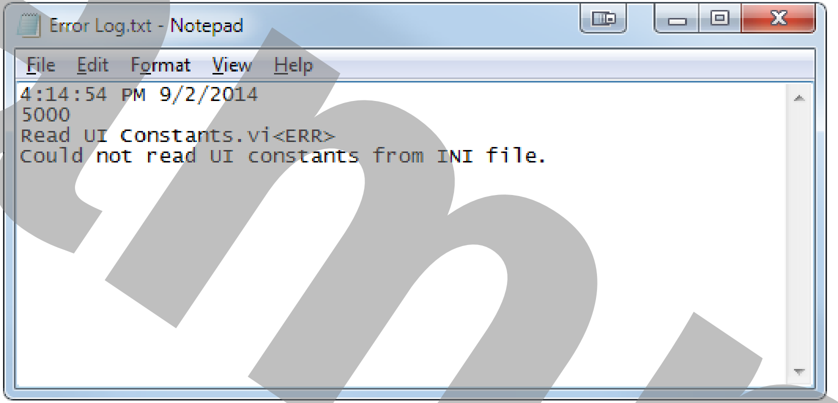
1. Test the VI.

☐ 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