

Using Pega Robotic Automation Diagnostic Logging Tools

1.0

Student Guide

Trademarks

For Pegasystems Inc. trademarks and registered trademarks, all rights reserved. Other brand or product names are trademarks of their respective holders.

For information about the third-party software that is delivered with the product, refer to the third-party license file on your installation media that is specific to your release.

Notices

This publication describes and/or represents products and services of Pegasystems Inc. It may contain trade secrets and proprietary information that are protected by various federal, state, and international laws, and distributed under licenses restricting their use, copying, modification, distribution, or transmittal in any form without prior written authorization of Pegasystems Inc.

This publication is current as of the date of publication only. Changes to the publication may be made from time to time at the discretion of Pegasystems Inc. This publication remains the property of Pegasystems Inc. and must be returned to it upon request. This publication does not imply any commitment to offer or deliver the products or services described herein.

This publication may include references to Pegasystems Inc. product features that have not been licensed by you or your company. If you have questions about whether a particular capability is included in your installation, please consult your Pegasystems Inc. services consultant.

Although Pegasystems Inc. strives for accuracy in its publications, any publication may contain inaccuracies or typographical errors, as well as technical inaccuracies. Pegasystems Inc. may make improvements and/or changes to the publication at any time.

Any references in this publication to non-Pegasystems websites are provided for convenience only and do not serve as an endorsement of these websites. The materials at these websites are not part of the material for Pegasystems products, and use of those websites is at your own risk.

Information concerning non-Pegasystems products was obtained from the suppliers of those products, their publications, or other publicly available sources. Address questions about non-Pegasystems products to the suppliers of those products.

This publication may contain examples used in daily business operations that include the names of people, companies, products, and other third-party publications. Such examples are fictitious and any similarity to the names or other data used by an actual business enterprise or individual is coincidental.

This document is the property of:

Pegasystems Inc.
One Rogers Street
Cambridge, MA 02142-1209
USA
Phone: 617-374-9600
Fax: (617) 374-9620
www.pegasystems.com

CONTENTS

COURSE INTRODUCTION	1
Before you begin	2
Diagnostic logging overview	2
Completing the exercises	4
DIAGNOSTIC LOGGING	5
Introduction to logging basics	6
Logging basics	7
Dissecting a log file	11
Exercise: Dissecting a log file	15
Introduction to automation playback	22
Pega Robotic Automation Playback	23
Using automation playback	28
Log file cleaner	31
Using log file cleaner	32
COURSE SUMMARY	36
Course Summary	37
Diagnostic logging conclusion	37

COURSE INTRODUCTION

This lesson group includes the following lessons:

- Lesson name
- Lesson name

Before you begin

Diagnostic logging overview

In the course, you learn the general concepts of a diagnostic log file from Pega Robotic Automation, dissecting a log file for issue resolution, using the Pega Robotic Automation Playback tool, and using the log file cleaner.

Objectives

After completing this course, you should be able to:

- List the diagnostic log files created
- Describe the basic diagnostic log file structure and content
- Differentiate the various common log file entries for issue resolution
- Dissect a diagnostic run-time log file to determine a resolution
- Define the Pega Robotic Automation Playback feature
- Complete the steps on the playback feature to determine a resolution
- Define the log file cleaner utility
- Explain the steps using the log file cleaner utility

Intended audience

This course is for Pega Robotic Automation architects looking to expand on the diagnostic tools for debugging and issue resolution.

Prerequisites

To succeed in this course, students should:

- Successfully complete the Pega Robotic Automation Architect Essentials course
- Successfully complete the Pega Robotic Automation Architect Assessment

Teaser Text

Simple debugging of a Pega Robotic Automation solution may not always be sufficient. The Diagnostic Logging course introduces another level for debugging solutions both in development and deployed to users by using the diagnostic log files. Dissecting log files, using the automation playback feature, and cleaning log files are included in the course.

Product and version: Pega Robotic Automation Studio version 8.0

Course length: 2 hours

Simple debugging of a Pega Robotic Automation solution may not always be sufficient. The Diagnostic Logging course introduces another level for debugging solutions both in development and deployed to users by using the diagnostic log files for issue resolution.

Completing the exercises

When learning new concepts or skills, there is no substitute for learning by doing. This course includes exercises that provide practical, hands-on experience to help you apply your new skills immediately. The exercises help reinforce the learning objectives of each lesson.

Exercise environment

Download the following files from the related content section:

File	Purpose
Log File Training	This zip file contains the supporting log files and solution.
Pega Robotic Log File Cleaner	This zip file contains the Log File Cleaner utility.
Diagnostic Logging Exercise Guide	Download the exercise guide to complete the course separately from Pega Academy.
Diagnostic Student Guide	Download the student guide to complete the course separately from Pega Academy.

To successfully complete the exercises within this course, you must have Pega Robotic Automation Studio installed on your computer. If Pega Robotic Automation Studio is not installed, you must complete the Pega Robotic Automation Architect Essentials course and Assessment first.

DIAGNOSTIC LOGGING

This lesson group includes the following lessons:

- Lesson name
- Lesson name

Introduction to logging basics

In this lesson, the general concepts of a diagnostic log file from Pega Robotic Automation, common log entries, and how to dissect a log file for issue resolution are covered.

After this lesson, you should be able to:

- List the diagnostic log files created
- Describe the basic diagnostic log file structure and content
- Differentiate the various common log file entries for issue resolution
- Dissect a diagnostic run-time log file to determine a resolution

Logging basics

Pega Robotic Automation Studio and Pega Robotic Runtime both provide a mechanism for recording events during run time. The **RuntimeLog.txt** contains everything that happened while the solution ran in development or in deployment. The **RuntimeLog.txt** is the primary diagnostic log file used by developers for issue resolutions.

Pega Robotic Automation Studio also generates other log files for review in case of errors or issues from other components of Studio.

Those log files are listed in the following table:

Log File	Description
StudioLog.txt	This includes diagnostic messages from the execution of Studio.
RuntimeLog.txt	This includes diagnostic messages from the execution of Runtime.
OSCLog.txt	This includes messages generated when building a Studio project.
OSDLog.txt	This includes messages generated when creating a Studio deployment package for a project.
Trace	This enables generating diagnostic traces for generating detailed application messages for Studio, Runtime, or both. Note: Use the Trace publisher under advisement of Pega Robotics Studio Support.

Studio also offers a Log4Net option for each of the standard generated log files. The Log4Net option gives you the ability to modify the first three-column structure, retention of log files, and appending new data.

You control and configure the **RuntimeLog.txt** for development from the **Tools > Options** menu in Studio. You can also use the **RuntimeConfig.xml** file found in user's **...Appdata\Roaming\OpenSpan** directory.

RuntimeLog structure and content

Before attempting to research and dissect a diagnostic log to resolve an issue, you should know the basic structure and content of the **RuntimeLog.txt**.

Important things that can be found at the beginning of a log file:

- The computer, from where Pega Robotic Automation Studio solution ran, that created the log file
- The date/time that the log started

The **RuntimeLog.txt** file consists of eight columns, listed in the following table:

Column	Description
MsgType	The options are Info, Message, Error, and Verbose.
TimeStamp	This is the exact time that the line was logged.
Thread	This is the thread number that the event was logged on.
State	STA or MTA — this describes if a single-threading or multi-threading model was used.
Category	This describes what activity was occurring when the log line was produced. Options include Matching, Automation, Web Adapter, Windows Adapter, Exception, and Window Message.
DesignComponent	This is a control name or an automation name that produced the log line. This column can be blank.
Component	This is a factory name or control name that produced the log line. This column can be blank.
Message	This is the message that the developer wanted to communicate. It is the reason for the log line.

The first 100 log lines contain information to verify what solution ran and from where it ran. There is also a complete file listing within the directory that Pega Robotic Runtime is running to verify file time, date, and version information.

Common log file entries

Pega Robotic Runtime generates the specific messages in the **RuntimeLog.txt** depending upon how an automation runs.

Runtime logs every execution and data path before and after they execute as well as when Runtime matches and detaches an application control.

The six most common log entries are listed in the following table:

Log Entry	Description
ExecutionLink From	Written before a yellow automation link executes
Executed From	Written after the automation stack unwinds

propagating From	Written before a blue data link executes
propagated From	Written after a blue data link executes
Control is matched	Written when the application control's isCreated property changes to True
Control is detached	Written when the application control's isCreated property changes to False

An automation works with the call stack. As new operations occur, Runtime adds them to the top of the stack. When the automation completes, the stack unwinds. This is evident in the log file as the majority of the *ExecutionLink* lines occur at the beginning of the file, and the *Executed From* lines occur at the end. The call stack is interesting but not pertinent in the majority of logs that you are analyzing. To minimize confusion, focus your attention on the *ExecutionLink* lines.

The *Control is matched* message is equivalent to saying that a control is created in an automation. The *Control is detached* message is equivalent to saying the control is destroyed. Detached and Destroyed are synonymous. Searching log files for *Control is matched* and *Control is detached* is an easy way to see how the user navigated the application. When a control is created, the log generates the *Control is matched* message and the control's Created event fires. Similarly, when a control is destroyed, the log generates the *Control is detached* message and the control's Destroyed event fires.

Dissecting a log file

Analyzing a verbose Runtime log file poses a challenge because everything that happens while Pega Robotic Runtime runs a solution is in the log file.

These files may be long, and you need good text editor that can handle large text files to dissect a log file. Notepad++ and TextPad are excellent editors that easily allow you to manipulate the data in the file. Both are available as a free download. The exercise for this course uses Notepad++ as the text editor.

The easiest way to work with a log file is to copy the needed data into a smaller and more manageable file. You can accomplish this by bookmarking or marking log entries that contain keywords, copying and pasting all bookmarked entries in a new or empty text file. For example, if you are trying to figure out a matching error, you could bookmark all lines that contain the name of your control, copy them to an empty file, and analyze this subset of lines.

Dissecting a log file

Dissecting a diagnostic log file is essential to resolving issues that occur during development and run time. A text editor such as Notepad++ or TextPad makes log analysis easier. The following procedure uses Notepad++ as the text editor. Depending on your choice of text editor, the steps searching and marking text may be different.

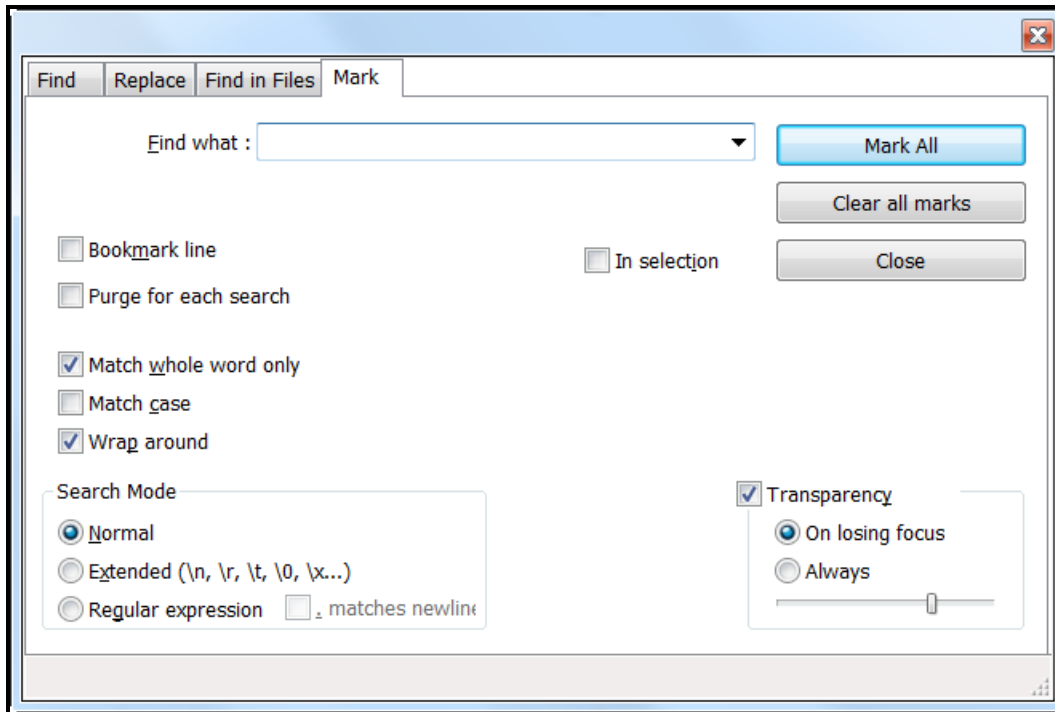
Follow these steps to dissect a log file.

1. From the user's AppData\Roaming\OpenSpan directory, copy the **RuntimeLog.txt** file to a local directory.
2. If necessary, use the Log File Cleaner utility to clean the log file of sensitive information.
3. From the Start menu, launch the text editor.
4. Using the Solution Explorer, determine the project item requiring investigation.
5. In the text editor, use the Search and Mark functions to locate all instances of the appropriate project item.
6. In the text editor, create a new text file. If necessary, save the new text file to a local directory.
7. From the results from the Search, copy and paste the log line entries into the new text file.
8. Review the log entries and times paying attention to the common log file entries for automation links, data links, and control matching.
9. Determine next steps based on findings.

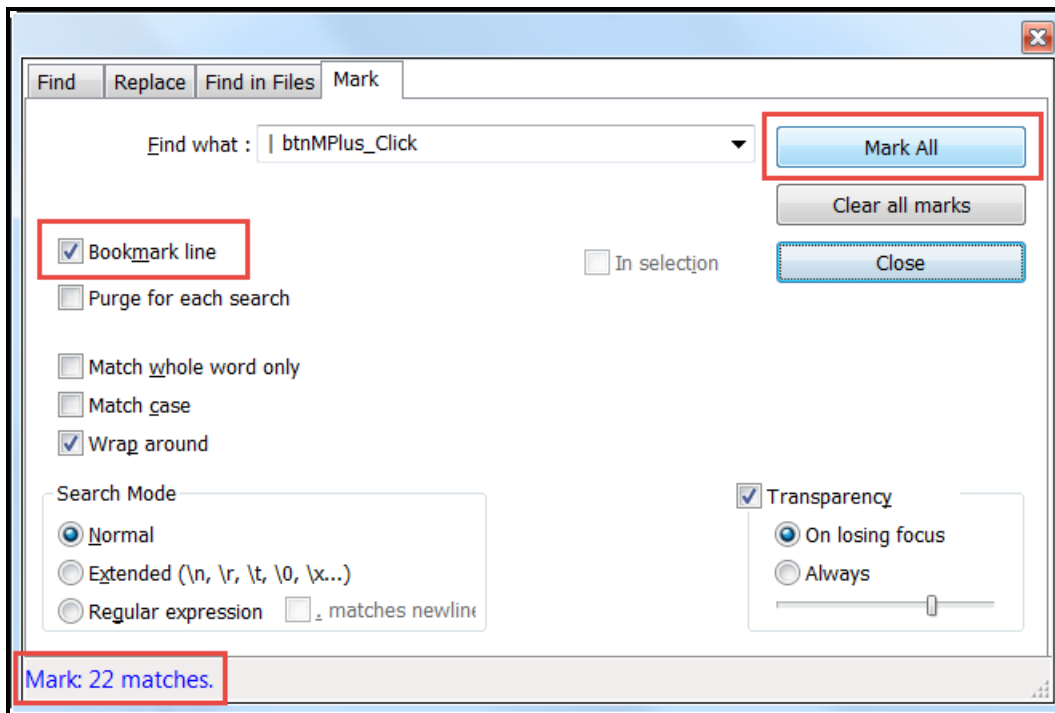
Marking text in Notepad++

Follow these steps on marking text in Notepad++ (assuming you have launched Notepad++ and opened the log file in the application).

1. From the menu, select **Search > Mark...**. The Search window is displayed with the **Mark** tab activated.

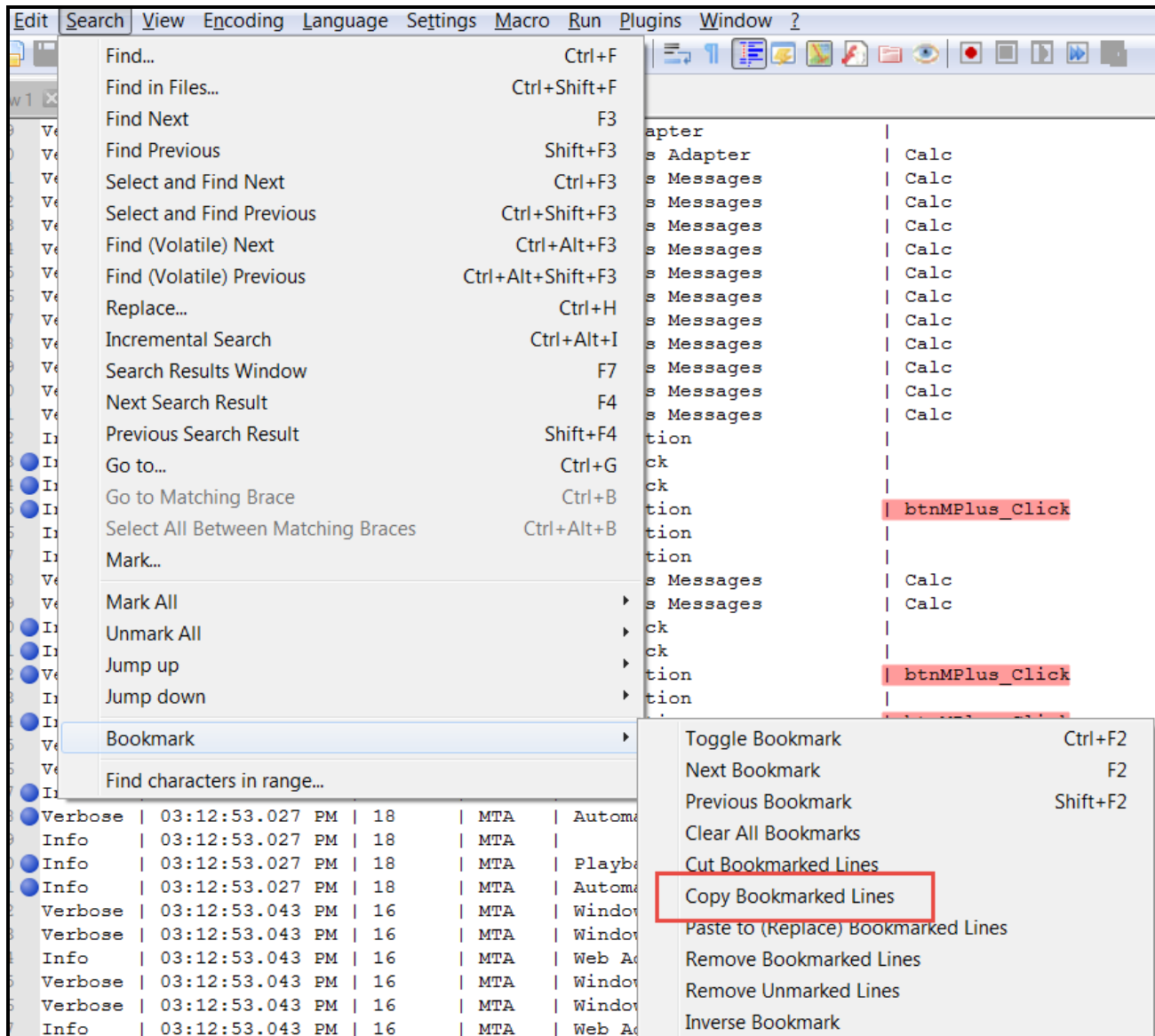


2. In the **Find What** field on the Search window, enter the text to search.
3. In the Search window, select the **Bookmark all** check box and click **Mark All**. The Search window updates with a status of matches from the search.



4. In the Search window, click **Close**. The Search window closes. The searched text in the log entries are displayed as highlighted, and the bookmarked lines have a blue dot in front.

5. From the menu, select **Search > Bookmark > Copy bookmarked lines**.

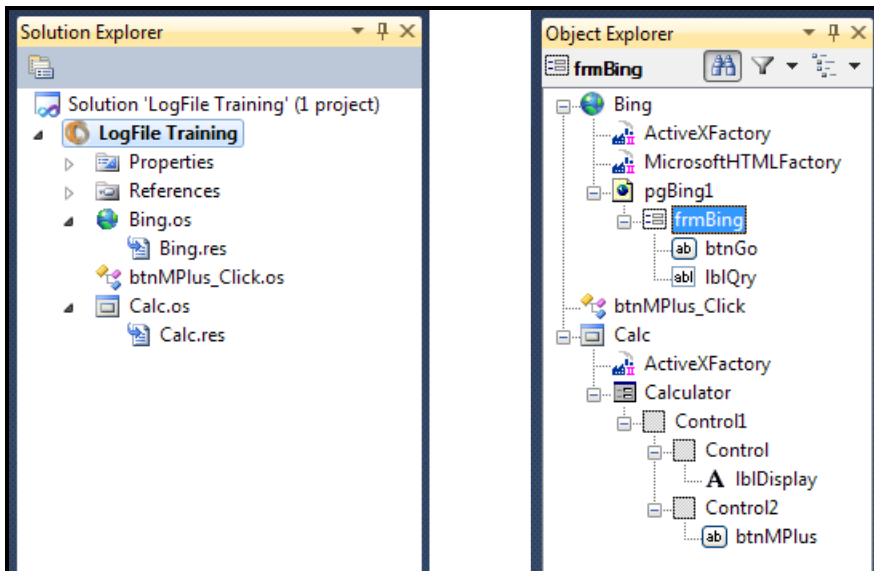


6. In the new text file, right-click and paste the bookmarked lines. The log entries appear in the new text file.
7. From the menu, select **File > Save as**. The Save As window is displayed.
8. On the Save As window, navigate to a local directory and enter a file name. Click **Save**. The window closes and the file appears in the selected local directory for later use.

Exercise: Dissecting a log file

Scenario

Another developer asks your help with a problem. The developer created a solution that performs a Bing search when the M+ button is pressed on calc.exe. Their solution has a Windows Application (calc.exe) and a Web Application (www.bing.com). The Solution Explorer and Object Explorer look like the following images:



The solution works for the first run, but fails on the second run. The developer cannot find the issue and has forwarded the log to you for help.

Your assignment

From the Related Content section, download the Log File Training zip file and copy the **RuntimeLog.txt** to a local directory. To complete the exercise, you need a text editor installed, such as Notepad++ or TextPad. The Detailed steps use Notepad++.

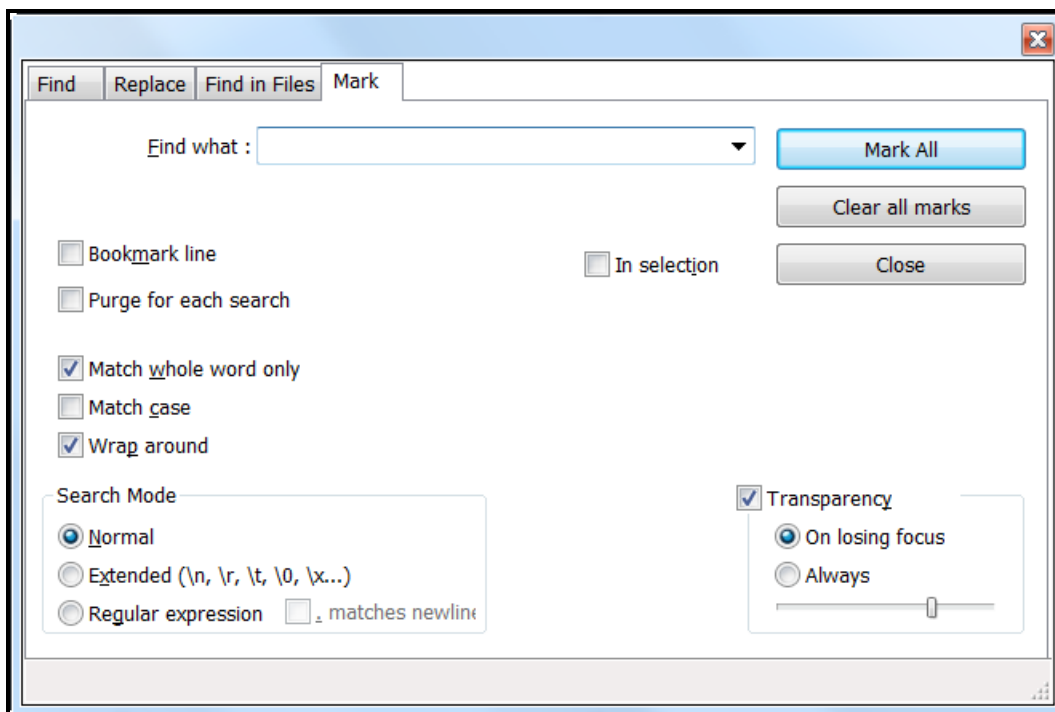
Using the **RuntimeLog.txt** and text editor, dissect the log file to analyze:

- The btnMPlus_Click automation in a new text file to determine if the all the automation links executed correctly
- The blQry control in a new text file to determine the control's availability for the automation

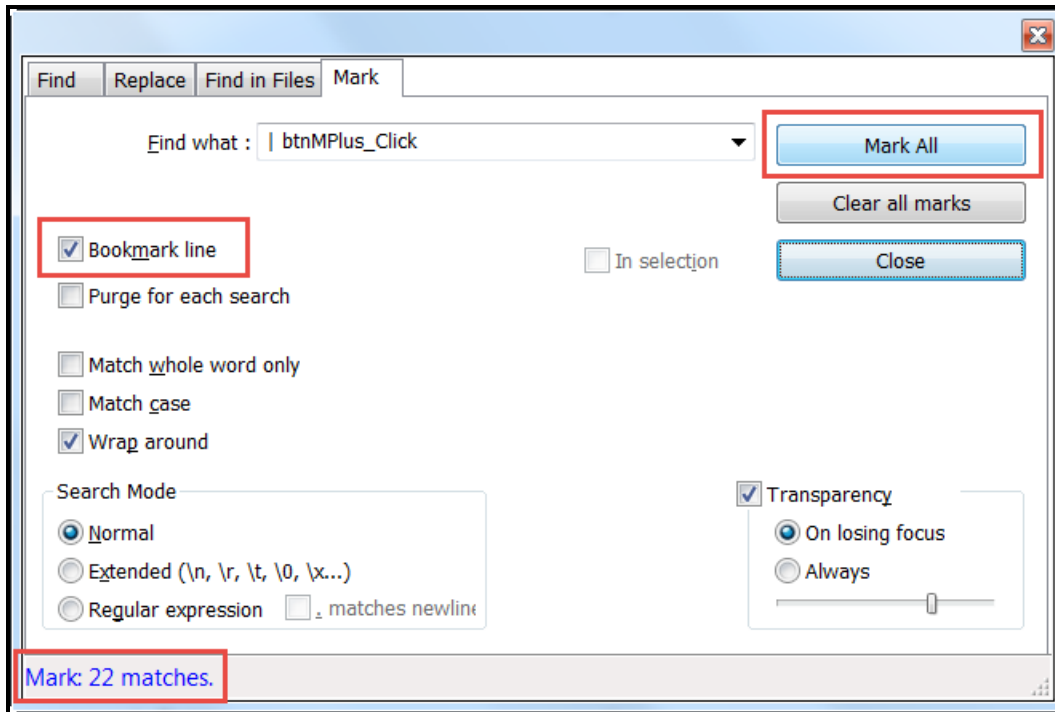
Detailed steps

Follow these steps to dissect the **RuntimeLog.txt** to determine a solution for the btnMPlus_Click automation.

1. From the Start menu, launch Notepad++.
2. From the menu, select **File > Open**. The File Open window is displayed.
3. From the File Open window, navigate to the local directory of the **RuntimeLog.txt** and click **Open**. The file opens in a new tab Notepad++.
4. From the menu, select **Search > Mark....** The Search window is displayed with the **Mark** tab activated.

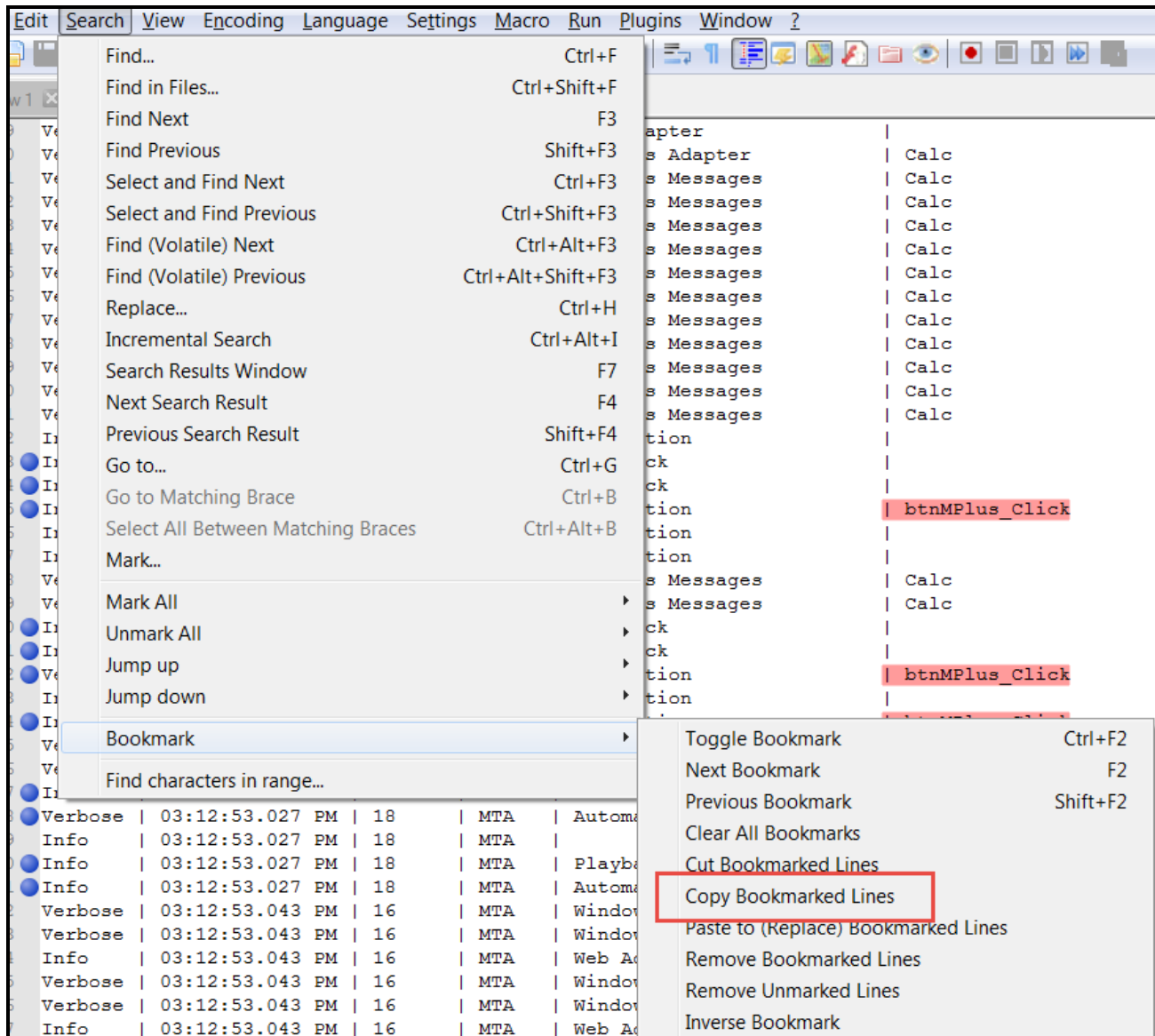


5. On the Search window, in the **Find What** field, enter | **btnMPlus_Click**.
6. In the Search window, click **Mark All** and select the **Bookmark line** check box. The Search window updates with a status of matches from the search.



7. In the Search window, click **Close**. The Search window closes. The searched text in the log entries are displayed as highlighted, and the bookmarked lines have a blue dot in front.

8. From the menu, select **Search > Bookmark > Copy bookmarked lines**.



9. From the menu, select **File > New**. A blank file opens in a new tab in Notepad++.
10. In the new text file, right-click and paste the bookmarked lines. The log entries appear in the new text file.

The pasted excerpt contains two attempts at running the btnMPlus_Click automation. The user reports that the first one was successful. It started at 03:12:52.996 and ended at 03:12:53.074. The second attempt started at 03:13:00.856. By going to the bottom line in the excerpted log, you see

the problem. The automation stopped at 03:13:00.872, when moving data from Calc.lblDisplay.Text to Bing.lblQry.Text. There is an *ExecutionLink From* entry for moving the data but no *Executed From* entry.

```
03:12:52.996 PM | Automation: btnMPlus_Click - ExecutionLink From: Calc.btnMPlus.Click To: Calc.lblDisplay.Properties ID: 46ad0cf5-62f5-4d41-b9b9-2d099f4a2965;TimeStamp=03:12:52.996;AutxId=Automator-8D08D8C847A0FA6
03:12:52.996 PM | Link^SessionId=46ad0cf5-62f5-4d41-b9b9-2d099f4a2965;TimeStamp=03:12:52.996;AutxId=Automator-8D08D8C847A0FA6
03:12:53.012 PM | End^SessionId=46ad0cf5-62f5-4d41-b9b9-2d099f4a2965;TimeStamp=03:12:53.012
03:12:53.012 PM | Automation: btnMPlus_Click - Executed From: Calc.btnMPlus.Click To: Calc.lblDisplay.Properties ID: 46ad0cf5-62f5-4d41-b9b9-2d099f4a2965;TimeStamp=03:12:53.012;AutxId=Automator-8D08D8C847A0FA6
03:12:53.012 PM | Automation: btnMPlus_Click - ExecutionLink From: Calc.lblDisplay.Properties To: Bing.lblQry.Properties ID: 46ad0cf5-62f5-4d41-b9b9-2d099f4a2965;TimeStamp=03:12:53.012;AutxId=Automator-8D08D8C847A0FA6
03:12:53.027 PM | Automation: btnMPlus_Click propagating From: Calc.lblDisplay.Text To Bing.lblQry.Text Value: 3.14
03:12:53.027 PM | Automation: btnMPlus_Click propagated From: Calc.lblDisplay.Text To: Bing.lblQry.Text
03:12:53.027 PM | Link^SessionId=46ad0cf5-62f5-4d41-b9b9-2d099f4a2965;TimeStamp=03:12:53.027;AutxId=Automator-8D08D8C847A0FA6
03:12:53.027 PM | Automation: btnMPlus_Click - ExecutionLink From: Bing.lblQry.Properties To: Bing.btnGo.PerformClick() ID: 46ad0cf5-62f5-4d41-b9b9-2d099f4a2965;TimeStamp=03:12:53.027;AutxId=Automator-8D08D8C847A0FA6
03:12:53.074 PM | Automation: btnMPlus_Click - Executed From: Calc.lblDisplay.Properties To: Bing.lblQry.Properties ID: 46ad0cf5-62f5-4d41-b9b9-2d099f4a2965;TimeStamp=03:12:53.074;AutxId=Automator-8D08D8C847A0FA6
03:13:00.856 PM | Start^SessionId=a1ab4c5d-16ff-4b7c-b2c1-23db05bb8dbf;TimeStamp=03:13:00.856;AutxId=Automator-8D08D8C847A0FA6
03:13:00.856 PM | Link^SessionId=a1ab4c5d-16ff-4b7c-b2c1-23db05bb8dbf;TimeStamp=03:13:00.856;AutxId=Automator-8D08D8C847A0FA6
03:13:00.856 PM | Automation: btnMPlus_Click - ExecutionLink From: Calc.btnMPlus.Click To: Calc.lblDisplay.Properties ID: a1ab4c5d-16ff-4b7c-b2c1-23db05bb8dbf;TimeStamp=03:13:00.856;AutxId=Automator-8D08D8C847A0FA6
03:13:00.856 PM | End^SessionId=a1ab4c5d-16ff-4b7c-b2c1-23db05bb8dbf;TimeStamp=03:13:00.856;AutxId=Automator-8D08D8C847A0FA6
03:13:00.856 PM | Automation: btnMPlus_Click - Executed From: Calc.btnMPlus.Click To: Calc.lblDisplay.Properties ID: a1ab4c5d-16ff-4b7c-b2c1-23db05bb8dbf;TimeStamp=03:13:00.856;AutxId=Automator-8D08D8C847A0FA6
03:13:00.872 PM | Automation: btnMPlus_Click - ExecutionLink From: Calc.lblDisplay.Text To: Bing.lblQry.Properties ID: a1ab4c5d-16ff-4b7c-b2c1-23db05bb8dbf;TimeStamp=03:13:00.872;AutxId=Automator-8D08D8C847A0FA6
03:13:00.872 PM | Automation: btnMPlus_Click propagating From: Calc.lblDisplay.Text To Bing.lblQry.Text Value: 3.14
03:13:00.872 PM | Automation: btnMPlus_Click propagated From: Calc.lblDisplay.Text To: Bing.lblQry.Text
```

Since the automation worked the first time, Bing.lblQry appears to be the correct control to copy data into. Add additional log lines to the existing excerpt by going back to the original

RuntimeLog.txt file and bookmarking more lines.

11. Repeat the process to search and bookmark the following text:
 - a. **Control is matched**
 - b. **Control is detached**
12. Copy and paste the new bookmarked lines in the new text file.
13. Trace through and find all references to the lblQry. It is matched at 03:12:49.240 and detached at 03:12:53.273.

The user attempted to use lblQuery at 03:13:00.872. If the user had waited 30 seconds for the implicit timeout to occur, there would have been an exception and an error message at around 03:13:30. Instead, the user ended the operation before the timeout occurred — users often describe this common situation as “is just disconnected from the application” or “the solution became unresponsive.”

```

03:12:47.275 PM | MicrosoftHTMLFactory-10755539 | Control is matched. Target: mshtml.dll, ParentTarget: null
03:12:49.240 PM | | lblQry-52012354 | Control is matched. Target: [OpenSpan.Adapters.Web.HtmlInputElement], ParentTarget: [OpenSpan.Adapters.Web.HtmlInputElement], ParentTarget: [OpenSpan.Adapters.Web.HtmlInputElement]
03:12:49.240 PM | frmBing-40778967 | Control is matched. Target: [OpenSpan.Adapters.Web.HtmlFormElement], ParentTarget: HtmlDocument(id:2E8B3C, window:HWND:00D03AA), ParentTarget: null
03:12:49.240 PM | pgBing1-54245401 | Control is matched. Target: HtmlDocument(id:2E8B3C, window:HWND:00D03AA), ParentTarget: null
03:12:52.980 PM | btnMPlus_Click-39086322 | Start^SessionId=46ad0cf5-62f5-4d41-b9b9-2d099f4a2965;TimeStamp=03:12:52.980;AutxId=Automator-8D08D8C847
03:12:52.980 PM | btnMPlus_Click-39086322 | Link^SessionId=46ad0cf5-62f5-4d41-b9b9-2d099f4a2965;TimeStamp=03:12:52.980;AutxId=Automator-8D08D8C847
03:12:52.996 PM | | Automation: btnMPlus_Click - ExecutionLink From: Calc.btnMPlus.Click To: Calc.lblDisplay.Properties ID: 46ad0cf5-62f5-4d41-b9b9-2d099f4a2965
03:12:52.996 PM | btnMPlus_Click-39086322 | Link^SessionId=46ad0cf5-62f5-4d41-b9b9-2d099f4a2965;TimeStamp=03:12:52.996;AutxId=Automator-8D08D8C847
03:12:53.012 PM | btnMPlus_Click-39086322 | End^SessionId=46ad0cf5-62f5-4d41-b9b9-2d099f4a2965;TimeStamp=03:12:53.012
03:12:53.012 PM | | Automation: btnMPlus_Click - Executed From: Calc.btnMPlus.Click To: Calc.lblDisplay.Properties ID: 46ad0cf5-62f5-4d41-b9b9-2d099f4a2965
03:12:53.012 PM | | Automation: btnMPlus_Click - ExecutionLink From: Calc.lblDisplay.Properties To: Bing.lblQry.Properties ID: 46ad0cf5-62f5-4d41-b9b9-2d099f4a2965
03:12:53.027 PM | | Automation: btnMPlus_Click propagating From: Calc.lblDisplay.Text To Bing.lblQry.Text Value: 3.14
03:12:53.027 PM | | Automation: btnMPlus_Click propagated From: Calc.lblDisplay.Text To Bing.lblQry.Text
03:12:53.027 PM | | Link^SessionId=46ad0cf5-62f5-4d41-b9b9-2d099f4a2965;TimeStamp=03:12:53.027;AutxId=Automator-8D08D8C847
03:12:53.027 PM | btnMPlus_Click-39086322 | Automation: btnMPlus_Click - ExecutionLink From: Bing.lblQry.Properties To: Bing.btnGo.PerformClick() ID: 46ad0cf5-62f5-4d41-b9b9-2d099f4a2965
03:12:53.074 PM | | Automation: btnMPlus_Click - Executed From: Bing.lblQry.Properties To: Bing.btnGo.PerformClick() ID: 46ad0cf5-62f5-4d41-b9b9-2d099f4a2965
03:12:53.074 PM | | Automation: btnMPlus_Click - Executed From: Calc.lblDisplay.Properties To: Bing.lblQry.Properties ID: 46ad0cf5-62f5-4d41-b9b9-2d099f4a2965
03:12:53.273 PM | | lblQry-52012354 | ControlBase.Detach(destroyControl:True) - Control is detached
03:12:53.274 PM | btnGo-13957688 | ControlBase.Detach(destroyControl:True) - Control is detached
03:12:53.275 PM | frmBing-40778967 | ControlBase.Detach(destroyControl:True) - Control is detached
03:12:53.275 PM | pgBing1-54245401 | ControlBase.Detach(destroyControl:True) - Control is detached
03:13:00.856 PM | btnMPlus_Click-39086322 | Start^SessionId=alab4c5d-16ff-4b7c-b2c1-23db05bb8dbf;TimeStamp=03:13:00.856;AutxId=Automator-8D08D8C847
03:13:00.856 PM | btnMPlus_Click-39086322 | Link^SessionId=alab4c5d-16ff-4b7c-b2c1-23db05bb8dbf;TimeStamp=03:13:00.856;AutxId=Automator-8D08D8C847
03:13:00.856 PM | | Automation: btnMPlus_Click - ExecutionLink From: Calc.btnMPlus.Click To: Calc.lblDisplay.Properties ID: alab4c5d-16ff-4b7c-b2c1-23db05bb8dbf
03:13:00.856 PM | btnMPlus_Click-39086322 | Link^SessionId=alab4c5d-16ff-4b7c-b2c1-23db05bb8dbf;TimeStamp=03:13:00.856;AutxId=Automator-8D08D8C847
03:13:00.856 PM | btnMPlus_Click-39086322 | End^SessionId=alab4c5d-16ff-4b7c-b2c1-23db05bb8dbf;TimeStamp=03:13:00.856
03:13:00.856 PM | | Automation: btnMPlus_Click - Executed From: Calc.btnMPlus.Click To: Calc.lblDisplay.Properties ID: alab4c5d-16ff-4b7c-b2c1-23db05bb8dbf
03:13:00.872 PM | | Automation: btnMPlus_Click - ExecutionLink From: Calc.lblDisplay.Properties To: Bing.lblQry.Properties ID: alab4c5d-16ff-4b7c-b2c1-23db05bb8dbf
03:13:00.872 PM | | Automation: btnMPlus_Click propagating From: Calc.lblDisplay.Text To Bing.lblQry.Text Value: 3.14
03:13:00.872 PM | | Automation: btnMPlus_Click propagated From: Calc.lblDisplay.Text To Bing.lblQry.Text
03:13:08.856 PM | ActiveXFactory-16020647 | ControlBase.Detach(destroyControl:True) - Control is detached

```

How control matching works

The matching of interrogated controls occurs when users navigate applications using buttons and menus.

Normally, the process of navigating a page contains the following steps:

1. A user clicks a navigation button or menu.
2. The system detaches the old page (*Control is detached* messages appear in the log).
3. When the old page and its child controls detach, a destroyed event fires for everything that has detached.
4. The new page is identified.
5. Its child controls are identified and then matched (*Control is matched* message appears in the log). When the child controls are matched, their created events fire.
6. The page is matched and the pages created event fires.

Why the solution failed

The solution failed because the user pressed the M+ button the second time. lblQry was not matched when the automation attempted to use it.

Refer to the excerpted log. Reading up from the lblQry *Control is detached* message at 03:12:53.273, you find btnGo.PerformClick occurred in automation btnMPlus_Click at 03:12:53.074. This PerformClick navigated away from the pgBing1 and onto another page. This navigation caused the detachment of the lblQry control, and therefore, breaks the btnMPlus_Click automation.

When you troubleshoot similar issues, ask these questions:

- Why is lblQry not matched?
- What caused lblQry to detach?

Introduction to automation playback

In this lesson, you learn about the Pega Robotic Automation Playback feature and how to use it.

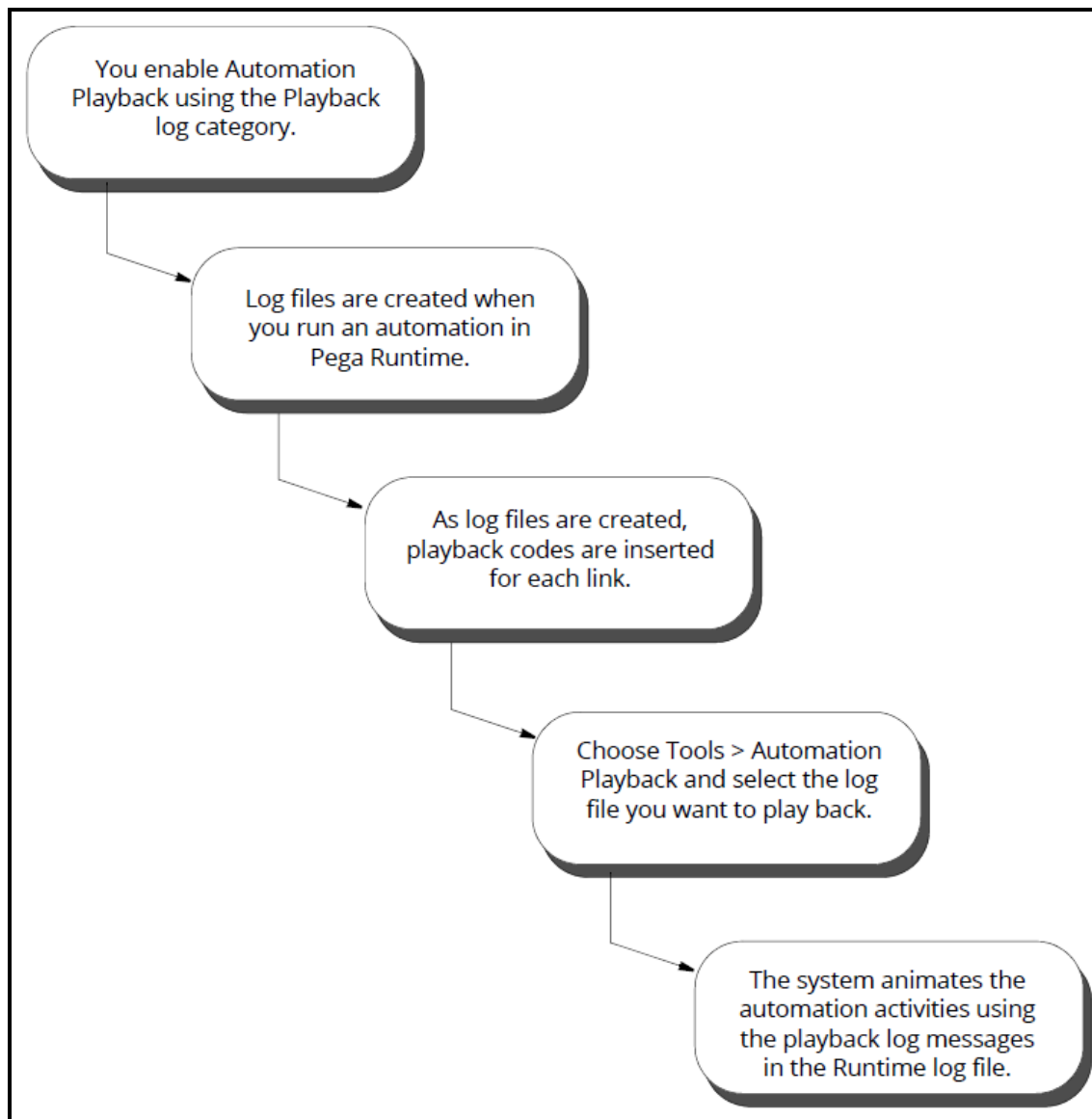
After this lesson, you should be able to:

- Define the Pega Robotic Automation Playback feature
- Explain the purpose of using the playback feature
- Describe the components of the playback window
- Perform the steps in implementing the playback feature

Pega Robotic Automation Playback

The Automation Playback feature lets you animate the automation activities in Runtime log files so you can better see what is happening as your automations execute. Automation Playback also works with files generated with the Log4Net Publisher.

The following diagram explains the process of automation playback:



You would use this feature when other debugging methodologies do not provide the information for issue resolution. Threading is a common reason for using the playback feature — it provides

information about when specific threads or contexts start and end to determine if the automation sequencing is causing an issue. You may also use this feature with other debugging methods.

Installation and configuration

The Automation Playback feature is included in Pega Robotics Studio Automation Playback and is set up during installation. You do not need to install or configure anything.

When you install Pega Robotics Studio Automation Playback, the installer makes some changes to the RuntimeConfig.xml file. If you have customized this file and prefer to add specific changes manually, you can enable the Automation Playback feature, as shown in the following image:

```
<LogCategories>
  <!-- Trace level used by all logging categories not in this list -->
  <Category name="Default" logLevel="3"/>
  ...
  <!-- Framework -->
  ...
  <Category name="Playback" logLevel="3" />
</LogCategories>
```

Set the Playback option to log level 3, as shown here:

```
<Category name="Playback" logLevel="3" />
```

This tells the Runtime system to include the information the Automation Playback feature needs to animate the log.

Playback window features

The Automation Playback window is displayed as a designer window in the Pega Robotic Automation Studio IDE. You may float, dock, or relocate the window to fit your needs.

The Playback window is separated into lower and upper sections. The lower section displays the contents of the opened log file. The upper portion provides details of each played back line from the opened log file. The following image includes the upper displayed information:

Column	Description
--------	-------------

Time Stamp	This is the time at which Automation Playback started playing back this link.
Initial Autx	This is the name of the first automation that was run.
Completed	This lets you know if the system was able to complete the processing of an automation and return. Use this status to determine which automation failed.
Run Time (ms)	This is the amount of time, in milliseconds, it took to process the link.
Links Executed	This is the total number of links that were executed.
Longest Loop	This is the number of iterations in the longest loop.
Thread Count	This is the number of threads started. Typically, this should be a low number. If the automation has spawned a high number of threads, you should check the automation and reduce the number of threads.
Autx Count	This is the number of automations used within the execution context.

These column definitions describe a single execution context.

The screenshot shows the 'Automation Playback' application window. The main table displays execution results for a single context:

Time Stamp	Initial Autx	Completed	Run Time (ms)	Links Executed	Longest Loop	Thread Count	Autx Count
02:19:32.119	Automation1	Yes	1281	9	10	3	2

To the right of the table is a 'Details' pane showing summary statistics:

- Execution Time: 1281 milliseconds
- Links Executed: 9
- Thread Count: 3
- Automation Count: 2
- Longest Loop - Iterations: 10 Automation: Autx

Below the summary is a list of 'Automations' with their respective statistics:

- Automation1**
 - Links Executed: 6
 - Thread Count: 3
 - Longest Loop - Iterations: 1
- Automation2**
 - Links Executed: 3
 - Thread Count: 1
 - Longest Loop - Iterations: 10











The bottom pane shows a detailed log of diagnostic messages. The log header includes:

- OpenSpan File Publisher diagnostic log
- Computer: WIN-ACMOKC2H2M
- File name: RuntimeLog.txt
- Created date: 4/10/2013 2:19:03 PM

The log table has columns: MsgType, Timestamp, Thread, Category, DesignComponent, Component, and Message (Plus Verbose message, if included). The messages show the progression of the application from diagnostic initialization to runtime application loading.

At the bottom of the window, the status bar indicates: Line #71 Thread ID: 47 Autx: Automation1

The Playback toolbar includes the following options to navigate through the log file automation entries.

Click	Or Press	Description
 Open Log File		Click to display the Open dialog so you can select the log file you want to run. The system defaults to the RuntimeLog.txt file.
 Close File		Click to close the log file currently loaded.
 First	Ctrl+Shift+ Up Arrow	Click to go to the first entry in the log file.
 Back	Ctrl+Up Arrow	Click to step backwards one entry.
 Next	Ctrl+Down Arrow	Click to step forward one entry.
 Last	Ctrl+Shift+ Down Arrow	Click to go to the last entry in the log file.
 Playback Selected Context Only		You can toggle this tool on or off. If it is toggled on, the Playback window only steps through the links in that Execution Context. Filtering lets you stay on the same automation path, as opposed to changing to a different path when another automation starts.
 Skip to Previous Breakpoint		Click to go back to the previous breakpoint.
 Skip to Next Breakpoint		Click to jump forward to the next breakpoint.
 Search		Search for text in the log file. Once the text is found, press Enter to go to the next match. You can press Shift+Enter to go to the previous match.

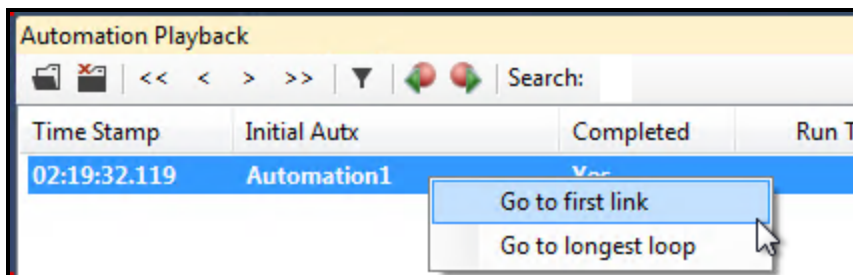
Alternative navigation options

Log files can be long and contain thousands of line entries. This depends on the configuration of the diagnostic settings found in the runtimeconfig.xml file for both Studio and the user's Runtime application.

The Playback feature provides a quick alternative navigation option that allows you to jump to specific points in the log file.

When you open a log file from the Playback window in Studio, the automation entries are displayed in the upper portion of the Playback window. You have two navigation options from which to choose:

1. Double-click an automation line in the upper portion. The playback moves to the initial link of that automation. You may step through the individual links using the Playback toolbar icons.
2. Right-click an automation line in the upper portion. A context menu displays prompting for a navigation option.



Option	Description
Go to first link	This takes you to the first link in the highlighted automation.
Go to longest loop	This takes you to the longest loop Automation Playback encountered when playing back the log file.

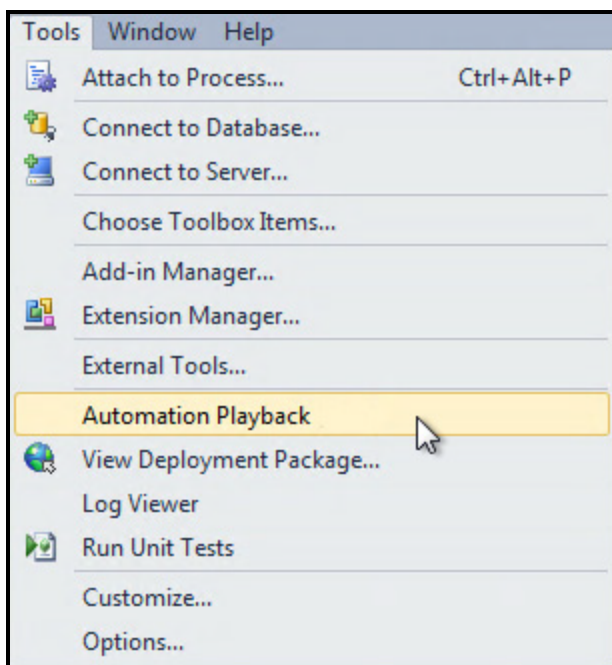
Using automation playback

After you enable the Automation Playback feature and you have created a log file by running an automation, you can play back the log file.

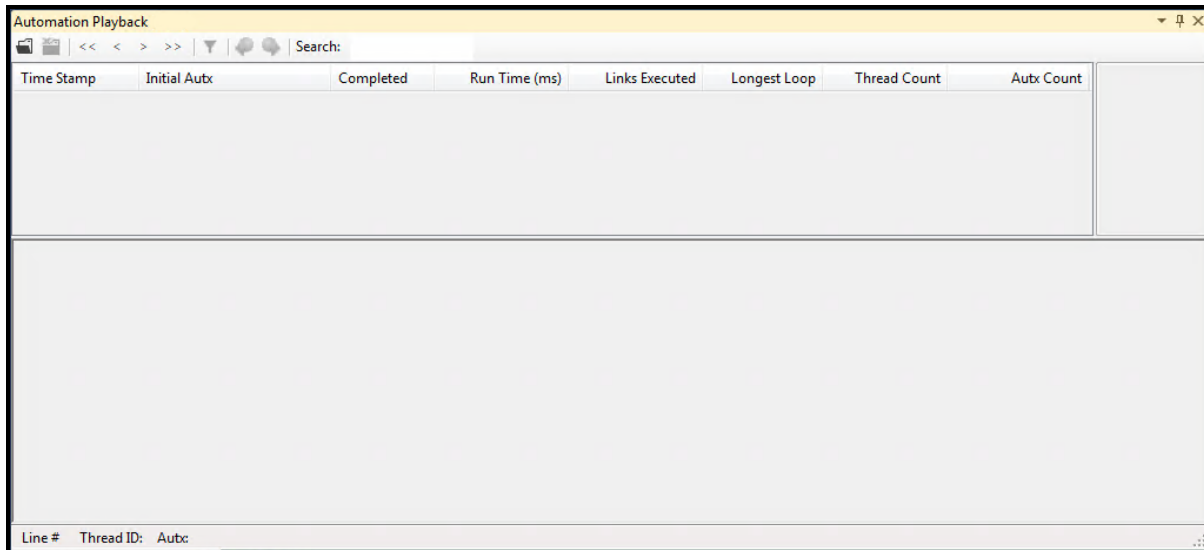
Assuming you have Pega Robotic Automation Studio started and a solution currently open in the workspace, follow these steps to play back the log file.

Note: You cannot open a playback file when you are editing an automation.

1. From the menu, choose **Tools > Automation Playback** .



The Automation Playback window is displayed.



2. On the toolbar, click the **Open Folder** icon. Open File window displays.
3. On the Open File window, navigate to the location of the log file.
4. On the Open File window, click **OK/Open**. The log file contents are displayed in the lower portion of the Playback window. All automations executed in the log file are displayed in the upper portion of the Playback window.

Log file cleaner

Often with the creation of diagnostic log files, Pega Robotic Runtime captures personal and confidential information during a solution run. Because of this, you should clean the log file of any information of this type to ensure or follow company confidentiality policies.

Pega Robotic Automation Studio provides a Log File Cleaner utility to quickly remove personal or account-sensitive data from the log files. Pega recommends that you run the Log File Cleaner utility before sending any log files to Pega Support. You may also run the log file prior to your own debugging and issue resolution if company or department requirements state to do so.

You can use the Log File Cleaner utility with the diagnostic log files generated from Studio, Runtime, Supertrace, and dump files. The utility examines each line in the log file and either masks or removes the data elements you specify, such as Social Security numbers. You can select to mask or remove on the data elements from a line, or you can mask or remove the entire line.

Installation

The Log File Cleaner utility is not part of Pega Robotic Automation Studio installation. In the Related Content section, click the **Pega Robotic Log File cleaner** link to download the cleaner utility zip file. No installation is required. Simply unzip the files to a folder location of your choice and run the executable file: **OpenSpanLogFileCleaner.exe**.

Using log file cleaner

If you do not have the correct version of Java Runtime Environment installed, a prompt is displayed stating which minimum version you need to install.

If you have unzipped the utility file, follow these steps to use the Log File Cleaner utility.

1. From the unzipped folder location, double-click the **OpenSpanLog FileCleaner.exe** to start the utility. The Log File Cleaner utility window is displayed.

OpenSpan Log File Cleaner

Step 1 Select the log file to clean.

Step 2 Select what to clean on each line of the file.

☒ Data elements only
Mask or remove only data elements within a line

☐ Entire line
Mask or remove entire line containing a specified data element

Step 3 Select the cleaning action and data element types.

Mask Data Elements

☒ Credit card numbers with

☒ US taxpayer ID numbers with

☐ Use RegEx to define what to mask

Step 4 Optional actions

☒ Mask HTML source code

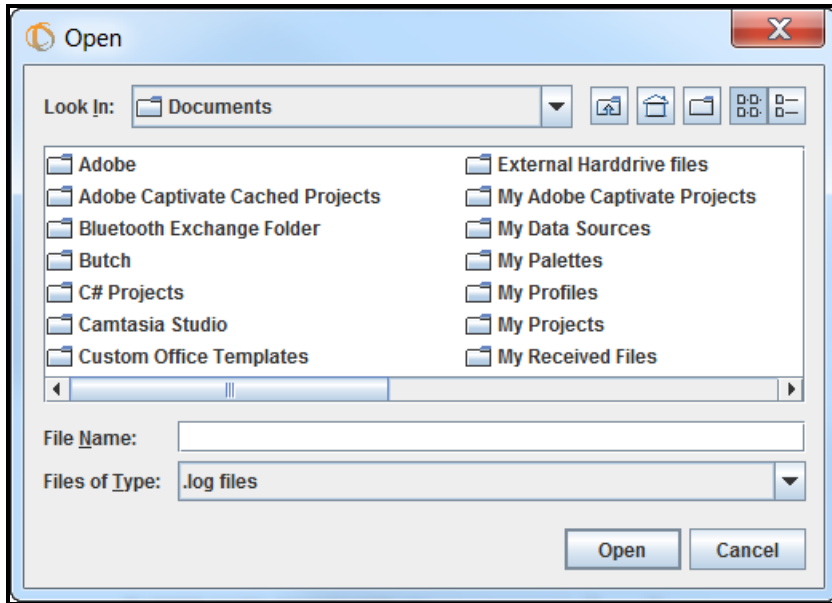
☐ Delete the original log file after cleaning

Step 5 Select the folder location and file name for the cleaned log file.

Clean Log File Close

Not ready. Select a log file to clean. v1.0

2. On the Log File Cleaner window, click the **File** icon. An Open window is displayed.



3. On the Open window, navigate to the log file location and click **Open**.
4. On the Log File cleaner window, determine how to clean the file by data elements or entire line and click the appropriate check box.
 - If you select to mask data elements, you must specify what to replace them with by making an entry in the associated text box. Leaving the text box blank removes the data element.
 - For credit card numbers, the Log File Cleaner detects all numeric strings which have lengths from 11 to 16 digits. The string can include any number of dashes. If it finds this pattern, it removes or masks these numbers with the replacement characters you specified.
 - For US taxpayers IDs (SSNs, EINs, ITINs), the Log File Cleaner looks for nine digits strings in the following formats: XXX-XX-XXXX, XXXXX-XXXX, XXXXXXXXXX, XXX-XXXXXX, or XX-XXXXXXX.
 - If removing the entire line, you do not set the mask or remove data elements option.
5. Select either **Mask HTML source code** and **Delete the original log file after cleaning**.
 - If you select **Mask HTML source code**, enter replacement text in the associated text field.

6. On the Log File Cleaner window, click the **File** icon to determine the file location and file name of the new file, if necessary.
7. Click **Save** to save the new file location and file name. The default file location is the directory of the original log file.
8. On the window, click **Clean Log File**. A display message appears in the status bar of the Log File Cleaner window.
9. On the Log File Cleaner window, click **Close**. The window closes.

Some masking example include:

- Entering -- **SSN** -- as the replacement characters for Social Security numbers, -- **SSN** -- replaces a Social Security number (for example, 123-45-6789)
- Leaving the text box blank to replace the Social Security number with a blank space

COURSE SUMMARY

This lesson group includes the following lessons:

- Lesson name
- Lesson name

Course Summary

Diagnostic logging conclusion

Now that you have completed this course, you should be able to:

- List the diagnostic log files created
- Describe the basic diagnostic log file structure and content
- Differentiate the various common log file entries for issue resolution
- Dissect a diagnostic run-time log file to determine a resolution
- Define the Pega Robotic Automation Playback feature
- Complete the steps on the playback feature to determine a resolution
- Define the log file cleaner utility
- Explain the steps using the log file cleaner utility