# Power Apps Testing – Troubleshooting and Monitoring Test Engine

## Overview

**Microsoft Power Apps** provide all users with the ability to rapidly develop business custom applications. Power Apps democratizes the development experience, enabling the creation of apps without writing code. When code is needed, both makers and professional developers are empowered with low code (**Power Fx**) and pro code options. The platform is completely extensible, allowing interaction with cross cloud and on-premise data. **Microsoft Power Apps** also provides custom connectors and integrations to connect to virtual any data source.

In this series, we are focusing on how organizations can incorporate **Power Apps** into their suite of business-critical applications. All organizations require some level of testing, from unit to stress, all applications needed by the enterprise must be resilient. In this series, we will attempt to lay out the necessary tools and design to empower organizations to implement a testing strategy for **Power Apps**.

For **Model Driven Applications**, I highly encourage you to check out [my other series on EasyRepro and Test Automation.](https://community.dynamics.com/365/b/crminthefield/posts/test-automation-and-easyrepro-01---overview-and-getting-started)

This specific section will discuss how to troubleshoot and extend the **PowerApps Test Engine**. We will look at common errors encountered when running the test engine and how to diagnose. We will explore how to extend instrumentation which can help further troubleshooting.

Finally, we will look to extend Power Apps Test Engine to work with **Microsoft Edge** browser and send telemetry to **Azure Application Insights**.

This section covers professional development topics and will require a basic understanding of the C# language. I’ll attempt to make this topic as approachable as possible but want to set the expectation of the skillset involved.

## Troubleshooting Basics

Its recommend, when troubleshooting, to try to reproduce the issue. Once the issue is reliably reproducible, troubleshooting can be as easy as walking through the steps, identifying a failure and correcting.

Start with reviewing the logs from **Power Apps Test Engine** and the test runner if running in an automated fashion. For test runner details, refer to the infrastructure documentation. Most likely, the issue will come from the steps needed to build and execute **Power Apps Test Engine**.

Once **Power Apps Test Engine** has been built and is executing, look to the information provided in the logs. **Power Apps Test Engine** has [a mechanism built in, covered in Section 03, allowing for different log levels to be output.](https://github.com/aliyoussefi/TestingPowerPlatform/blob/main/docs/Blogs/03%20Power%20Apps%20Testing%20-%20Extending%20Power%20Apps%20Test%20Engine.md#configuring-debug-properties) Review the current configuration and if needed, increase the level of logging down to Verbose.

Let’s look at some common errors and how to diagnose.

### Fixing Name isn't valid. ‘Control’ isn't recognized errors

This one is straightforward, the **Power Fx Engine** cannot find the control. W**hile the error signifies that the control can’t be found, this doesn’t necessarily mean it’s not there.** To begin, look for any changes to the control, ensuring that it is in fact on the app and named the same as the test. [If the organization is using source control and extracting](https://learn.microsoft.com/en-us/power-platform/developer/cli/reference/canvas#pac-canvas-unpack) ***the msapp package*** content each time an app is published this will save a significant amount of time. If the organization is not, it’s suggested to begin doing so. **A video detailing the steps can be found here.**

**A picture containing text, outdoor, screenshot

Description automatically generated**

In the above image, we see that ‘Button1’ isn’t recognized. In this example, simply correcting the control reference will fix the test. That said, let’s look deeper in the stack trace to understand what’s happening during the execution.

Near the end of the stack at the bottom of the image is this line:

**at Microsoft.PowerApps.TestEngine.PowerFx.PowerFxEngine.Execute(String testSteps) in C:\Users\username\source\repos\PowerApps-TestEngine\src\Microsoft.PowerApps.TestEngine\PowerFx\PowerFxEngine.cs:line 124**

[Referring to this Section 03 image showing the execution of the test, we can see the exact line that is called out above.](https://github.com/aliyoussefi/TestingPowerPlatform/blob/main/docs/Blogs/03%20Power%20Apps%20Testing%20-%20Extending%20Power%20Apps%20Test%20Engine.md#defining-the-execute-function)

**Text

Description automatically generated**

### NOTE: System.ArgumentException: locale error during automation

If you are running **Power Apps Test Engine** on a test runner, such as **GitHub workflows**, you may encounter something like this:

Text

Description automatically generated

**This originates from the same error as we see locally**, *Button1 not recognized*. However, in this error, the message is not as clear. Looking back into our troubleshooting steps, reproducing this locally gave insight into the true cause of the error. This allowed for a quick fix to the test case avoiding any source code modification.

### Fixing Executable Doesn’t Exist errors

Typically, this means the test is trying to run with a browser driver that doesn’t exist where playwright expects it. [A good question that highlights the potential issue can be found on the Playwright GitHub Issues page.](%5bQuestion%5d%20Chromium%20distribution%20'msedge-canary'%20is%20not%20found%20·%20Issue%20#15859 · microsoft/playwright (github.com))

If running **Power Apps Test Engine** locally, try running the **playwright.ps1** file again. This will remove the drivers used by **Playwright**, located in the *AppData* folder, and reinstall.

If running on a test runner, confirm that the browser driver exists. [Here is a reference for the Ubuntu Microsoft hosted agent included software.](https://github.com/actions/runner-images/blob/main/images/linux/Ubuntu2204-Readme.md)

The supported out of the box **Power Apps Test Engine** browsers include **Chrome**, **Firefox** and **Webkit**. These require no modifications to the code and can be referred to in the **Power Fx** test safely.

To add an additional browser, such as **Microsoft Edge**, the source code will have to be modified. Here is an example of modifying the configuration prior to launching the browser that will run **Microsoft Edge**.

Graphical user interface, text

Description automatically generated

This example shows the hard coded version, ideally this is extended as a test setting. The following video discusses in detail and provides an example of how to implement for **Microsoft Edge**.

## Extending Logging Information

Logs in **Power Apps Test Engine** are stored as text files in the test output. These logs are produced when running locally or in an automated fashion. The logs will write based on the severity level configured or applied at run time. To set at run or debug time, follow [the Debugging and Testing Locally section](https://github.com/aliyoussefi/TestingPowerPlatform/blob/main/docs/Blogs/03%20Power%20Apps%20Testing%20-%20Extending%20Power%20Apps%20Test%20Engine.md#debugging-and-testing-locally) in **03. Power Apps Testing – Extending and Contributing to the PowerApps Test Engine.** For a configurable setting, refer to the *config.json* showing how to set the “*logLevel*” property.

The out of the box logger uses the *Log* function to write to the text file. **This message can be extended to include additional information such as timestamps or test runner details.** The image below shows the *Log* method that can be extended.

Text

Description automatically generated

***NOTE: Simply adding a timestamp can help understand when a command executed and how long commands are taking to execute.***

## Extending Logging into Azure

The **Power Apps Test Engine** leverages the Microsoft.Extensions.Logging.ILogger interface. The benefit of this interface is that we can easily add providers to help us collect logs from the tests and test engine. The ***PowerAppsTestEngine*** console uses the *IServiceCollection* which provides the ability to add items used within, such as reporting tools, test tools and in this case: logging tools.

Here we can extend the *ILoggerBuilder* to include **Application Insights**. The benefit here is that **Azure Application Insight**s, as part of **Azure Monitor**, will **collect telemetry and provide a source for reporting, alerting and storage.** [Detailed information on how to add Application Insights into ASP.NET Core applications can be found here.](https://learn.microsoft.com/en-us/azure/azure-monitor/app/asp-net-core?tabs=netcorenew%2Cnetcore6)

Text

Description automatically generated

Once configured, the messages delivered to **Application Insights** will include information about the test runner, the user running the test and timestamps**. As we look to scale testing globally and in parallel, having these properties automatically delivered will help determine performance or availability issues.**

## Next Steps

By now, you should have a firm understanding of test tools available for Canvas Apps. You should also be able to articulate and define test suites and cases. You should be able to show how to configure tests within a specific suite and globally across all tests.

You have now learned how to modify the source code for the Power Apps Test Engine. Additionally, you can now articulate the steps needed to contribute back to the open source project.

Continue evaluating any gaps in the **Power Apps Test Engine** for your business needs. If a more complex command is needed review other commands such as the Select, which perform an action and retrieve updates to the app.