# Power Apps Testing – Deep Dive into Power Fx Tests

## 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 build tests for **Power Apps Test Engine** and **Test Studio**. We will dive deep into the structure of the tests and key call outs for configurations. We will also investigate storing tests and delivery of tests including dynamic variables to our test engines. If you have not reviewed the Overview, I suggest you do as this is a continuation of that section.

## What is Power Fx?

[Power Fx](https://learn.microsoft.com/en-us/power-platform/power-fx/overview) is the low-code language that is used across **Microsoft Power Platform**. It's a general-purpose, strong-typed, declarative, and functional programming language. The anticipation is it will continue to grow as the preferred language of the platform, for apps, flow, bots, etc. **Power Fx** even has the ability to [transform natural language](https://learn.microsoft.com/en-us/power-apps/maker/canvas-apps/power-apps-ideas-train-examples) to low-code as well as [provide recommendations](https://learn.microsoft.com/en-us/power-apps/maker/canvas-apps/power-apps-ideas-transform?source=recommendations).

In this section, **Power Fx** is used to define what actions the tests are performing.

To learn more about **Power Fx** including a curated learning path, [start here](https://learn.microsoft.com/en-us/training/paths/use-basic-formulas-powerapps-canvas-app/?ns-enrollment-type=Collection&ns-enrollment-id=m0js310oz3r5z).

## Using YAML to define the Test Definition

**YAML** is the language used for our **Power Apps Test Suite** and **Power Fx** is the language used for the test definition.

## Configuring the Test Suite

The test suite

### Core Properties

|  |  |
| --- | --- |
| **Name** | **Description** |
| testSuiteName | Required field naming the test. Will show in the logs for each test suite and it is useful for reporting. |
| testSuiteDescription | Optional field describing the nature of the test suite. By default, this is not added to the trx file so would mainly be used by test writers. |
| persona | Required field of user running the tests. Must match a user in the environmentVariables section below. |
| appLogicalName | Required for solution aware apps. All business-critical apps should be in solutions per operational excellence architecture. You can find this in the solution in the Maker portal or through the Dataverse API. |
| appId | Required for non-solution aware apps. See above. |
| onTestCaseStart | Triggered for all test cases in a suite before the case begins executing. [Use this for common code (test data, initializing variables, etc) that needs to run before every case.](https://learn.microsoft.com/en-us/power-apps/maker/canvas-apps/working-with-test-studio#setup-your-tests) |
| onTestCaseComplete | Triggered on completion of a test case. Includes information about the test case. Schema:  {  TestPass: TestCaseResult.TestCaseName & ":" & Text(Now())  ,TestSuiteId: TestCaseResult.TestSuiteId  ,TestSuiteName: TestCaseResult.TestSuiteName  ,TestCaseId: TestCaseResult.TestCaseId  ,TestCaseName: TestCaseResult.TestCaseName  ,StartTime: TestCaseResult.StartTime  ,EndTime: TestCaseResult.EndTime  ,TestSuccess: TestCaseResult.Success  ,TestTraces: JSON(TestCaseResult.Traces)  ,TestFailureMessage: TestCaseResult.TestFailureMessage  } |
| onTestSuiteComplete | Trigged on complete of the test suite. Includes information about successful and unsuccessful tests. Schema:  {  TestSuiteId: TestSuiteResult.TestSuiteId  ,TestSuiteName: TestSuiteResult.TestSuiteName  ,StartTime: TestSuiteResult.StartTime  ,EndTime: TestSuiteResult.EndTime  ,TestPassCount: TestSuiteResult.TestsPassed  ,TestFailCount: TestSuiteResult.TestsFailed  } |
| networkRequestMocks | Provide mock requests for connectors. Highly recommended for shift left tests. Examples are [here](https://github.com/microsoft/PowerApps-TestEngine/blob/main/samples/connector/testPlan.fx.yaml) and here. |
| testCases | Includes one or more test cases. |

### Test Case Properties

|  |  |
| --- | --- |
| **Name** | **Description** |
| testCaseName | Required field naming the test case. This is used in the trx file and test suite complete information. I recommend making each of these unique and should inform what the test is trying to do. |
| testCaseDescription | Optional field but can further define the test case. |
| testSteps | Required list of Power Fx operations. [For PowerApps-TestEngine, refer to this document.](https://github.com/microsoft/PowerApps-TestEngine/tree/main/docs/PowerFX) This can be extended and is covered in an upcoming section. |

## Configuring the Test Settings

The test settings section allows testers to define which devices and browsers will be used to run the tests. Test settings also provide additional properties to define how the tests are run such as interactive mode and parallelism. Also included are default settings to record a video of the tests or how long to wait until the test should end if it doesn’t receive a response from an action.

A reference to the test settings can be [found here](https://github.com/microsoft/PowerApps-TestEngine/blob/main/docs/Yaml/testSettings.md). I’ll provide additional thoughts in the table below.

|  |  |
| --- | --- |
| **Name** | **Description** |
| filePath | The file path to a separate yaml file with all the environment variables. If provided, it will override all the users in the test plan. |
| browserConfigurations | Must identify at least one browser. I would recommend [making the properties within variables if running in an automated fashion](https://docs.github.com/en/actions/learn-github-actions/variables). |
| recordVideo | Creates a recording of the tests as a WEBM file. I would recommend setting this to true unless you have an extremely long running test. |
| headless | If running tests manually or locally I would set this to false allowing you to watch the test run. For automated tests, this can be set to true to reduce possible resource consumption. |
| enablePowerFxOverlay | Will show the PowerFx formula on the screen. This could be helpful when running manually or locally. For automated tests, determine usefulness especially when there are concerns with source control. |
| timeout | Default is 30 seconds. Depending on the type of test and the number of tests being run consider reducing or increasing. This setting is one I would recommend making a variable in automated tests. |
| workerCount | Default is 10. I haven’t tested the boundaries of this yet but would expect this to be determined by the test runner’s capabilities. If parallelism is a concern, consider [running multiple test runners in unison.](https://learn.microsoft.com/en-us/azure/container-apps/overview) |

### Browser Configurations

Browser configurations let us choose how to launch the test. [This includes the latest and most prolific browsers and devices such as Chromium, iPhone, Android, etc.](https://playwright.dev/dotnet/docs/browsers) For Chromium based browsers such as Chrome and Edge, Playwright will use the open-sourced builds. This means while the browser currently installed on your machine maybe version 100, the test will run on version 101. This does provide the advantage of understanding a potential change to the browser that could impact your users.

A reference to the browser configurations can be [found here](https://github.com/microsoft/PowerApps-TestEngine/blob/main/docs/Yaml/testSettings.md#browser-configuration). I’ll provide additional thoughts in the table below.

|  |  |
| --- | --- |
| **Name** | **Description** |
| browser | Must identify at least one browser. I would recommend [making the properties within variables if running in an automated fashion](https://docs.github.com/en/actions/learn-github-actions/variables). |
| device | Optional value. Needs to be a [device supported by Playwright](https://playwright.dev/dotnet/docs/api/class-playwright#playwright-devices). [Here is a comprehensive list.](https://github.com/microsoft/playwright/blob/main/packages/playwright-core/src/server/deviceDescriptorsSource.json) I would recommend [making the properties within variables if running in an automated fashion](https://docs.github.com/en/actions/learn-github-actions/variables). I would also recommend testing a single app across all types of devices it may be used: mobile and workstation. |
| screenHeight | Height of the browser screen. It defined will require screenWidth. In the extending Test Engine article, we can [further define browser arguments as shown here](https://github.com/microsoft/playwright/issues/4046). |
| screenWidth | Width of the browser screen. It defined will require screenHeight. In the extending Test Engine article, we can [further define browser arguments as shown here](https://github.com/microsoft/playwright/issues/4046). |

## Configuring Environment Variables

Environment variables can be applied to a single test suite or across all test suites. The reason to include across all test suites might be to have a single source of personas to choose from, like variable groups within Azure DevOps. This allows us to centralize and reduce the sprawl of user personas. This also helps as we test across environments during application deployments.

A reference to the environment variables can be [found here](https://github.com/microsoft/PowerApps-TestEngine/blob/main/docs/Yaml/environmentVariables.md).

|  |  |
| --- | --- |
| **Name** | **Description** |
| filePath | The file path to a separate yaml file with all the environment variables. If provided, it will override all the users in the test plan. Consider making this a [GitHub action environment variable](https://docs.github.com/en/actions/learn-github-actions/variables#defining-environment-variables-for-a-single-workflow). |

## Configuring Users

Microsoft Power Apps users come to apps with different roles and different business requirements. As we define our tests we must consider how these users will use the app and what their experience using the app will be. Its recommended to execute tests as the primary persona who will interact with the app, which can vary depending on the app’s purpose. Luckily, the environment variables all of us to define multiple personas and programmatically define which user to run our test suite as.

|  |  |
| --- | --- |
| **Name** | **Description** |
| personaName | The name of the type of user (e.g., User1, ContosoUser1, HRManager, etc) |
| emailKey | Username of the user (xxx@xxx.onmicrosoft.com) |
| passwordKey | Password of the user |

## Test Best Practices

The documentation for Power Apps Test Studio does a really good job of laying out fundamental best practices including:

* Keeping test cases small.
* Keeping expressions to a single action in a test.
* Building deterministic tests.
* Managing multiple tests with test suites.

A few points I want to call out about tests come from the Test Desiderata. Building upon the deterministic attribute above, we must consider the fact we are running tests against an ever-changing variable, the browser. Fundamentally our tests should always provide the same result and I expect they should.

However, it is key that we are testing against the build version (vNext) understanding this is not what the current user base may see. I suggest, running the same tests across not only the vNext but the vCurrent of the browser. This also applies to devices. Consider the landscape of devices available that users interact with. Tablets, phones, watches, of all different builds and versions. Attempting to test across this can provide cumbersome. It’s important that as we define our test strategy, we make every good faith attempt to provide coverage but clearly set expectations to the user base and decision makers. If you are interested, I highly recommend finding a used copy of [Dino Esposito’s “Architecting Mobile Solutions for the Enterprise”.](https://www.microsoftpressstore.com/store/architecting-mobile-solutions-for-the-enterprise-9780735663022) The section covering WURFL is especially appealing.