

Automated Testing of Blaise 5 Survey Instruments using Playwright

Patrick Wenzel, Svetlana Rabinovich, Todd Flannery, Westat, Rockville, Maryland, USA

1. Abstract

Automated Testing of large survey instruments remains a challenge. With the introduction of the web survey mode in Blaise 5, modern end-to-end automated web application testing tools can be used. This paper suggests an approach using Excel files to drive automated tests for web survey instruments using the Playwright Testing framework¹.

2. Introduction

The authors are applying the concept of shift-left² testing to Blaise survey instrument development. Shift-left testing is a software development approach that emphasizes moving testing activities earlier in the software development process to improve software quality, reduce costs, and accelerate time to production by catching defects earlier.

Shift-left testing relies on automated testing so tests can be run quickly with each instrument build. The authors have developed an automated testing approach that uses the Playwright testing framework to move testing activities earlier in the instrument development process.

These automated tests augment manual testing activities which occur later in the survey instrument development process, with the goals of 1) reducing the number of test cases that must be manually tested, and 2) finding and fixing more defects before manual testing begins 3) ensuring that new test releases continue to follow known routes through an instrument.

3. The Playwright Testing Framework

Playwright, like other automated testing frameworks, has the ability to locate controls, labels, and other HTML elements on a web page. The scripts can make assertions and then test those assertions in a series of controlled and repeatable tests. These tests are then repeated upon each section release to assure that routes are followed as per specification.

4. Challenges

There were several challenges that we encountered in getting Playwright to work against Blaise 5 surveys. The initial approach was to start with simple Blaise 5 surveys and then move to more and more complex surveys, adjusting our approach as new issues became evident.

4.1 Getting Playwright to Work with Blaise 5 Surveys

Blaise 5 surveys are rendered as a single-page application (SPA) using the Angular JavaScript framework, so our first challenge was in determining when the page has loaded and when the page has changed. We tried several approaches, such as monitoring network transfers and the state of HTML document, but we found the most reliable method was simply checking for changes to the data field controls on the page.

One of the advantages to using a testing framework like Playwright is that testers do not need to know about the internals of the Blaise surveys, as they just use the rendered HTML page as a source

for the tests.

Here is a snippet of one of our first tests against a simple Blaise 5 survey question:

```
//  
// First Question...  
//  
console.log('On Question 1');  
  
// Check if the question text is present  
const textLocator_q1 = page.locator('text=Which of the following kinds of beers, ales or malt liquors have you ever consumed?');  
const isVisible_q1 = await textLocator_q1.isVisible();  
  
// Assert the text is present  
await expect(textLocator_q1).toBeVisible();  
  
// Log the result  
console.log('Question Text is ${isVisible_q1 ? 'visible' : 'not visible'}');  
  
// try to select the option checkbox that has a label of "Indian Pale Ale (IPA)"  
page.locator('label:has-text("Indian Pale Ale (IPA)")');  
await selectOptionByLabelText(page, "Indian Pale Ale (IPA)");  
  
// call function to click the 'Next' button  
await clickNextButton(page);
```

Playwright, like other testing frameworks provides tools for locating elements on an HTML page, asserting conditions and testing those assertions. And since the Playwright tests scripts are written in JavaScript, programmers can write functions to do common methods like selecting an option by its label or clicking the next button. We found that it was very easy to program simple tests in Playwright, as the example above illustrates.

4.2 Using a Data-Driven Approach

Once we knew that we could test a small Blaise 5 survey using a Playwright test script, the next challenge was that we understood that writing Playwright test scripts for the hundreds of tests cases would be an overwhelming task. And we wanted to make the test cases easier for everyone involved in survey testing to understand. Our approach was to create an Excel file with the Blaise field names, field types, question IDs, and possible responses, and columns for the tests cases that we wanted to test. The approach allows the process to be mapped to our specification documentation, and it allows tests to be run against patterns in the data without requiring direct access to the active Blaise record via the API.

Blaise Field Name	Blaise Field Type	Question ID	DBTC001	DBTC002	DBTC003	Question Text	Question Response Codes and Label
SelectTheID	STRING	Preload Scenario Code	DBPreloads001	DBPreloads002	DBPreloads003	n.a.	n.a.
BEER.DB1001	ENUMERATION	DB1001	No	No	No	You were just asked questions about imported beers. Besides those products, have you ever consumed any domestic beers?	1) Yes (Specify) 2) No
BEER.DOMESTIC.DB1002	ENUMERATION	DB1002	<skip>	<skip>	<skip>	Have you ever consumed a domestic beer, even one or two times? (Domestic beer includes any beer brewed in the United Status.)	1) Yes 2) No [GO TO BOX DBV02]
BEER.DOMESTIC.DB1004	ENUMERATION	DB1004	Yes	No	Yes	In the past 30 days, have you drank a domestic beer, even one or two times? (Domestic beer includes any beer brewed in the United Status.)	1) Yes 2) No [GO TO BOX DBR04]
BEER.DOMESTIC.DB1002_12M	ENUMERATION	DB1002_12M	<skip>	Yes	<skip>	In the past 12 months, have you drank a domestic beer, even one or two times?	1) Yes 2) No
BEER.DOMESTIC.DB1003	ENUMERATION	DB1003	Every day	Some days	Some days	Do you now consume domestic beers...? Please do not include imported beers when answering.	1) Every day 2) Some days
BEER.DOMESTIC.DB9011	ENUMERATION	DB9011	<skip>	<skip>	<skip>	Around this time 12 months ago, were you consuming domestic beers in bottles, cans or draft?	1) Yes 2) No [GO TO BOX DBV02]
BEER.DOMESTIC.DB1009.NN	INTEGER	DB1009	<skip>	<skip>	<skip>	About how long has it been since you last took a sip from an domestic beer? (If it was earlier today, enter 1 day.)	1) ___ Days 2) ___ Months
BEER.DOMESTIC.DB1009.UN	ENUMERATION	DB1009	<skip>	<skip>	<skip>		Days Months

In the “test cases” columns, users could enter the value that they wanted for each test case. If a question is not asked for a given test case, the users are instructed to enter “<skip>” for the response. The Playwright test script will report an error if a question marked as skip is encountered. Likewise, at the end of a tests case, the Playwright script will check that every question that was assigned a response was asked. If any questions were not asked, the script returns an error for the test case.

4.3 Survey Section Testing

As we implemented our data driven approach, we realized that in our larger surveys, with more than 1,000 questions, setting up and maintaining hundreds of test cases is also an overwhelming task. This led to our solution of implementing survey section testing. The idea behind survey section testing was that typically, the Blaise programmers implement a survey *by sections*. The ability to test a survey section gives them immediate feedback on the survey section rather than waiting weeks or months until the full survey is programmed and tested. Survey section testing is part of a “shift-left” approach to systems testing that attempts to give developers immediate feedback to their work.

4.4 Preload Scenarios

The advantage to survey section testing is that the test cases file become much more manageable, but it did create one problem for us: the number of preloads for a survey section has to include the responses to any questions answered in any previous survey sections, as well as any values that are used as input into the whole survey, if those values affect the routing rules for the section being tested.

One survey section we tested had 8 preload variables, most of which were flags that could either be true or false. But that meant that for this section, there could potentially be 256 possible combinations of preload variables. By adding constraints to the script, we were able to reduce it to only 36 possible sets of preload variables. We refer to these as “preload scenarios” and they are documented in an Excel file so that anyone running the Playwright test scripts knows the values for all the preload variables. A subset of the file is shown below:

Flag	Parameter	Blaise Field Name	Blaise Field Type	Description	Question ID	Preload Scenarios
[ignore]					2024 Beer Survey Domestic Section Preloads	DBPreloads001 DBPreloads002 DBPreloads003 DBPreloads004
	1	BEER.LASTUSE	tLastTimeUse	tLastTimeUse = 1) "Earlier today" 2) "Not today but sometime in the past 7 days"	LASTUSE	
	2	PL.in_WNR	WNR	WNR = 1) NonResponder 2) Responder	WNR	1 1 1
	3	PL.P_EVER_NEVER_ALCOHOL	EVERNEVER	EVERNEVER = 1) EVER 2) NEVER	EVER_NEVER_ALCOHOL	1 1 1
	4	GLOBAL.GBL_DRINKING_AGE	DRINKAGE	DRINKAGE = 1) 18 Years of Age 2) 19 Years of Age	DRINKINAG_AGE	4 4 4
	5	PL.inP_REGUSE_BEER	YESNO	tYesNo = 1) Yes 2) No	RO8_REGUSE_BEER	1 1 2
	6	PL.inP_NUMLIFE_BEER	NUMLIFE	NUMLIFE = 1) HUNDRED_OR_MORE 2) LESS_THAN_HUNDRED	RO8_NUMLIFE_BEER	2 1 2
	7	GLOBAL.GBL_PAST30DAY_BEER	YESNO	tYesNo = 1) Yes 2) No	PAST30DAY_BEER	

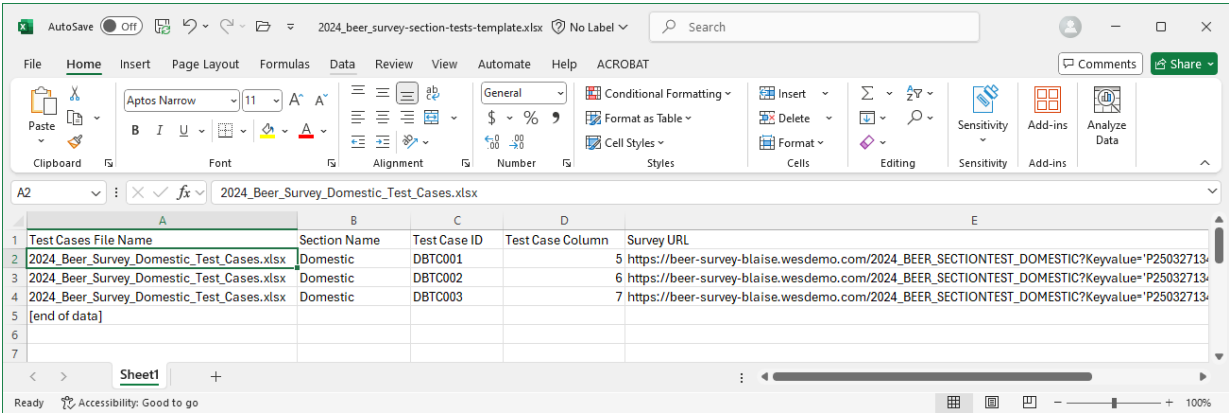
We know that some sections of our surveys have dozens of preload variables; the worst case that we have come across had 6,006 potential preload scenarios. Adding more preloads or dependencies can

easily allow thousands of scenarios to grow to millions. Since that is once again overwhelming, we have analyzed prior waves of the survey and identified the 50 most common combinations of preload variables that accounted for over 90% of the survey respondents. Although the process of defining which preload values are possible given constraints and then determining the common routes can be burdensome, we felt that the potential timesaving and efficiency of the automated scripts would provide enough value to justify the effort.

4.5 Running Multiple Tests

The final challenge that we encountered was running multiple test cases using an Excel file with the test run parameters. Since we were already using Excel to load the responses for the test cases, this seemed like the best approach. But when we implemented this in the Playwright test script, we ran into a problem: Playwright could not find any tests to run. Upon doing some research into the issue, we found out that test run parameters must be static, i.e. they must be hard coded in the Playwright test script.

Our workaround was to write a simple Node.js script³, which would open an Excel file with the test run parameters, build a JavaScript array from the Excel file, and then insert the parameters array into our survey section test script. Here is an example of the test run parameters in the Excel file:



The “insert_test_data.js” script processes this Excel file and inserted an array into the beginning of the Playwright test script:

```
const testRunData = [
  {
    "test_case_file": "2024_Beer_Survey_Domestic_Test_Cases.xlsx",
    "survey_section": "Domestic",
    "name": "DBTC001",
    "column": 5,
    "url": "https://beer-survey-blaise.wesdemo.com/2024_BEER_SECTIONTEST_DOMESTIC?KeyValue='P2503271341376610329',91&Fields=Token:'b6fde6d6-417d-46a7-8335-e43fa09a05b6',PL_MODE:"
  },
  {
    "test_case_file": "2024_Beer_Survey_Domestic_Test_Cases.xlsx",
    "survey_section": "Domestic",
    "name": "DBTC002",
    "column": 6,
    "url": "https://beer-survey-blaise.wesdemo.com/2024_BEER_SECTIONTEST_DOMESTIC?KeyValue='P2503271341376641940',91&Fields=Token:'b6fde6d6-417d-46a7-8335-e43fa09a05b6',PL_MODE:"
  },
  {
    "test_case_file": "2024_Beer_Survey_Domestic_Test_Cases.xlsx",
    "survey_section": "Domestic",
    "name": "DBTC003",
    "column": 7,
    "url": "https://beer-survey-blaise.wesdemo.com/2024_BEER_SECTIONTEST_DOMESTIC?KeyValue='P2503271341376653359',91&Fields=Token:'b6fde6d6-417d-46a7-8335-e43fa09a05b6',PL_MODE:"
  }
];
```

The Playwright script is set up to loop over the “testRunData” array to get the parameters for each test run.

An additional consideration regarding test scripts is that they must be maintained continually to reflect changes in specification. Although mature code often is maintained rather easily, starting with a new instrument that is subject to revision can increase the burden associated with test case maintenance.

5. Summary

Currently our Playwright test script is “garden path” testing, by which we mean that the responses entered in the test cases are all legitimate values. As such, we are testing the routing rules for the survey, which we believe is the source of most Blaise survey programming errors. We are testing that questions which should be skipped are in fact skipped, and that every question that was supposed to be asked for a given test case was in fact asked. We believe that our approach to data-driven testing gives us more confidence in our surveys and reduces the need for manual testing.

6. Future Directions

We have several ideas for future testing of Blaise surveys using Playwright, these include validation testing and visual testing, described below.

6.1 Validation Testing

Validation testing is a method that deliberately inputs invalid data into fields to verify that the survey responds appropriately. Unlike the “garden path” testing in our current test scripts, this approach pushes system limits to ensure that invalid data is handled correctly. This is especially crucial for numeric and date fields, as it tests boundary conditions for each input.

We have already developed small test scripts for selected fields, but we aim to expand our data-driven approach by integrating these validation tests into the “Test Case” Excel files. This will allow them to be automatically executed after any survey programming changes. We believe these validation tests will be particularly beneficial for Blaise programmers, enabling them to confirm that their validation logic functions correctly.

6.2 Visual Testing

One Playwright feature we’re aware of but haven’t yet utilized is the ability to take screenshots at any point in a test script and automatically detect differences between runs. This capability could be particularly useful for identifying unintended UI changes, ensuring visual consistency across updates, assuring that fill text is applied correctly and layouts meet requirements, and catching regressions early in the development cycle.

7. Conclusions

An automated approach like Playwright that integrates into complex Blaise instruments can prove effective in reducing burden on manual testers if those tests are efficiently integrated into a test plan in the earlier stages of the development cycle. By “shifting left” in constructing and running these sorts of tests on sections with well-defined dependencies we can prevent costlier errors further in the testing cycle and reduce the likelihood of missing regressions prior to our data collections.

8. References

¹Microsoft. "Playwright." GitHub. Accessed March 27, 2025. <https://github.com/microsoft/playwright>.

²Smith, Larry. 2001. "Shift-Left Testing." *Dr. Dobb's Journal*. September.

³Node.js Foundation. "Node.js." Accessed March 27, 2025. <https://nodejs.org/en>.