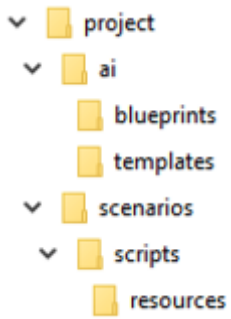
AIQ Best Practices for Stable Test Execution

Project Best Practices

Create a collaborative documentation space

1. Document, create and share the recommended settings, naming conventions, approaches, Accessor Preferences, SmartTags, etc. with your team in a common collaborative space.
2. Take these best practices, add to them, and make them your own!

Repository Organization



- **project** is the root folder (often renamed to the given project or application name)
- **ai** folder houses the ai **blueprints** and **templates**
- **scenarios** folder contains scenarios and related content within sub folders
  - **scripts** folder contains test designer scripts
    - Note: The scripts folder can contain subfolders that groups the test scripts into specific categories
- **resources** folder contains resources referenced by scripts, such as test data files

Read more about Repository Organization and Branching Best Practices here

Why do we need naming conventions?

A good naming convention is the one that is short, descriptive and that provides context to the users. Naming conventions play an important role in standardization, helping streamline development and testing processes. Consider an example where a test is broken, someone who is looking at a failed test should be able to figure out the functionalities that are broken without going through the implementation. This saves lots of time and brings effectiveness.

Different case styles

Camel Case

- It is the practice of writing phrases without spaces or punctuation, indicating the separation of words with a single capitalized letter, and the first word starting with lowercase. Example: pageCount

Pascal Case

- It is very similar to the camel case, but the first letter starts with Upper case. Example: PageCount

Snake Case

- Here we separate the words using underscore (_) and the words can be either in lower or upper case. Example: page_count or PAGE_COUNT

How should I select a case style for naming?

Folder/Package: Camel Case

- Example, ai, scenarios, utilityFunctions

Test Scripts: Custom

- It’s very important to name test scripts appropriately such that the name provides clear context. Here is the recommended naming syntax:

  [name of functionality under test]_[test input]_[expected result]

  Example: Checkout_NewUser_Pass.ds

  Checkout_NewUser_Fail.ds

Helper/Utility Scripts: Camel Case

Examples:

- generateIncrementedNumber.ds
- navigateToHomePage.ds

Test Data Files: Camel Case

Examples

- emailGenerator.csv
- filterSelector.csv

Javascript in AIQ

If you write custom Javascript code in AIQ, below are the recommended naming conventions:

Functions: Camel Case

Examples

- getUserDetails()

Variables: Camel Case

Examples:

- userName
- password
- shippingAddress

Constants:Upper Snake Case

Examples

- TOTAL_COUNT
- PRIMARY_COLOR

Test Designer Best Practices

Script Documentation

1. Include a description for EVERY script, indicating what it is and reference the source document identifier

Accessor Preferences

Before creating any planned scripts, exercise the application by recording with Test Designer, and determine the answers to these questions:

1. What are the recommended Accessor Preferences settings?
2. Which objects require manual Accessor edits when recording?
    1. Do SmartTags make sense in these cases?
    2. For example, the edits are more complex or frequent, and cannot be handled by the Accessor Preferences settings alone
3. What SmartTags are required?
4. What are common object types?
    1. Which of the common object types work with the above recommended Accessor Preferences automatically?
    2. Which objects require alternate Accessor Preferences when recording?

Simplify Navigate statements

1. When a **Navigate To** statement in a script causes or creates multiple entries or a redirect, simplify the URL to the base an end-user would actually enter
2. Remove unnecessary **Navigate To** statements
    1. At times unnecessary **Navigate To** statements may record.
    2. Remove all **Navigate To** statements which the end-user wouldn’t explicitly enter.

Wait Statements

Adjusting the automatic wait duration and using manually entered Wait Statements

It is best to never include a hard-coded duration **Wait** statement which does not also rely on other logic.

Should the application need a longer wait duration, consider these options:

1. AIQ automatically contains a ‘wait’ duration for all steps using setSearchRetries.
    1. Default setSearchRetries = 12
    2. Should the application react slowly and often need to be more patient before the next step is read, consider increasing the setSearchRetries higher than 12.
2. For a one-off longer wait duration, consider using a conditional wait by combining Wait and isVisible
    1. Wait with isVisible
    2. isVisible
3. More about Waits in AIQ: Ultimate Guide Conditional Wait

Script Modularization

Test scripts that are hundreds of lines long are difficult to maintain, can be even more difficult to debug, and can be more fragile and subject to failure when an application changes. Tips:

1. Keep your scripts compact
2. Break scripts into functional subunits and use an Include statement to combine modularized components in a main or parent script
3. As you build your library of scripts, the re-usable subunits or sub scripts will enable you to move faster when
    1. Creating new scripts
    2. Maintaining script changes
    3. Collaborating with a team

Data Driving Tests

Data-driven scripts enable varying data without maintaining multiple scripts. You may data-drive form field information, navigate statements and even accessors. Just about anything which may change over time may be data driven.

1. Ask yourself: What can and should be data-driven?
2. The most common area to data drive is the destination URL and credentials for an application
3. Form data is also another common data-driven area
4. Read more about data-driving tests and AIQ’s data production library in Operational Test Data with DPLs
5. Other common areas where data may impact test design:
    1. Test prerequisites
        1. Ask yourself: what data must exist or what state must the application reside in for the test to succeed?
    2. Any dates?
        1. Ask yourself...will this test run tomorrow, next month, next year?
6. Remember, every test script that uses DPL data must also specify the DPL data to use in a Scenario’s Resources and Test Case Scripts

Relative Paths

1. Relative paths are key to script portability, as the root of the path on your system or repository may vary from the path of other usersf
2. Please try to remember never to use absolute paths
3. Relative Paths in Test Designer
4. Test Designer script steps with Relative and Absolute path options
5. Scenario Editor Resources and Relative Paths

Naming Conventions

1. Use meaningful variable names
2. Establish and agree upon clear naming conventions of all variables, scripts and test assets

Important Scenario Design & Options

1. For Functional testing especially, ensure the following are enabled in Scenario Editor > Options > Test Designer Options:
    1. Fail step when getting failing http status codes
    2. Take Snapshot on failure
    3. Take Script Snapshot
    4. Attach Test Designer Script
    5. Attach Instant Replay
    6. Save requests and responses
2. Scenario Execution
    1. Always design your scenarios to run in parallel with other scenarios
    2. Even when just getting started, run all scenarios in parallel - do this from the beginning, not at the end of the project to test for unforseen conflicts along the way
    3. Pay attention to scenario dependencies - What must run first, next, last?
    4. Consider and document which test user ids can be used multiple times concurrently
    5. The execution duration goal should be the minimum possible wall-clock-time
3. Script Design & Parallelization
    1. When creating scripts, it's possible a process could become extremely slow, sometimes due to the nature of the application. Let Appvance know if you experience, are anticipate any potential issues before beginning to create scripts.
    2. It's important to understand if scripts can be written/executed in parallel vs sequential: There are applications where you cannot create scripts in parallel with the same account. For those, multiple accounts are required to perform parallel execution (and even recording in some cases).

Other Best Practices to Keep in Mind

1. Leadership
    1. SME (subject matter expert) - to lead, divide scripting work and review changes
    2. Communicate findings and add to best practices
2. Design for Multiple Environments
    1. DPL
        1. Coming soon...
        2. (Add design option and best practices here)
    2. CICD & Build.xml
        1. Working with multiple environments
            1. Coming soon...
        2. How can I pass scenario parameters from build.xml
            1. Coming soon...
            2. How many concurrent users?
            3. How many concurrent Browsers per Test Node?
3. Self-Healing and Fallback Accessors best practices

    1. Coming soon...
4. SmartTags
    1. To determine which SmartTags are necessary to create, see Accessor Preferences topic (above)
    2. Coming soon...

Last updated: August 14, 2023

If you have any feedback on this topic, please contact us at documentation@appvance.ai

---

Revision #8
Created 2026-07-31 15:38:38 UTC by Admin
Updated 2026-11-17 21:57:51 UTC by AI_Mananger