Manual Test Framework – Strategy

Table of Contents

[Objective(s): 2](#_Toc24629905)

[Centralized Test Assets: 2](#_Toc24629906)

[What Resources are available: 4](#_Toc24629907)

[What Configurations are supported: 4](#_Toc24629908)

[Add New/Update Test Assets for Test Cases for Existing Configurations: 4](#_Toc24629909)

[Adding a New Supported Configuration: 5](#_Toc24629910)

[Add/Update Tests that will Reside in a Repository: 6](#_Toc24629911)

[Test Plan Strategies 9.0.0.0 and Beyond: 6](#_Toc24629912)

[End to End Test Cases: 7](#_Toc24629913)

[FAQs: 8](#_Toc24629914)

# Objective(s):

Centralize Test Assets so that they can be shared across engineering, reduce the effort to configure .ktr/.kjb in order to run successfully on a user machines remove redundancy with many tests using the same test data, reduce stale/out of date test assets, enable branching for test assets to test backward compatibility release to release.

VM information can be found here: <http://iwiki.pentaho.com/display/QA/Engineering+Test+Databases#EngineeringTestDatabases-TestAssets>

Leverage the power of the Pentaho tool suite to add validation steps into the ETL, use schedulers, repositories, and scripts to execute multiple tests at once, reducing the manual effort to open each test, update configuration/paths, and execute each step manually.

# Centralized Test Assets:

All attachments for test cases will be managed in a central test asset repository. We are currently using a VM, but plan on moving to GitHub for better version control.

The current hierarchy is the following:

Branch (9.0.0.0)

Repositories

This will be where we manage Pentaho repositories (jobs and transformation stored on a repository). They can be either .xml or zip files.

These are the files created using backup/restore or import/export

Naming Convention: no specific naming convention, but it should align with the purpose of the repository when possible.

Baselines

Baseline files are any file that is used for comparison validation in your job / transformations. These files are what the expected results should be and compared to actual results.

Naming Convention: <test case id> \_baseline.<ext>

If there is more then one baseline for your test add a short description. <test case id>\_<short description>\_baseline.<ext>

Output

output files are any file(s) that are used for comparison validation in your job / transformations. These files are what the actual results are.

Naming Convention: <test case id> \_output. <ext>

If there is more than one baseline for your test add a short description. <test case id>\_<short description>\_output. <ext>

Transformations

This is where we will manage all transformation that are not within a job. This is the same structure as samples.

Naming Convention: <test case id> \_<short description>\_<additional description>

Jobs

This is where we will manage all job and transformation/jobs that belong to it. This is the same structure as samples.

Naming Convention: <test case id> \_<short description>\_<additional description>

Resources

Resources are for any files that can be shared across the team to reduce set up/configuration of environments for testing. These include JDBC drivers, share.xml (DB connections, Slave connections), named cluster xml, kettle property file with custom variables set up.

Documentation

This is where any “How To” documentation needed to set up/configuration an environment to prepare for test execution, and/or any documentation needed to execute the pseudo automated test plans.

Examples are “how to” install oracle client, “how to” execute the core regression basic config test plan, “how to” use the manual framework

Data

This is where any test data that is needed for test execution will live.

Naming Convention: no specific naming convention, but it should align with the type of test data or purpose.

# What Resources are available:

1. JDBC drivers for 9.0.0.0 supported RDBMS versions
2. kettle-basicconfig.properties or .kettle-basicconfig-Win.properties with paths for baseline, output, data as well as path to oracle client for Oracle testing for Ubuntu and Windows
3. Shared.xml with 2 slave servers, and DB connections

# What Configurations are supported:

1. Core Regression Basic Configuration
   1. Window/Ubuntu
   2. Evaluator or Archive builds
   3. Any tests that doesn’t require a configuration beyond build deployment
   4. Any test that uses a JDBC driver
   5. Any test that uses an Oracle database
   6. Kettle properties with baseline, output, data, and oracle client path

# Add New/Update Test Assets for Test Cases for Existing Configurations:

1. Download a copy of the test asset library from the VM using scp -r or tool of your choice to /home/devuser on Ubuntu or C: on Windows

scp -r [devuser@172.16.10.71:/home/devuser/9.0.0.0](mailto:devuser@172.16.10.71:/home/devuser/9.0.0.0) /home/devuser

1. Follow the configuration set up document for the types of tests you are adding/updating to test assets.
2. Write or update test case as usual
3. Include validation in the job/transformation
4. Rules for Jobs and Transformations
   1. Naming conventions for all files listed above should be followed.
   2. Jobs and all associated transformation should be copied to Jobs folder
   3. Transformations with NO associated job should be copied to the Transformations folder
   4. Ensure that all jobs and transformation paths are using ${Internal.Entry.Current.Directory} for the path. This is to ensure that the when executed the application knows where everything is in the test asset structure.
   5. If you are using a comparison for validation, baseline files should be managed in the baseline folder and the path should use the kettle.properties ${regression\_baseline\_path}
   6. If you are using a comparison for validation, output files should be managed in the output folder and the path should use the kettle.properties ${regression\_output\_path}
   7. If you are using any test data files, they should be managed in the data folder and the path should use the kettle.properties ${regression\_datafile\_path}
5. Run your job or transformation and make sure it is working properly.
6. Copy the assets (jobs, transformation, data, output, baseline files) back to the test asset repository on 172.16.10.71 (DON’T copy any other files)
7. Check the test asset folders and make sure that it updated properly
8. In the test case description add which configuration set up the test needs, and the location of the .ktr or .kjb (Transformation/Jobs folders) and put the following assets can be found at:

<http://iwiki.pentaho.com/display/QA/Engineering+Test+Databases#EngineeringTestDatabases-TestAssets>

# Adding a New Supported Configuration:

The goal with documenting configurations is to provide step by step set up of a configuration that supports a group of tests rather than document the same set up in each test case.

Create a step by step document on how to set up an environment to support a group of tests (i.e. Data Lineage, Kafka, Snowflake).

Any configuration component (resource) that can be shared, i.e. keystore files, drivers, etc. should be copied to the test asset library (if they aren’t already available) in the appropriate folder and referenced on the document.

Use kettle. Properties to manage any variables that can be shared in one place and remove any additional set up by users based on their environment. For example: if your job or transformation needs to know where the keystore file is located, add a keystore variable in kettle.properties with the correct test asset folder where it lives.

Start with the kettle.properties basic configuration file and add any variables specific to the configuration you are creating. Rename the properties file to align with the configuration. For example: kettle-snowflake.properties will be the file that should be used for snowflake tests.

If you don’t need to add any additional properties, then reference the kettle-basicconfig.properties file.

When completed copy the configuration document to the documentation folder on the test asset library, and update tests cases to use that configuration. You can know add update test case for that configuration to use the manual framework following the above steps.

# Add/Update Tests that will Reside in a Repository:

1. Download a copy of the test asset library from the VM using scp -r or tool of your choice to /home/devuser on Ubuntu or C: on Windows

scp -r [devuser@172.16.10.71:/home/devuser/9.0.0.0](mailto:devuser@172.16.10.71:/home/devuser/9.0.0.0) /home/devuser

1. Follow the configuration set up document for the types of tests you are adding/updating to the repository
2. Depending on the repository, you will either need to run the import or import-export command
3. From Spoon connect to the repository
4. For new transformation and jobs, save them to the repository
5. For updates, open the transformation or job, make your changes and save them.
6. Follow the document for updating/managing the repository that you are working on.
   1. Naming conventions for all files listed above should be followed.
   2. Jobs and Transformation location on the repository should follow any rules laid out in the repository document.
   3. If you are using a comparison for validation, baseline files should be managed in the baseline folder and the path should use the kettle.properties ${regression\_baseline\_path}
   4. If you are using a comparison for validation, output files should be managed in the output folder and the path should use the kettle.properties ${regression\_output\_path}
   5. If you are using any test data files, they should be managed in the data folder and the path should use the kettle.properties ${regression\_datafile\_path}
7. Follow any run configuration or scheduling rules for the repository
8. Run your job or transformation and make sure it is working properly.
9. When completed, backup or export the repository based on the type of repository
10. Copy the assets (repository, data, output, baseline files) back to the test asset repository on 172.16.10.71 (DON’T copy any other files)
11. Check the test asset folders and make sure that it updated properly
12. In the test case description add which configuration set up the test needs, and the repository it lives in and put the following assets can be found at:

<http://iwiki.pentaho.com/display/QA/Engineering+Test+Databases#EngineeringTestDatabases-TestAssets>

# Test Plan Strategies 9.0.0.0 and Beyond:

We want to move away from cherry picking test cases from test suites for our test plans, and focus more on creating, maintaining robust test plans that can be copied and reused for each release.

1. All the tests in the test plan need to be able to use the same environment configuration.
2. Test Plans should be organized so that it’s easy to tell what capability/feature/component is being covered. Test Plans will be used for KT to maintenance instead of walking through test suites.
3. All test assets should be managed in the manual framework
4. If the test plan has tests that require execution of jobs/transformation, leverage the capabilities of our product by using a Pentaho repository and scheduler, or create a script to run all the jobs/transformation from pan/kitchen so that you can execute multiple tests at the same time.
5. If you chose to move to a repository, create a folder in the repository that aligns with the test plan name, and remember to remove the trans/jobs from the transformations and Jobs folder on the test asset library and update the test to reflect which repository they are in, so the tests are only in one place.
6. Create integrations test to execute a set of test cases in sequence
7. Create a “How to” guide on how to set up, execute and validate the Test Plan

# End to End Test Cases:

End to End test cases are important because they are more realistic to how a user would interact with out product. End to End test cases, are really a bunch of test cases that are sequenced together to perform a complete scenario. End to End test cases will improve our productivity during regression, decrease the time it takes to find bugs and will help our maintenance team increase product coverage during their tight timelines for regression.

Identify the scenario and objective you want to test.

Identify the test cases that will be covered.

Create the test in TestRail with a test type of End to End

For each step that covers a test case, reference the test case with [test case id]

Pull the end to end test and all the referenced test cases into the same test plan



Execution of an end to end.

There are a few scenarios that may occur:

A step with a referenced test case may fail, but it doesn’t impact the objective of the scenario or block the execution of the scenario.

An example: executing a repository test scenario that validates admin can execute a job via kitchen using the command line. The job fails because it’s broken, but admin was able to execute without permission errors, and/or kitchen executed properly.

1. Fail the referenced test case, and write up a defect, reference the defect on the test case
2. Continue testing the scenario
3. Pass the end to end if all objectives are met.

A step with a referenced test case may fail, but it does impact the objective of the scenario but it doesn’t block the execution of the scenario.

1. Fail the referenced test case, and write up a defect, reference the defect on the test case
2. Continue testing the scenario to see if there are any other issues
3. Fail the scenario and reference the same defect

A step with a referenced test case may fail, but it does impact the objective of the scenario, and it does block the execution of the scenario.

1. Fail the referenced test case, and write up a defect, reference the defect on the test case
2. Block the scenario and reference the same defect

# FAQs:

1. Do I have to use the manual framework for 9.0.0.0?
   1. No, but it is encouraged to support what you can to increase execution productivity. This will be fully implemented after 9.0.0.0
2. Where do I start?
   1. Identify a set of test cases that use the same configuration in any of your regression plans. If that configuration matches the existing basic configuration, then you just need to update .ktr/.kjb and other attachments from those tests and move them to the test asset library and remove the references from the test case, and add the references to the library as stated above.
   2. If your set of tests require a different configuration, then follow adding a supported configuration, then move to the framework.
3. What steps are used for validation of my job or transformation?
   1. Many steps can be used dependent on what you want to validate. The most common are merge join (diff) – step, data validator – step, file compare – job entry, filter rows – step.
4. What if a mistake in the test asset library is made?
   1. There is a backup made of the test asset library each night.
5. For end to end tests, do I need to execute the referenced tests?
   1. No, the step in the scenario should align with the objective of the test.
   2. You will want to flag the step in the test case that is broken when you create a defect.
6. For end to end tests, what if the reference test has more than just what I want to test.
   1. If the main objective of the test meets the scenario step it’s fine.
   2. Or, you can break down the test case to be more focused.