Integration with Ranorex

What you'll learn

- Ranorex concepts and mapping to Xray
- Learn Ranorex core concepts
- Learn how creating test automation test cases looks like, at a high-level, using Ranorex Studio
- Run the tests and push the test report to Xray
- Validate that the test results are available in Jira
- Learn how to assess the impacts of related user stories in Jira, using Xray

Overview

Ranorex is a keyword-driven framework used to implement GUI test automation across a broad set of technologies, including desktop, web, and mobile. It's a codeless test automation solution. Ranorex provides comprehensive support for test automation which is supported by its core features. Data-driven testing is also supported.

Ranorex provides Ranorex Studio, the main application to implement and organize automated test scripts. Ranorex Studio has multiple components, including a recorder (Ranorex Recorder) and an object /element identifier (Ranorex Spy).

In this article we'll highlight some of the core Ranorex concepts and see how you can have visilbity of your test automation results in Jira, using Xray.

Integrating with Xray is straightforward, using JUnit XML reports that Ranorex can generate. To get the integration done, you just need to get that working.

Ranorex concepts and mapping to Xray

Ranorex Studio provides a complete GUI for implementing automated tests. Therefore, we find some concepts typical in IDEs (e.g. solution, project).

There are some specifics concepts related to test automation.

The only concept with a direct mapping to Xray will be Ranorex' Test case which will be abstracted as a Xray Test issue (unstructured/generic).

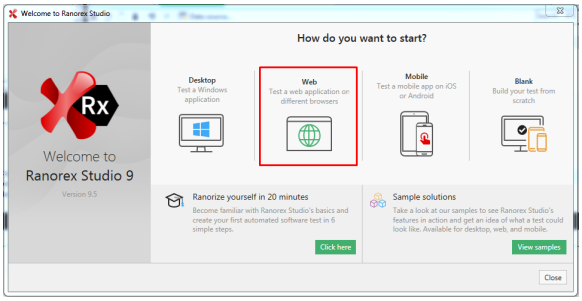
Ranorex concept	Description	Xray concept
Solution	In Ranorex Studio, a solution is the top-level container that contains all other test files. Solutions are organized into one or more projects. Whenever creating a solution, we may identify the type of application we aim to test (e.g. desktop, web, mobile). A solution always has a "test suite" project.	
Project	A tailored place to organize test files. A project can be of one of several types, including "test suite," which offers different capabilities.	
Test suite	The test suite is where you build, organize, and run your tests in Ranorex Studio. A test suite consists primarily of test cases.	Doesn't exist as an entity. Will be visible and part of the definition of each Test issue.
Test case	A test, composed of Modules, which in turn are composed of Actions.	Test issue.

Module	A modularized sequence of actions, that have a certain goal. Can be seen as a grouped sequence of steps in a test case. Modules can be reused between Test cases.	
Action	A step, inside a Module. Can be a mouse/keyboard interaction or a validation.	
Validation	An assertion, a validation.	
Repository	A Repository contains Repository Items (i.e. UI elements) organized in a tree-like structure. UI elements that contain other UI elements are represented as folders in the repository, with app folders acting as top-level elements and rooted folders as children.	
Repository item	A representation of a user interface (UI) element used in a test. Each repository item has a name and is defined by its RanoreX Path (i.e. path).	

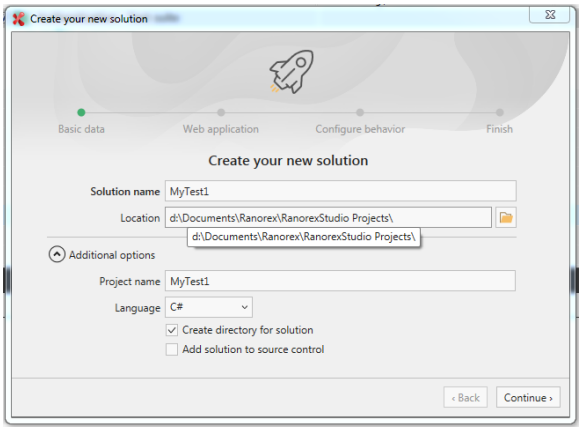
The overall result of a given test case will be available on the Test Run that will be created in Xray and associated to the corresponding Test issue.

Prerequisites

Start by creating a solution, choose web-based in order to test a web site.

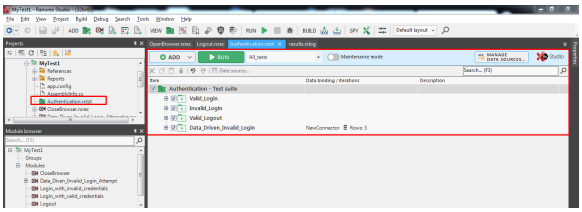


Under "Additional options" we can customize the language of the underlying code that Ranorex will use to support the test automation.



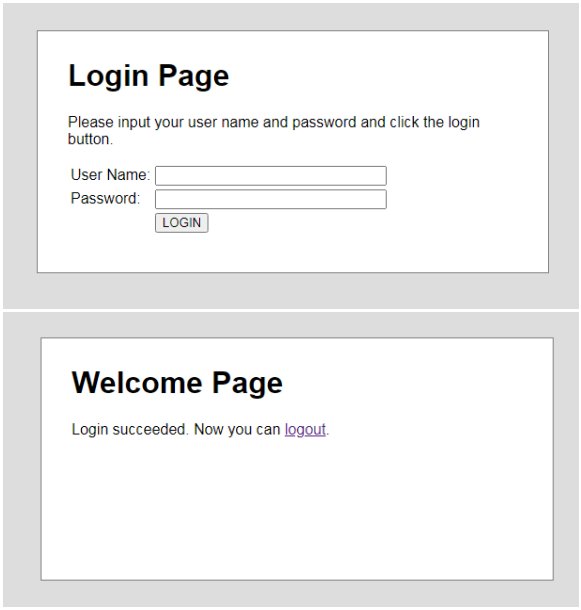
A project with the type "test suite" will be created.

It contains one Test Suite, where we can create and manage our test cases. You can rename this Test Suite to have a more meaningful name (e.g. "Authentication").



Implementing automated tests

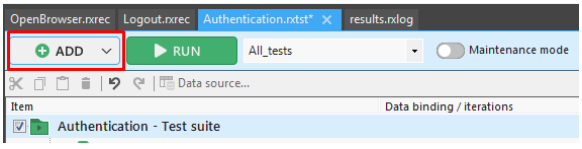
The example we will test is to implement automated tests for a dummy web site, providing an authentication mechanism that we aim to check, namely the login and logout features.



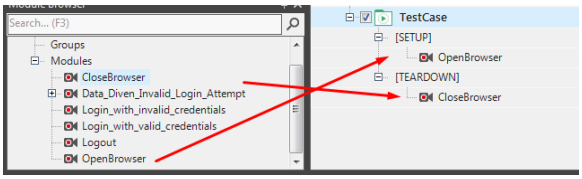
Our tests that are a part of a test suite, include these scenarios:

- valid login
- invalid login
- valid logout
- invalid login (data-driven scenario)

To add some Test cases to the Test Suite, we can use the "Add" button.

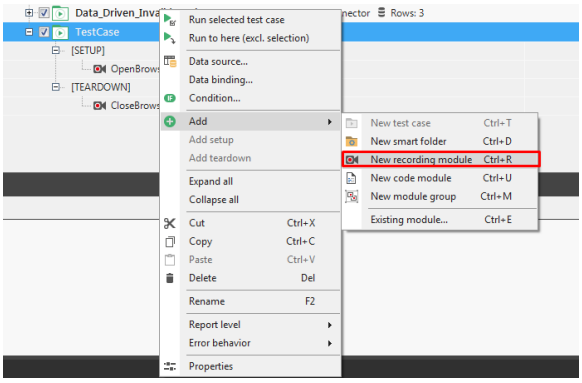


We can rename the Test case (i.e. from TestCase to whatever describes it). Then add a Setup and a Teardown section and add the OpenBrowser and CloseBrowser modules, to each section respectively. Opening the OpenBrowser module, by double-clicking on it, will allow us to set the URL to be used.

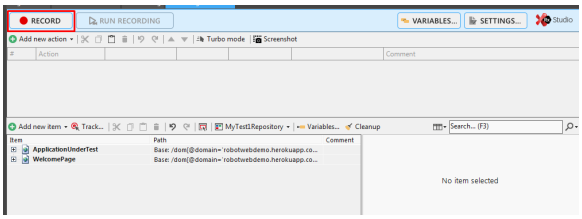


We can then use Ranorex Recorder to create a "module" (i.e. a set of sequential actions and/or validations on UI elements).

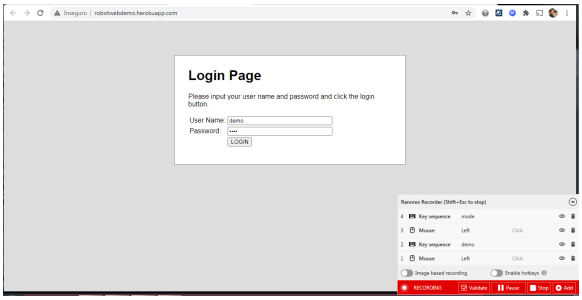
This will be used to implement interactions with our web site, without having to code.



We choose "Record" and then we're redirected to the browser, where we can perform actions which will be recorded.

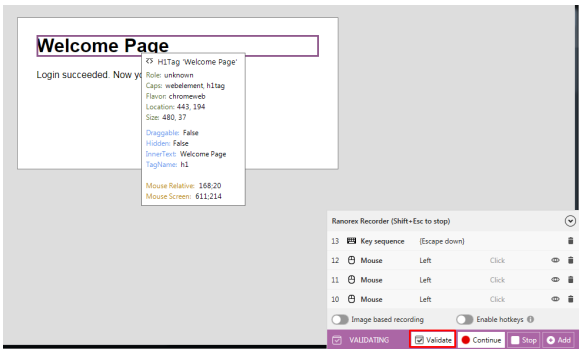


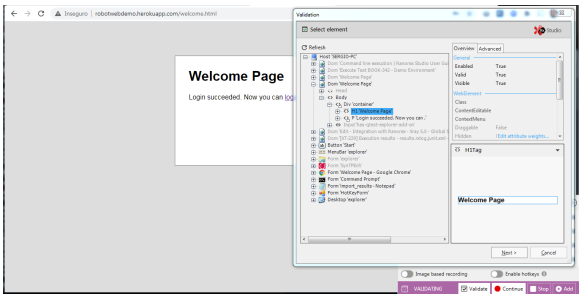
While recording the actions of our module, we can remove some that may be added by accident for example. We can also pause and stop recording.



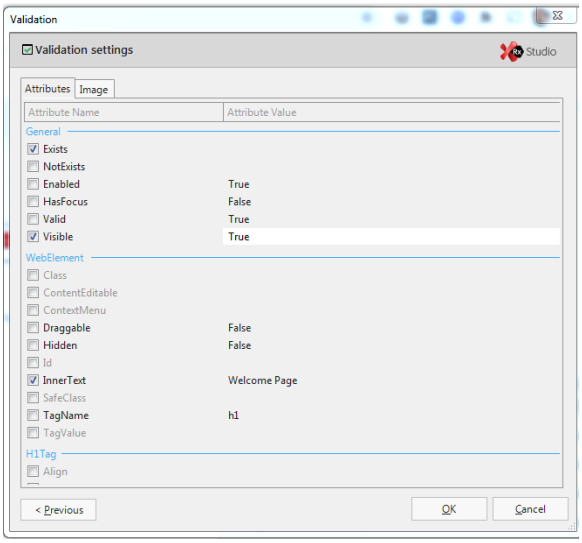
In order to implement a test, we need to add at least one validation.

For that, we choose "Validate" and move the mouse over the element we want to validate and click on it.

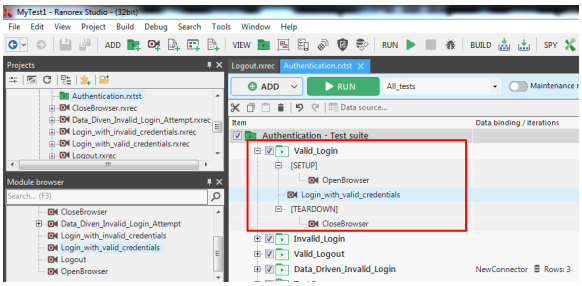




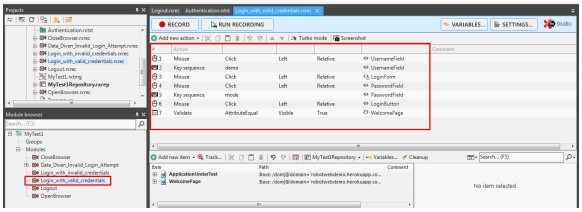
A validation dialog enables us to perform multiple asserts at the same time, on a given element (e.g. element exists, is visible, contains a given text). Each assert/validation will be created as a "Validate" action in our module.



A possible "Valid Login" test case could be composed of a setup section to open the browser (i.e. using the "OpenBrowser" module), a recorded module where we enter the credentials and validate the welcome page, followed by the close browser instruction (as part of the CloseBrowser module).

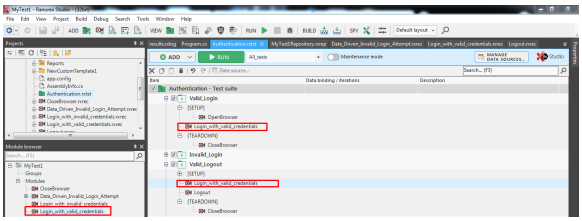


The "Login_with_valid_credentials" module could be composed of these actions, depending on the actions you defined earlier.



We may decide to implement additional tests, reusing existing modules.

As an example, a test case that checks the valid logout procedure (i.e. "Valid_Logout") can use the "Login_with_valid_credentials" module as the first macro step, before executing the module and its actions that perform the logout and verify its result ((i.e. "Logout" module).

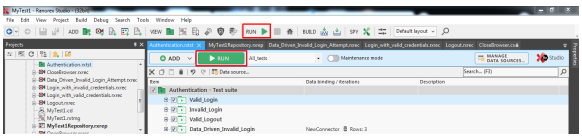


Running the tests

Before running the tests, you have to build the current project which will produce an executable file. Tests are in fact performed by this executable file and not by Ranorex Studio itself.

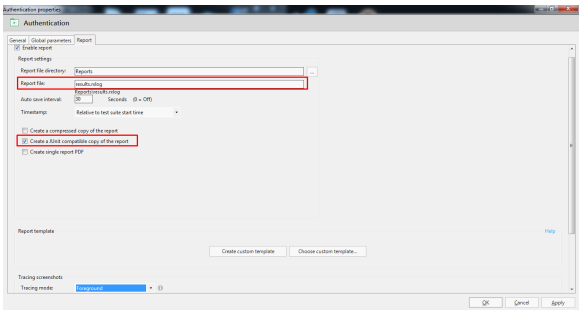
Running tests using Ranorex Studio

Test cases can be "run" from within the Ranorex Studio UI; a build happens in the background, if needed.



Whenever running, Ranorex Studio uses a run configuration to know which test cases to run; in the previous screenshot it is called "All_tests" and contains all test cases, since they're selected.

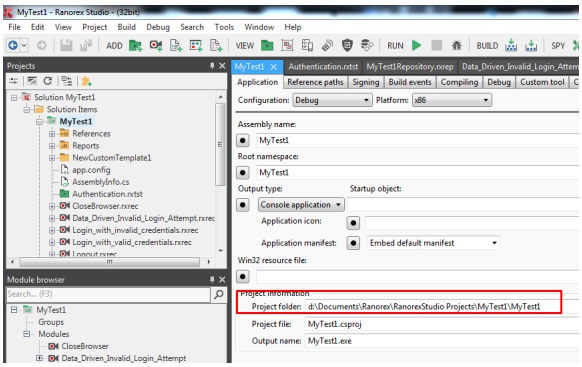
In the properties of our test suite we can customize the report file name, the reports directory, and enable the JUnit XML report which Xray can later on process.



Running tests from the command-line

Tests can be run from the command-line, by calling the executable built by Ranorex Studio.

The execute can be found inside the project folder (obtainable by looking at the project properties), either in the `bin\Debug` or `bin\Release` folder.



To run the tests, we execute the file and pass some arguments to enable the JUnit XML report, customize the report file base and file name; (use /help to find available options).

example of a shell script to run the tests

```
MyTest1.exe /junit /reportfile:results
```

In this case, the report will be stored in the current directory and will be named `results.rxlog.junit.xml`.

Integrating with Xray

In order to have visibility of our test automation results in Jira, we need to generate a JUnit XML report whenever running the tests, that can then be submitted to Xray as shown in the previous section.

To submit the report to Xray, we can use our favourite CI/CD tool or a simple script.

Once you have the report file available you can upload it to Xray through a request to the REST API endpoint for JUnit.

In the API request we can specify some common fields on the Test Execution, such as the target project, project version, linked test plan, etc.

```
curl -H "Content-Type: multipart/form-data" -u jira_username:jira_password
-F "file=@results.rxlog.junit.xml" jiraserver_base_url.example.com/rest
/raven/2.0/import/execution/junit?projectKey=XT
```

Sample batch (.bat) script to import results to Xray

```
@echo off
curl -H "Content-Type: multipart/form-data" -u %JIRA_USERNAME%:%
JIRA_PASSWORD% -F "file=@Reports\results.rxlog.junit.xml"
https://jiraserver_base_url.example.com/rest/raven/2.0/import/execution
/junit?projectKey=XT
```

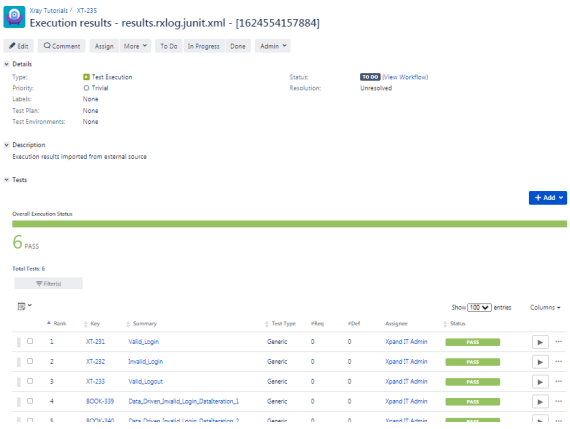
Sample PowerShell script to import results to Xray

```
try {
 $user = $env:JIRA_USERNAME
 $pass = $env:JIRA_PASSWORD
 $jira_base_url = 'https://jiraserver_base_url.example.com'
 $project_key = 'XT'
 $multipartFile = 'results.rxlog.junit.xml'

 $pair = "$($user):$($pass)"
 $encodedCreds = [System.Convert]::ToBase64String([System.Text.Encoding]::
ASCII.GetBytes($pair))
 $basicAuthValue = "Basic $encodedCreds"
 $multipartContent = New-Object System.Net.Http.MultipartFormDataContent
 '
 $FsMode = [System.IO.FileMode]::Open
 $FsAccess = [System.IO.FileAccess]::Read
 $FsSharing = [System.IO.FileShare]::Read
 $FileStream = New-Object System.IO.FileStream($multipartFile, $FsMode,
$FsAccess, $FsSharing)
 $fileHeader = New-Object System.Net.Http.Headers.
ContentDispositionHeaderValue("form-data")
 $fileHeader.Name = "file"
 $fileHeader.FileName = $multipartFile
 $fileContent = New-Object System.Net.Http.StreamContent($FileStream)
 $fileContent.Headers.ContentDisposition = $fileHeader
 $fileContent.Headers.ContentType = [System.Net.Http.Headers.
MediaTypeHeaderValue]::Parse("text/xml")
 $multipartContent.Add($fileContent)
 $uri = "$($jira_base_url)/rest/raven/2.0/import/execution/junit?
projectKey=$($project_key)"
 $res = Invoke-WebRequest -Uri $uri -Body $multipartContent -Method POST -
Headers @{"Authorization" = $basicAuthValue} }
catch {
 write-host $_.Exception.Message
}
```

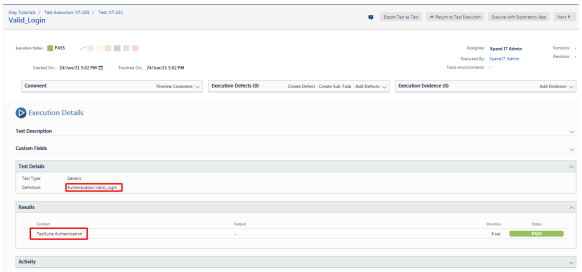
After submitting the test automation report, a Test Execution will be created in Jira, containing the results for each Test case.

A Test issue will be auto-provisioned, unless it already exists, per each Test Case.

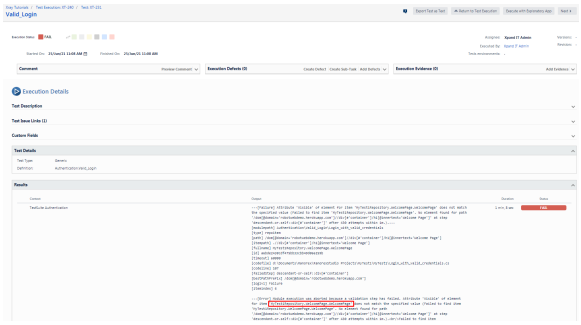


The Test Suite name along with the Test Case name will be used as unique identifier for the Generic Test that will be created.

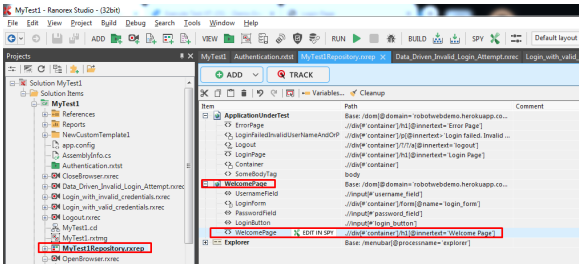
In the Test Run details of the corresponding Test in Xray, we can see this information along with the overall result. The original Test Suite name appears also as a context.



When a test case fails, the corresponding Test Run in Xray will show details about the repository item that failed; its full identifier includes the name of the repository followed by the hierarchical location of the repository item.



The failed element was in this case the "Welcome Page" header, in the Welcome Page, identified by "My Test1Repository.WelcomePage.WelcomePage".



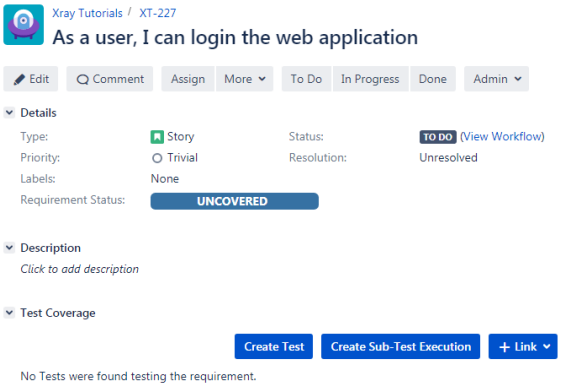
Tips

Seeing the impacts of test automation results on user stories or requirements

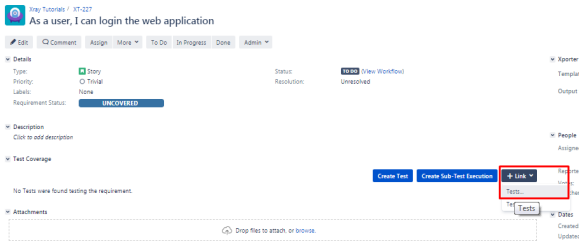
After uploading the test automation results, users can link the Test issues to existing user stories or requirements. That will enable users to track coverage and thus assess if user stories are covered by automated test scripts and if based on that, the corresponding user story can be considered OK or NOK.

Please remember that coverage is heuristic but it can still be quite helpful to assess the readiness of user stories, individually or at the release level.

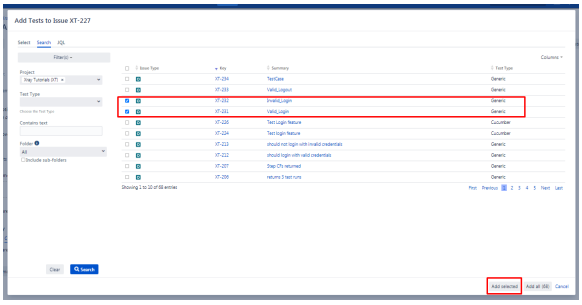
Assuming we have a user story (new or existing), we can then link it to the Tests that correspond to the Test Cases implemented using Ranorex.



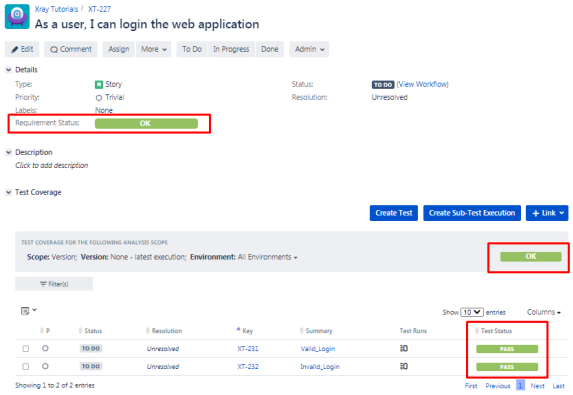
We can do that right from the user story issue screen, using the "Link" action.



Then, we select the Tests that were auto-provisioned earlier on upon the first import of the test report.



Finally, we can see the latest test results right from the user story issue screen along with the calculated coverage information for the user story; the latter is available (both on the Requirement Status custom field and also on the Test Coverage panel.



Any additional import of results, will appear automatically reflected on the user story issue screen as the Tests are already linked to the user story.

Run iterations and data-driven tests

Ranorex Studio has support for run iterations and data-driven tests.

These are two different concepts; while run iterations are just a way to run the same test case multiple times by executing the exact same modules and actions, data-driven tests will impact the action being performed (e.g. for exercising the same test case but with different inputs).

It's possible to have visibility of the corresponding test results in Jira using Xray but some care should be taken.

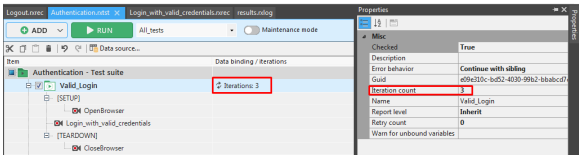
Please note

Due to the way Ranorex Studio reports these results on the JUnit XML report, different Test issues will be created for each run iteration or data row.

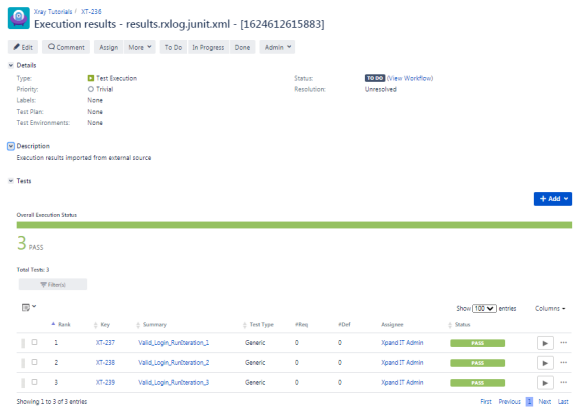
We should have this in mind as it becomes more difficult to manage these Test issues (e.g. number of unrelated Tests, linkage to user stories).

Run iterations

On the properties of a test case, we can configure the number of run iterations (i.e. iteration count).

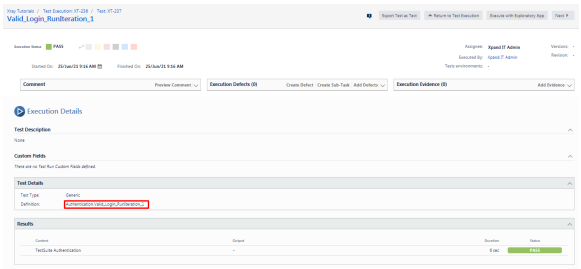


After running the test case and importing the results to Xray using the JUnit XML report, a Test Execution with three Tests is created in Xray.



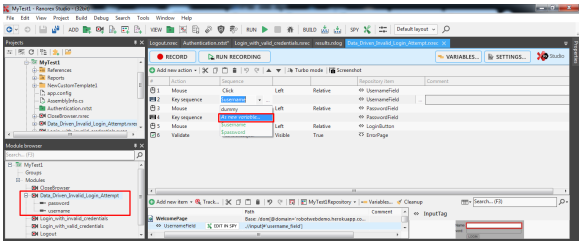
Due to the way run iterations are reported in the JUnit XML report, each run iteration for our test case is abstracted as a different Test issue, with the run iteration being part of its definition.

In other words, we'll have as many Test issues as the iteration count configured for the test case.

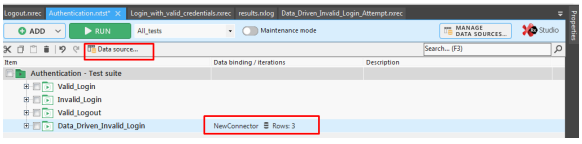


Data-driven tests

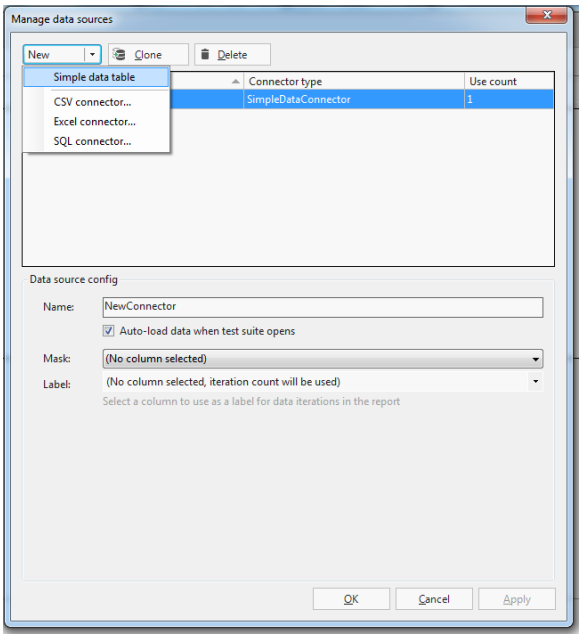
To make a data-driven test we need to use some variables in our actions, instead of using hardcoded values/strings.

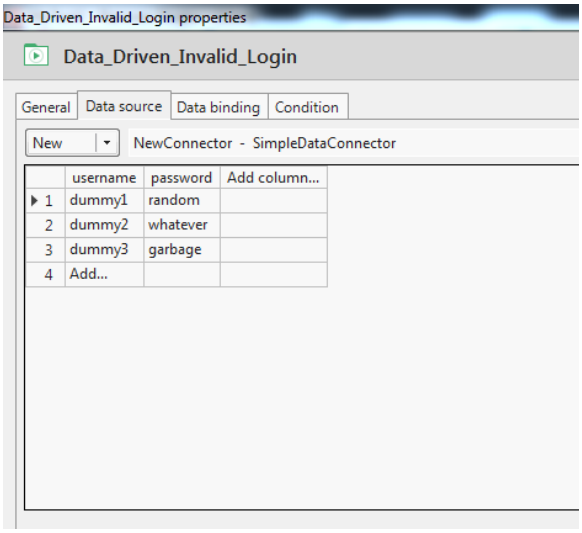


Then we need to configure a data-source for the selected test case.

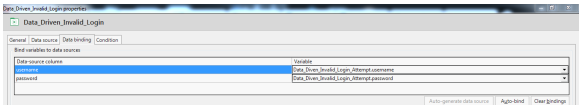


There are different types of data-sources; a simple built-in data table can be used to specify several named columns and some rows of data for them.

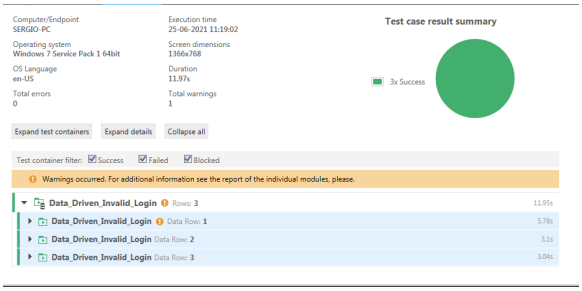




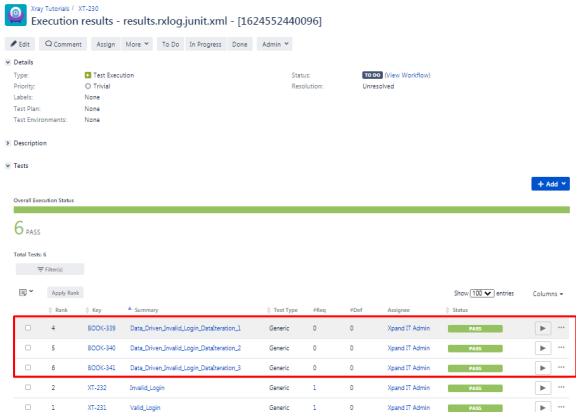
We then need to bind the columns of our data-source to the variables used in the previous module.



We can run the test case in Ranorex Studio and see the results for each data row.

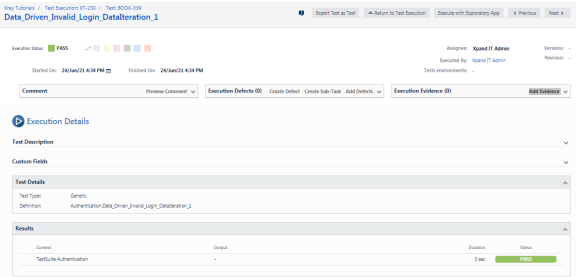


After running the test case and importing the results to Xray using the JUnit XML report, a Test Execution having three Tests related to our data-source is created in Xray.



Due to the way run iterations are reported in the JUnit XML report, each run iteration for our test case is abstracted as a different Test issue, with the run iteration being part of its definition.

In other words, we'll have as many Test issues as the iteration count configured for the test case.



Ranorex's built-in integration with Jira

Ranorex Studio has a built-in integration with Jira. It can be used, for example, to open bugs, resolve, and reopen them depending on testing results.

The built-in Jira integration is totally independent from the Xray integration described in the current article and may be eventually complementarily.

References

- Ranorex web site
- Ranorex User Guide
- Ranorex vs Selenium WebDriver
- Integrating Ranorex with Jenkins
    - blog post
    - documentation