# Working with Test Environments

> "Test Environment" is a concept introduced in Xray v2.
>
> Here is a short video that explains Test Environments and how you can use them.


- What is a Test Environment
- Benefits of using Test Environments
- Test Environments in Xray at a glance
- How it works
- How to use
- Using multiple environments at the same time
  - How to use
  - Example
- Advanced
  - Test Environments and the TestRunStatus custom field
- Tips and Recommendations
  - Do's
  - Don'ts


## What is a Test Environment

Generically speaking, a test environment is an environment that contains all necessary elements, including the SUT, so you can perform testing on it.

Depending on your context, a test environment may represent:

- a testing stage (e.g. "development", "staging", "preproduction", "production")
- a device model or device operating system (e.g. "Android", "iOS")
- an operating system (e.g. "Windows", "macOS", "Linux")
- browser (e.g. "Edge", "Chrome", "Firefox")

Thus, semantics of what a "Test Environment" represents depends on your specific context.

> **Please note**
>
> **In Xray, Test Environments are focused on the execution aspect, providing the means to schedule tests and analyze their results in different environments.**
>
> Thus, they're explicitly associated with Test Execution issues.

## Benefits of using Test Environments

- avoid duplication of Tests, whenever you have to run the same test on different environments
- ability to track the latest status of tests on different environments
- ability to track coverage on each environment
- ability to track overall coverage, considering the coverage/results on each environment
- ability to perform reporting, including traceability, per each environment or globally (i.e. considering all results on all different environments)

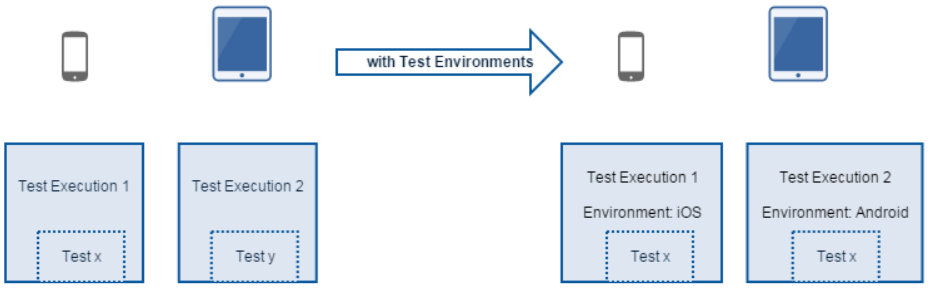
## Test Environments in Xray at a glance

The current status of a Test is generally calculated by looking at the Test's last execution (more detail here). However, this does not work well if you execute the same Test in two different test environments (e.g., devices) and you want the two results to be consolidated.

Within a Test Execution, you may specify the **Test Environment**(s) where the tests will be executed in the respective attribute. A Test Environment is similar to a label, but Xray has special logic to deal with it.

If you use Test Environments, you may reuse the same Test for multiple test environments and create Test Executions for each one. If you don't use Test Environments and you want to track tests for multiple environments (e.g., devices), then the only way to do it is to create multiple tests, one per each test environment.

Let's say that you have executions for two Test Environments: "Android" and "iOS". The test is considered as PASS only if the **latest** executions for Android and iOS are **both** PASS; otherwise, it will be FAIL.




Note: If you don't use Test Environments, then only the latest result matters for assessing the current status of the Test.

# How it works

Let's say we have a Test being executed two times. We will start by creating Test Executions TE1 and TE2.

TE2 will be executed after TE1, so TE1 is considered older.

Below are example scenarios and shows how the overall, **consolidated status** of the Test is calculated in each case.

If you have a mix of Test Executions (i.e., with and without Test Environment as in scenario D), it will be treated the same way as scenarios A, B, and C.

| Scenario | Test Environment (s) of TE 1 | Test Environment (s) of TE 2 | Test run status in TE1 | Test run status in TE2 | Calculated value for the overall, consolidated status of the Test (i.e. for the "All Environments") | Other |
|---|---|---|---|---|---|---|
| **A** | Android | iOS | PASS | PASS | **PASS** | The test will be considered to be PASS in both Android and iOS environments. |
| **B** | iOS | iOS | PASS | FAIL | **FAIL** | The test will be considered to be FAIL in iOS. |
| **C** | iOS | iOS | FAIL | PASS | **PASS** | The test will be considered to be PASS in iOS. |
| **D** | iOS | - | FAIL | PASS | **FAIL** | The test will be considered to be FAIL in iOS and PASS for the empty environment. |
| **E** | - | - | PASS | FAIL | **FAIL** | The test will be considered to be FAIL for the empty environment. |
| **F** | - | - | FAIL | PASS | **PASS** | The test will be considered to be PASS for the empty environment. |

**Please note**

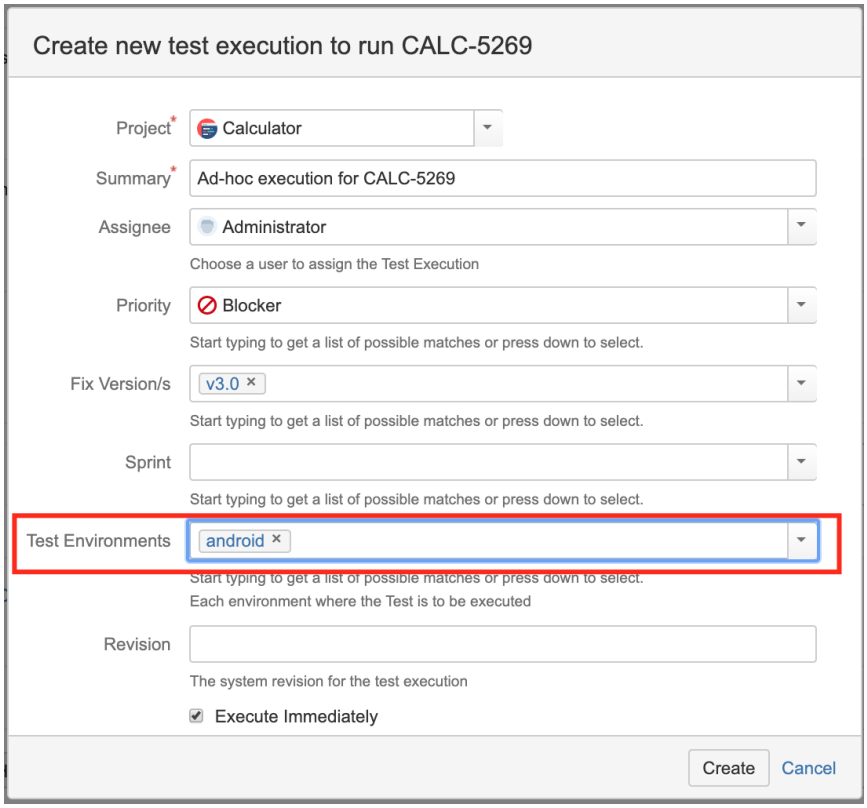
The *empty* Test Environment is treated similarly to any other environment having a well-defined name.

# How to use

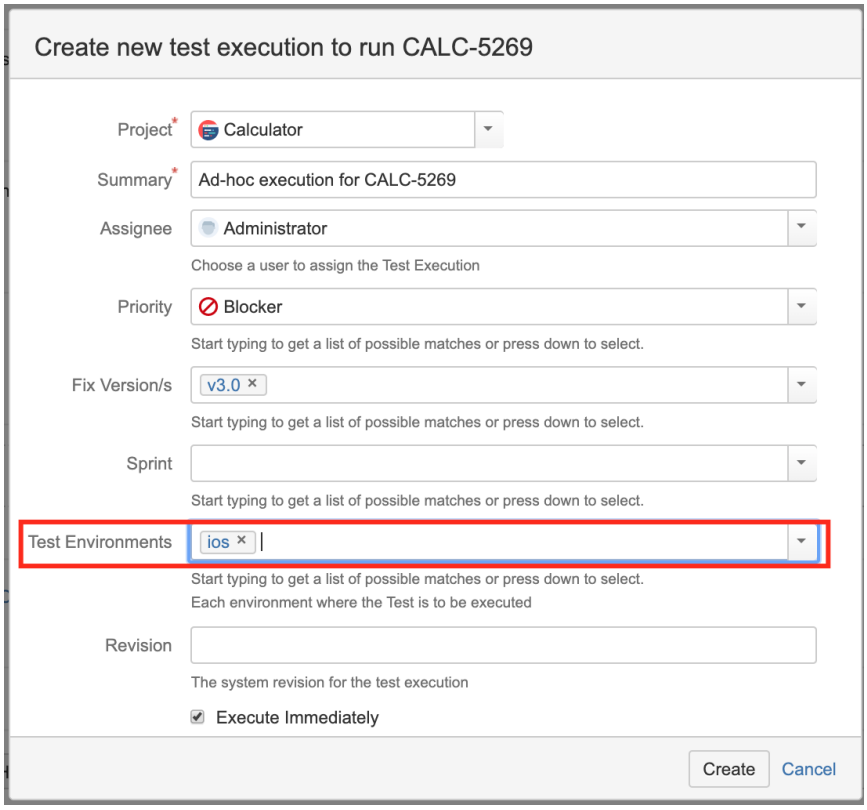
Whenever creating a Test Execution, you must set the Test Environment in which the execution will be executed. You can use this field as a simple label: just add the environment or reuse a previously created one.

Please see some important Tips and Recommendations ahead.

**Creating a Test Execution**




*Test Execution for “android” Test Environment*

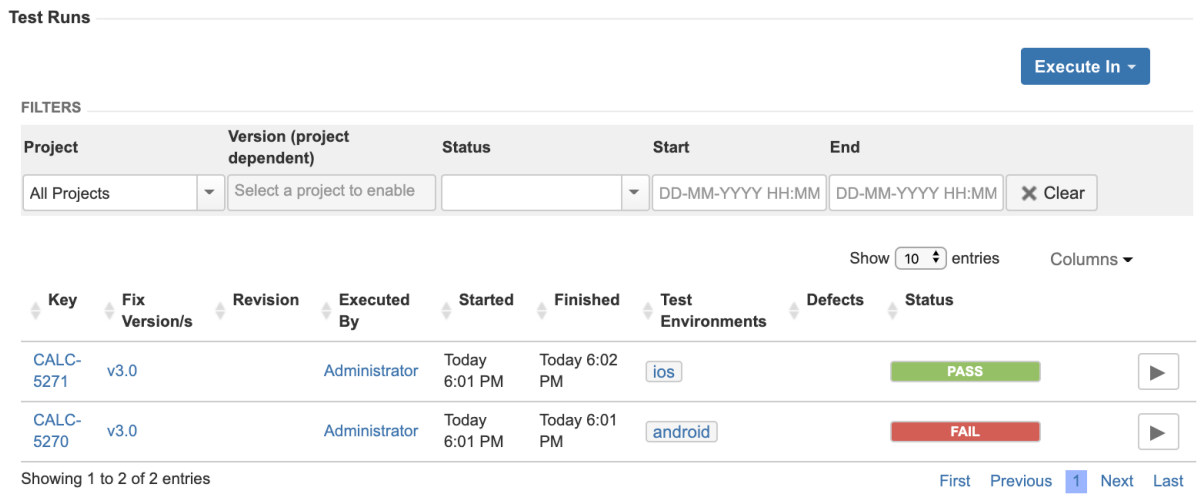



*Test Execution for “ios” Test Environment*

**Tracking the results on different environments**

The Test Environments column is shown in your Test Runs table so you can distinguish each execution of the Test between the different environments.

This information can be seen in the Test issue screen (see next screenshot) or in other places that show a list of Test Runs (e.g. Test Plan issue screen).

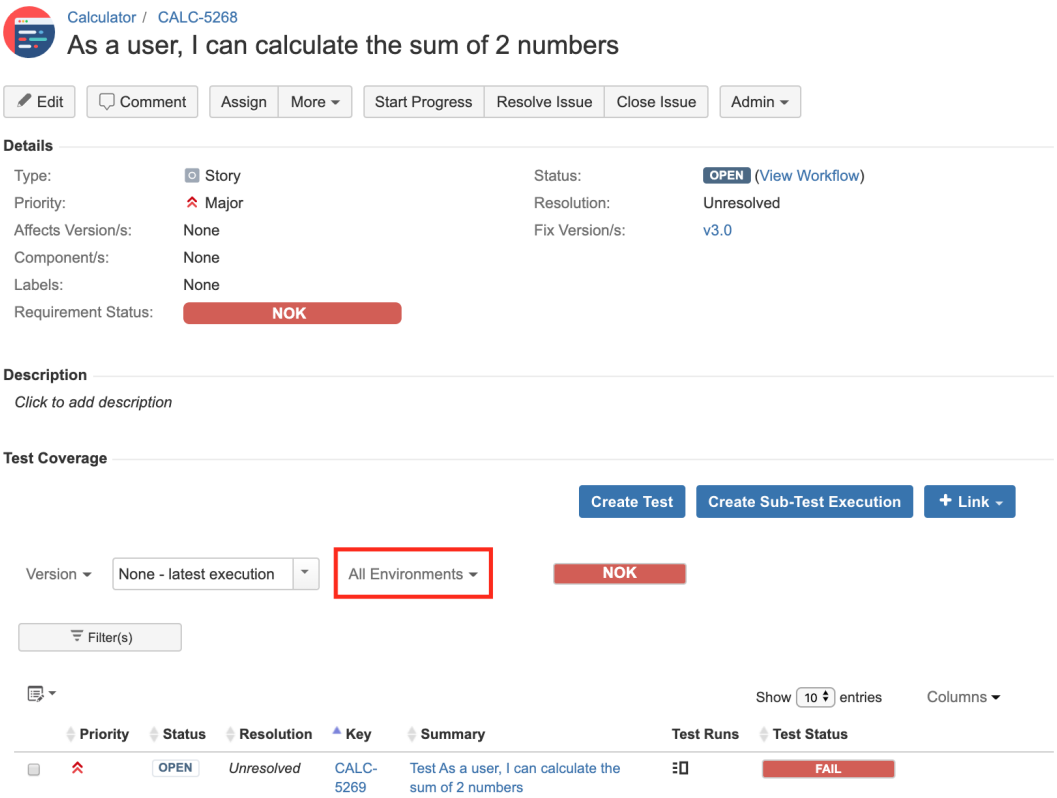



*The same test has been executed in both Test Environments (a Test Execution per Test Environment).*

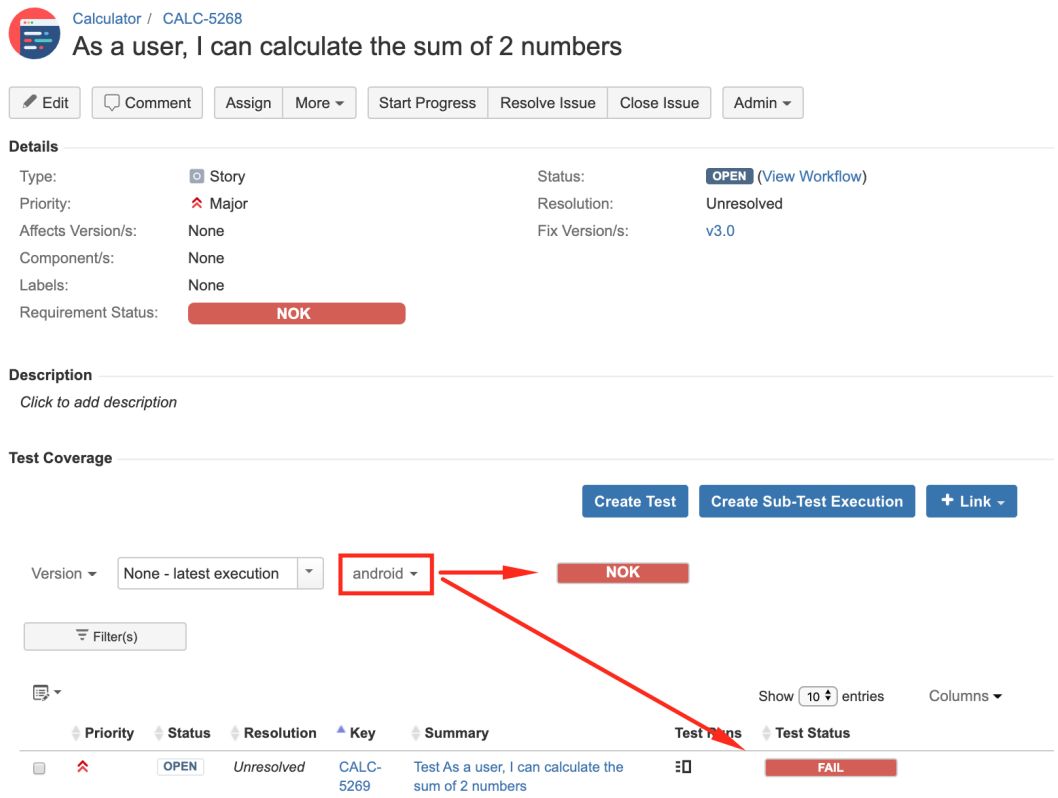
**Analyzing the impact of the results on different environments**

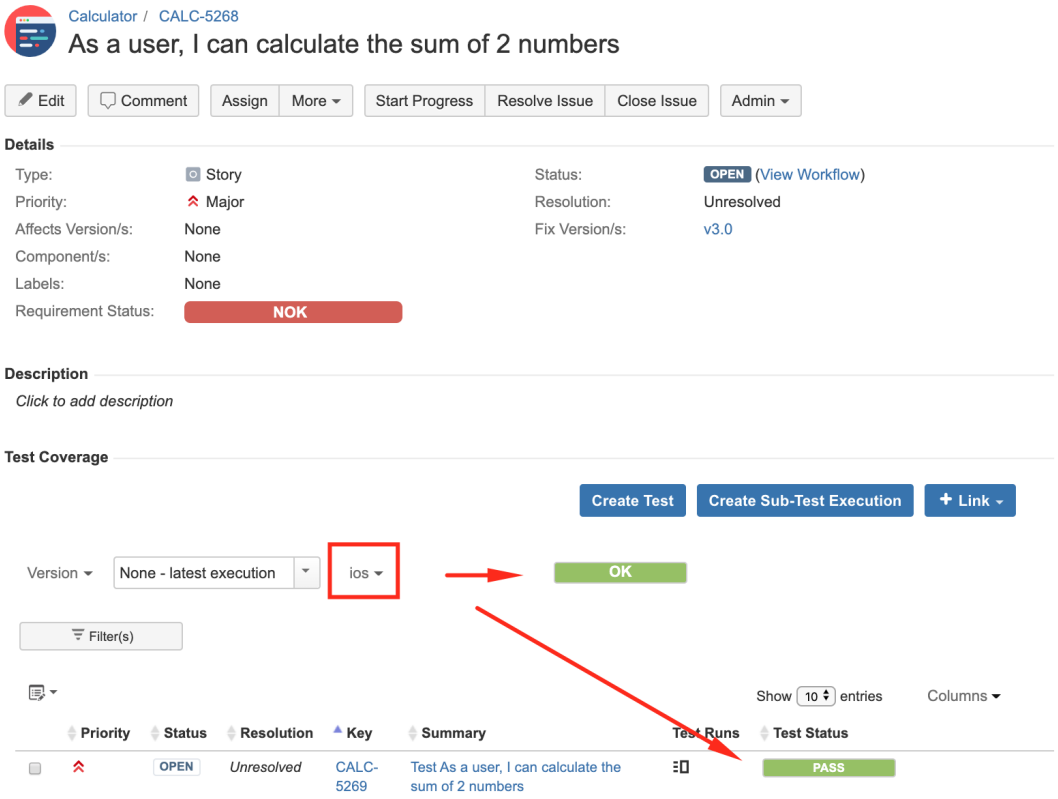
Results obtained for Test Environments will impact coverage.

Considering the previous screenshot, the "Requirement Status" custom field for the Test issue will show **NOK** because the Test has failed for one of the environments. This information is independent of the environment picker below within the "Test Coverage" section, which in turn is used to calculate the coverage on request for the selected scope, showing it on the right side along with the corresponding test results.




If you want to analyze the coverage for the requirement (e.g. "story") and show the latest results on that environment, just use the picker on the "Test Coverage" section. As seen ahead, this will produce different results because different results were obtained in different environments.

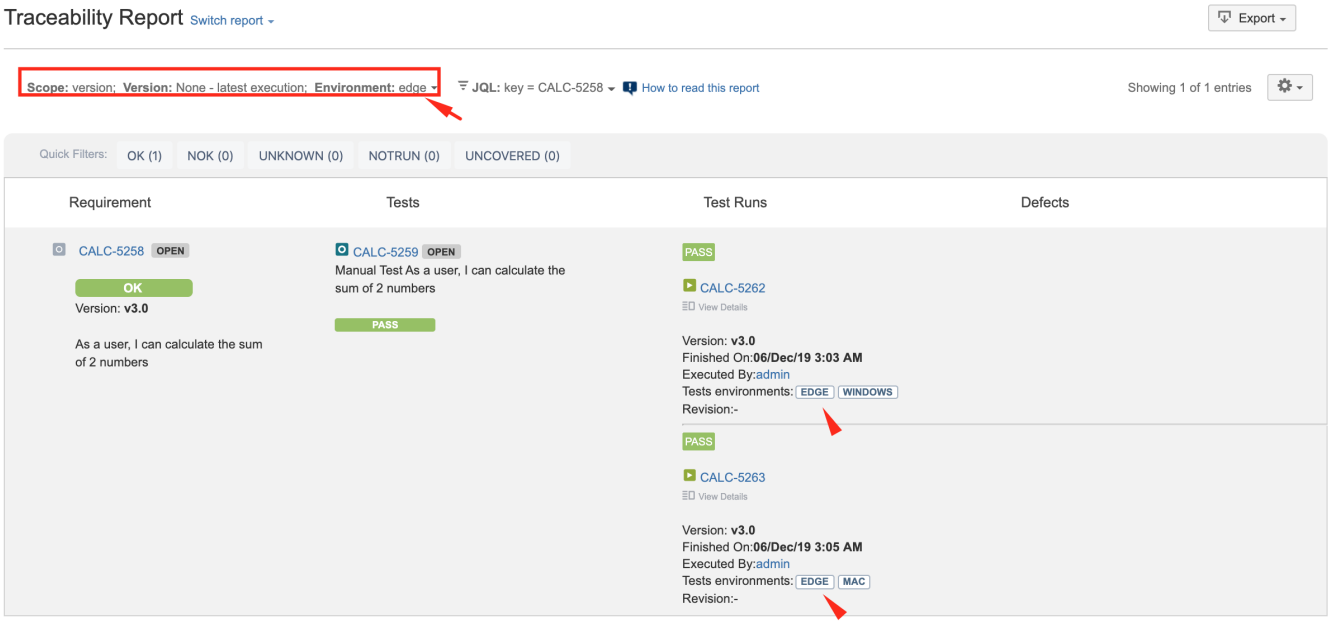


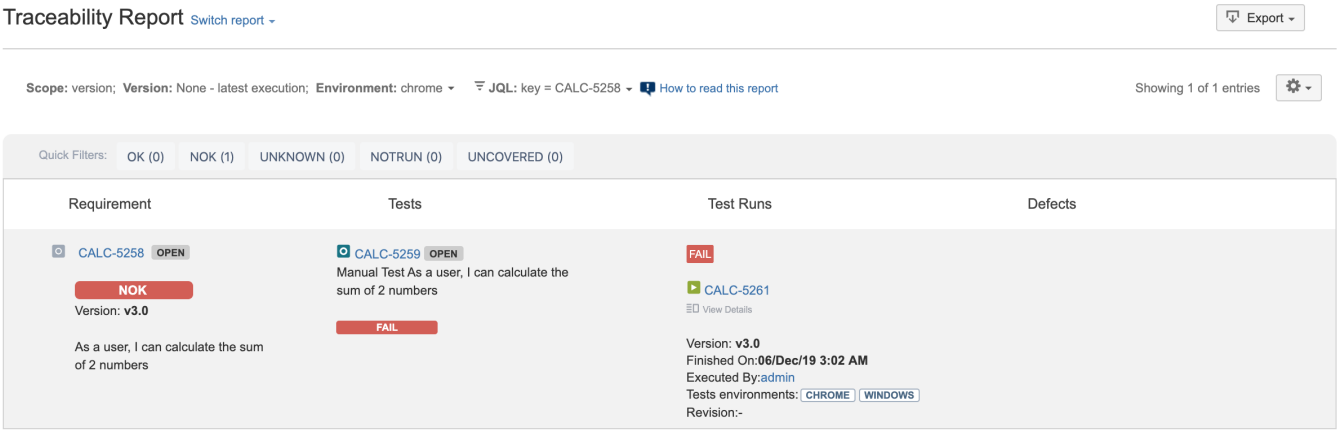

Please check Coverage Analysis to learn more about coverage analysis possibilities.

It is also possible to analyze testing thoroughly considering Test Environments; this analysis can be done using the Traceability Report or the Overall Coverage Report, among others.

The exact behavior upon choosing a specific Test Environment depends on the report itself but, either explicitly or implicitly, Test Runs will be filtered by the selected Test Environment and reports will reflect it.



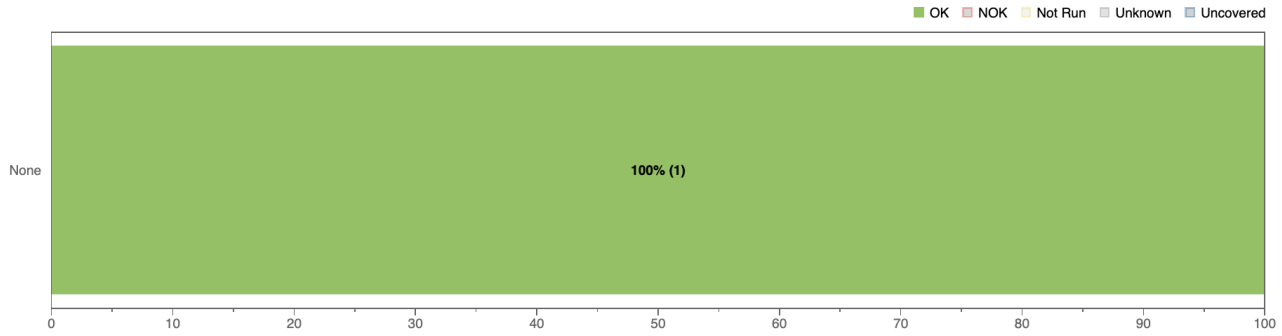

*Traceability Report being used to analyze the results on the "edge" test environment.*




*Traceability Report being used to analyze the results on the "chrome" test environment.*

## Overall Requirement Coverage Report Switch report

**Scope:** version; **Version:** None - latest execution; **Environment:** edge  Component/s  **JQL:** issue = CALC-5258  How to read this report




Requirements with Component/s "None" and Status "OK"

View Issues

Show 10 entries

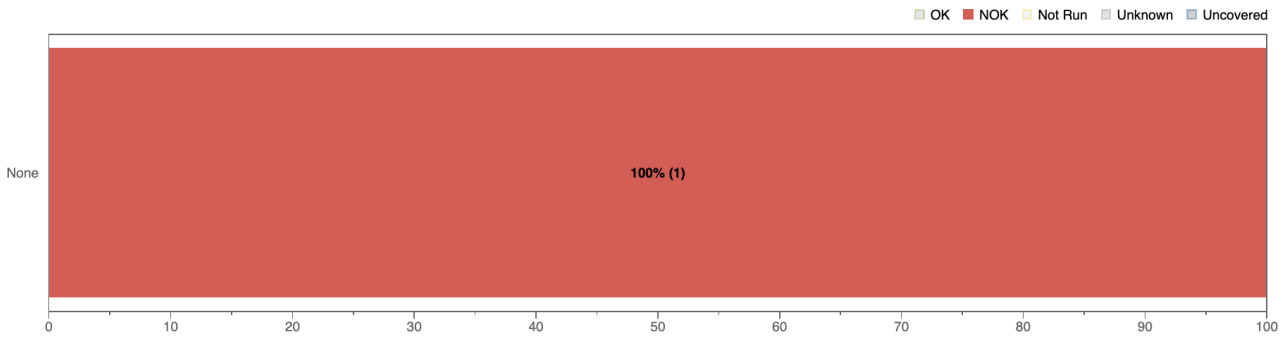
Search

| Key | Summary | Total Tests | Tests Passed | Tests Failed | Tests Unknown | Completeness |
|---|---|---|---|---|---|---|
| CALC-5258 | As a user, I can calculate the sum of 2 numbers | 1 | 1 | 0 | 0 | 100% |

*Analyzing coverage of "requirements" on the "edge" test environment.*

## Overall Requirement Coverage Report Switch report

**Scope:** version; **Version:** None - latest execution; **Environment:** chrome  Component/s  **JQL:** issue = CALC-5258  How to read this report




Requirements with Component/s "None" and Status "NOK"

View Issues

Show 10 entries

Search

| Key | Summary | Total Tests | Tests Passed | Tests Failed | Tests Unknown | Completeness |
|---|---|---|---|---|---|---|
| CALC-5258 | As a user, I can calculate the sum of 2 numbers | 1 | 0 | 1 | 0 | 0% |

*Analyzing coverage of "requirements" on the "chrome" test environment.*

# Using multiple environments at the same time

Sometimes, you may have multiple categorizations for a given environment; in theory, you can think as it being something multidimensional.

Consider a very basic example: whenever performing web/UI based testing you will be using a browser and an operating system and you may want to analyze the results per a browser perspective or per an operating system perspective.

**The recommended way to deal with environments having multiple dimensions is to treat each dimension (e.g. browser name, operating system name) individually. In other words, add the values of each dimension to the "Test Environments" field separately.**

Whenever you assign "mac" and "edge" to the Test Environments of a given Test Execution, it's equivalent to saying that your Test Run is scheduled for /was run in the "mac" and also in the "edge" environment.

This approach will limit the number of environments to the total number of possible values for each dimension, as opposed to having *<number_of_values_dimension_1>*<number_of_values_dimension_2>**... environments.

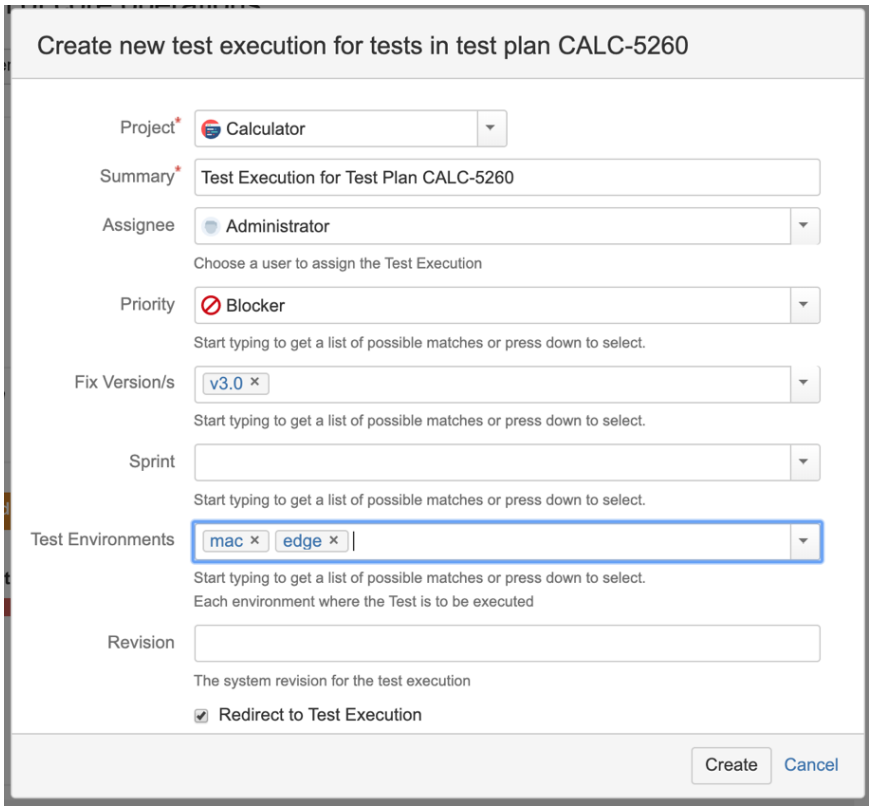
The drawback of this solution is that you won't be able to analyze the results for an environment tagged as "mac" and "edge" at the same time, for example; you can just analyze results from a specific dimension.

> **To have in mind**
>
> One way to deal with these kinds of environments would be to flatten them and treat them as usual, i.e. you could name the environment such as “windows_edge” or “mac_chrome” but…
>
> - You could have a ton of composed environments which wouldn’t be manageable at all
> - You couldn’t analyze coverage just from the perspective of one of those variables (e.g. “mac” or “edge”); you would be restricted to analyze it from the perspective of the composed environment

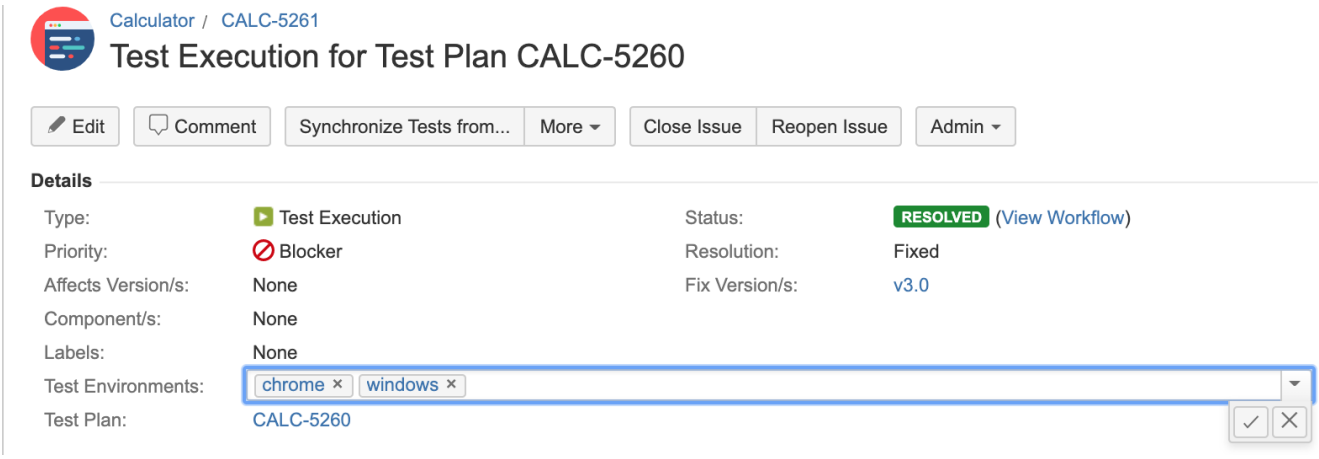
## How to use

Assign each environment (e.g. name of operation system, name of browser vendor) as you do for a single environment; in other words, just add the multiple environment names as multiple, distinct labels.

**Whenever creating a Test Execution (e.g. from a Test Plan)**




**Whenever updating an existing Test Execution**




## Example

**<u>Test executed in the context of Test Execution assigned to several environments at the same time</u>**

1. windows, chrome (fail)
2. windows, edge (pass)
3. mac, edge (pass)

| Key | Fix Version/s | Revision | Executed By | Started | Finished | Test Environments | Defects | Status | |
|---|---|---|---|---|---|---|---|---|---|
| CALC-5261 | v3.0 | | Administrator | 06/Dec/19 3:02 AM | 06/Dec/19 3:02 AM | chrome windows | | FAIL | ▶ |
| CALC-5262 | v3.0 | | Administrator | 06/Dec/19 3:03 AM | 06/Dec/19 3:03 AM | edge windows | | PASS | ▶ |
| CALC-5263 | v3.0 | | Administrator | 06/Dec/19 3:05 AM | 06/Dec/19 3:05 AM | edge mac | | PASS | ▶ |

The calculated status of the test, per Test environment, will be the following.

| | Status | Why? |
|---|---|---|
| **windows** | **PASS** | due to the last result obtained in "windows" environment on CALC-5262 |
| **mac** | **PASS** | due to the last result obtained in the "mac" environment on CALC-5263 |
| **chrome** | **FAIL** | due to the last result obtained in "chrome" environment on CALC-5261 |
| **edge** | **PASS** | due to the last result obtained in "edge" environment on CALC-5263 |
| **"All Environments"**<br><br>**(if analyzing the status of the test without identifying a specific environment)** | **FAIL** | as the last result for one of the environments ("chrome") was FAIL (i. e. on CALC-5261) |

# Advanced

## Test Environments and the TestRunStatus custom field

The "TestRunStatus" custom field is associated with Test issues and can be used to provide information about the latest status of your test; more info here.

This custom field calculates the status of the test for "all environments" (i.e. the consolidated status), giving you a high-level view; it cannot be configured to show the status for a specific environment.

Internally, this field will store the status of the test for all possible scopes, which besides other things includes the information about the status in all different environments.

> ⓘ If you start using Test Environments in your Test Executions, then it's not only your test status calculation that will change (i.e. the one stored in the TestRunStatus custom field). All custom fields that depend on it (e.g., Requirement status, Test Sets status) will change. Consequently, the requirement coverage calculation and all associated charts/gadgets are also affected.

# Tips and Recommendations

## Do's

- Use Test Environments only if you want to run the **same** Test case in different environments and track results per each environment.
- Simplify the names of Test Environments (i.e. lowercase it, shorten it)
    - Example: macOS => mac
- Evaluate if you really need to assign multiple environments at the same time; using just one is preferable if you can afford that simplicity

**Learn more**

For advanced Test Environment management capabilities, please check our Integration with Apwide Golive.

## Don'ts

- Don't create thousands of Test Environments as it will harden their usage and add some performance overhead
- Don't try to do data-driven testing using Test Environments; they're not tailored for that

**Please note**

Besides other usage issues, if you have a large number of environments (>500), it will impact the calculations that need to be done and the size of the Lucene index.

Please try to have a limited, restricted, and well-defined list of Test Environments.