Import Execution Results

External execution results from either automated and manual Tests can be imported to Jira. This operation may be done in one of two ways:

- **Manually**, using the *Import Execution Results* action accessible from the Test Execution issue screen
- via the **REST API** to integrate with Continuous Integration (CI) platforms or other external execution processes. Please refer to the Xray REST API

> When manually importing execution results, the current Test Execution issue will be updated with the results. When using the REST API, you can specify an existing Test Execution issue in Jira; if the Test Execution key is missing, Xray will create a new Test Execution issue based on the information provided.

Supported Import Formats

Xray supports the following formats for importing execution results:

1. Cucumber JSON output format
2. JUnit XML output format
3. NUnit XML output format
4. Xray JSON format

Cucumber JSON output format

The Cucumber tool is capable of generating multiple reports for an execution. In order to import the execution results to Xray, Cucumber must generate a JSON output (example here) using the following arguments:

```
-f, --format FORMAT       How to format features. Available formats compatible with Xray for JIRA:
                     json        : Prints the feature as JSON
                json_pretty : Prints the feature as prettified JSON
-x, --expand              Expand Scenario Outline Tables in output.

Ex:
> cucumber -x -f json
```

JUnit XML output format

Xray supports a JUnit XML format for importing execution results.

```
<?xml version="1.0" encoding="UTF-8"?>
<testsuite tests="1" failures="0" name="ut.com.xpandit.raven.service.impl.IssueDataSetTest" time="0.114"
errors="0" skipped="0">
  <properties>
    ...
  </properties>
  <testcase classname="ut.com.xpandit.raven.service.impl.IssueDataSetTest" name="
testApplyOptions_withValidIssueAndValidLimitOverflowOption_returnsExpectedSubset" time="0.114"/>
</testsuite>
```

Xray-JUnit XML extension format

Users are also able to import JUnit results using the Xray extended format.

This format adds the following property elements:

- testrun_comment
- requirements
- test_key
- test_id
- test_description
- test_summary
- tags
- testrun_customfields
- testrun_evidence

For more detailed information, visit the Xray App GitHub repository: https://github.com/Xray-App/xray-junit-extensions.

NUnit v3.0 XML output format

Xray supports an NUnit v3.0 XML format for importing execution results.

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<test-run id="0" testcasecount="3" total="3" passed="3" failed="0" inconclusive="0" skipped="0" asserts="3"
result="Passed" portable-engine-version="3.3.0.0" start-time="2016-12-26 14:36:03Z" end-time="2016-12-26 14:36:
03Z" duration="0.015218">
  <test-suite type="Assembly" id="1021" name="x, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null"
fullname="x, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null" runstate="Runnable" testcasecount="14"
result="Failed" site="Child" start-time="2016-12-26 14:36:03Z" end-time="2016-12-26 14:36:03Z" duration="
0.015218" total="3" passed="3" failed="0" inconclusive="0" skipped="0" asserts="3">
    <settings>
      ...
    </settings>
    <test-suite type="TestSuite" id="1022" name="x" fullname="x" runstate="Runnable" testcasecount="12" result="
Passed" start-time="2016-12-26 14:36:03Z" end-time="2016-12-26 14:36:03Z" duration="0.015218" total="3" passed="
3" failed="0" inconclusive="0" skipped="0" asserts="3">
    <test-suite type="TestFixture" id="1004" name="CalculatorTests" fullname="x.CalculatorTests" classname="x.
CalculatorTests" runstate="Runnable" testcasecount="12" result="Passed" start-time="2016-12-26 14:36:03Z" end-
time="2016-12-26 14:36:03Z" duration="0.014979" total="3" passed="3" failed="0" inconclusive="0" skipped="0"
asserts="3">
      <test-suite type="ParameterizedMethod" id="1008" name="CanAddNumbers" fullname="x.CalculatorTests.
CanAddNumbers" classname="x.CalculatorTests" runstate="Runnable" testcasecount="3" result="Passed" start-time="
2016-12-26 14:36:03Z" end-time="2016-12-26 14:36:03Z" duration="0.004228" total="3" passed="3" failed="0"
inconclusive="0" skipped="0" asserts="3">
        <properties>
              <property name="Requirement" value="DEV-771" />
            </properties>
    <test-case id="1005" name="CanAddNumbers(1,1,2)" fullname="x.CalculatorTests.CanAddNumbers(1,1,2)"
methodname="CanAddNumbers" classname="x.CalculatorTests" runstate="Runnable" seed="1846389584" result="Passed"
start-time="2016-12-26 14:36:03Z" end-time="2016-12-26 14:36:03Z" duration="0.001194" asserts="1" />
    <test-case id="1006" name="CanAddNumbers(-1,-1,-2)" fullname="x.CalculatorTests.CanAddNumbers(-1,-1,-2)"
methodname="CanAddNumbers" classname="x.CalculatorTests" runstate="Runnable" seed="1113780989" result="Passed"
start-time="2016-12-26 14:36:03Z" end-time="2016-12-26 14:36:03Z" duration="0.000067" asserts="1" />
    <test-case id="1007" name="CanAddNumbers(100,5,105)" fullname="x.CalculatorTests.CanAddNumbers(100,5,105)"
methodname="CanAddNumbers" classname="x.CalculatorTests" runstate="Runnable" seed="1585332966" result="Passed"
start-time="2016-12-26 14:36:03Z" end-time="2016-12-26 14:36:03Z" duration="0.000103" asserts="1" />
    </test-suite>
  </test-suite>
  </test-suite>
</test-suite>
</test-run>
```


NUnit v2.6 XML output format

Xray supports an NUnit v2.6 XML format for importing execution results.

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<test-results name="MobilityServices.ShoppingLists.Tests\bin\Debug\MobilityServices.ShoppingLists.Tests.dll"
total="2" errors="0" failures="0" not-run="0" inconclusive="0" ignored="0" skipped="0" invalid="0" date="2017-
01-18" time="15:46:53">
  <environment nunit-version="2.6.4.14350" clr-version="2.0.50727.8745" os-version="Microsoft Windows NT
6.2.9200.0" platform="Win32NT" cwd="D:\Work\Apps\NUnit-2.6.4\bin" machine-name="LAPTOP-UNKNOWN" user="unknown"
user-domain="LAPTOP-UNKNOWN" />
  <culture-info current-culture="pt-PT" current-uiculture="en-GB" />
  <test-suite type="Assembly" name="MobilityServices.ShoppingLists.Tests\bin\Debug\MobilityServices.
ShoppingLists.Tests.dll" executed="True" result="Success" success="True" time="4.473" asserts="0">
    <results>
      <test-suite type="Namespace" name="MobilityServices" executed="True" result="Success" success="True"
time="4.473" asserts="0">
        <results>
          <test-suite type="Namespace" name="ShoppingLists" executed="True" result="Success" success="True"
time="4.473" asserts="0">
            <results>
              <test-suite type="Namespace" name="Tests" executed="True" result="Success" success="True" time="
4.473" asserts="0">
                <results>
                  <test-suite type="Namespace" name="Controllers" executed="True" result="Success" success="
True" time="4.473" asserts="0">
                    <results>
                      <test-suite type="TestFixture" name="ShoppingListsControllerTests" executed="True"
result="Success" success="True" time="4.473" asserts="0">
                        <results>
                          <test-case name="JMMobilityServices.ShoppingLists.Tests.Controllers.
ShoppingListsControllerTests.RemoveShoppingListItem" executed="True" result="Success" success="True" time="
4.118" asserts="2" />
                          <test-case name="JMMobilityServices.ShoppingLists.Tests.Controllers.
ShoppingListsControllerTests.TestAuthorization" executed="True" result="Success" success="True" time="0.355"
asserts="1">
                            <properties>
                              <property name="Requirement" value="DEV-771" />
                            </properties>
                          </test-case>
                        </results>
                      </test-suite>
                    </results>
                  </test-suite>
                </results>
              </test-suite>
            </results>
          </test-suite>
        </results>
      </test-suite>
    </results>
  </test-suite>
</test-results>
```


Xray JSON format

Xray also provides a proprietary JSON format for importing execution results.

Although Xray supports multiple report formats used by different testing frameworks/runners (e.g. JUnit, NUnit, xUnit, TestNG, Cucumber, Robot Framework), there are scenarios where using these formats is not an option like:

- Importing results for manual Tests;
- Using a testing framework report that is not supported by Xray;
- Having your own testing framework;
- Limited support of existing formats to import detailed execution results back into Jira.

Importing Execution Evidence

This format supports importing execution evidence (attachments) embedded in the JSON file encoded in ***Base 64***.

Creating new Test Execution issues

The **info** object allows you to specify specific execution information when creating new Test Execution issues.

Importing results to Manual Tests

You can import manual Test execution results using the **steps** element for specifying the Test step results.

testExecutionKey	The test execution key where to import the execution results
info	The info object for creating new Test Execution issues (link)
tests	The Test Run result details (link)

"info" object - Test Execution issue

You can specify which Test Execution issue to import the results by setting the test execution key on the **testExecutionKey** property. Alternatively, you can create a new Test Execution issue for the execution results and specify the issue fields within the **info** object.

project	The project key where the test execution will be created
summary	The summary for the test execution issue
description	The description for the test execution issue
version	The version name for the Fix Version field of the test execution issue
revision	A revision for the revision custom field
user	The username for the Jira user who executed the tests
startDate	The start date for the test execution issue
finishDate	The finish date for the test execution issue
testPlanKey	The test plan key for associating the test execution issue
testEnvironments	The test environments for the test execution issue

"tests" object - Test Run details

The test run details object allows you to import any detail about the execution itself. All Xray test types are supported.

It is possible to import a **single** result (the test object itself with the **"steps"** (Manual tests) or **"examples"** (BDD tests)) or **multiple** execution results into the same Test Run (data-driven testing) using the **"iterations"** array.

testKey	The test issue key
testInfo	The testInfo element (link)
start	The start date for the test run
finish	The finish date for the test run
comment	The comment for the test run
executedBy	The user id who executed the test run
assignee	The user id for the assignee of the test run
status	The test run status (PASSED, FAILED, EXECUTING, TODO, custom statuses ...)
steps	The step results (link)
examples	The example results for BDD tests (link)
iterations	The iteration containing data-driven test results (link)

defects	An array of defect issue keys to associate with the test run
evidence	An array of evidence items of the test run (link)
customFields	An array of custom fields for the test run (link)

"testInfo" object - Creating Test issues

It is possible to create new test issues when importing execution results using the Xray JSON format. For this, a **testInfo** element must be provided in order for Xray to create the issues.

If it is the first time you are importing an execution with a **testInfo,** Xray will create the tests automatically. Subsequent executions will reuse the same test issues.

Xray will first try to match test issues by the **testKey** if present. Otherwise, **Manual or BDD** tests are matched by **summary** whilst **Generic** tests are matched using the **generic definition** field, within the **same project**.

Any changes to the **testInfo** element will update the test issue specification in Jira.
If the match field (summary or definition) is changed, Xray will search for another issue and will create a new test case, or update an existing test case if no one is found. If you need to change the summary or the definition, you can do it manually (go to Jira and change the field), or you can include the **testKey** within the **test** element.

projectKey	The project key where the test issue will be created
summary	The summary for the test issue
description	The description of the test issue
testType	The test type (e.g. Manual, Cucumber, Generic)
requirementKeys	An array of requirement issue keys to associate with the test
labels	The test issue labels
steps	An array of test steps (for Manual tests) (link)
definition	The generic test definition
scenario	The BDD scenario
scenarioType	The BDD scenario type (Scenario or Scenario Outline)

"steps" object - step definition

This object allows you to define the step fields for manual tests.

action	The step action - native field
data	The step data - native field
result	The step expected result - native field

"iterations" object - Data-driven test results

If you need to import data-driven test results you need to use the iterations object. Xray will store all iterations within the same Test Run object.

It is also possible to import iteration results with parameters. **Currently, this is only supported for manual tests.**

In this case, Xray will create a dataset automatically within the Test Run object.

name	The iteration name
parameters	An array of parameters along with their values (link)
status	The status for the iteration (PASSED, FAILED, EXECUTING, TODO, custom statuses ...)

"parameters" object - parameters within iteration results

name	The parameter name
value	The parameter value

"evidence" object - embedded attachments

data	The attachment data encoded in **base64**
filename	The file name for the attachment
contentType	The Content-Type representation header is used to indicate the original media type of the resource

"customFields" object - store test run custom fields

It is possible to import test run custom field values into the Test Run object. Xray will use the **"id"** to find the existing test run custom field in the project settings.

id	The test run custom field ID
value	The test run custom field value

Xray JSON Schema

The JSON results file must comply with the following JSON Schema:

```
{
  "$id": "XraySchema",
  "type": "object",
  "properties": {
    "testExecutionKey": {
      "type": "string"
    },
      "info": {
      "type": "object",
      "properties": {
        "project": {
          "type": "string"
        },
        "summary": {
          "type": "string"
        },
        "description": {
          "type": "string"
        },
        "version": {
          "type": "string"
        },
        "revision": {
          "type": "string"
        },
        "user": {
          "type": "string"
        },
        "startDate": {
          "type": "string",
          "format": "date-time"
        },
        "finishDate": {
          "type": "string",
          "format": "date-time"
        },
        "testPlanKey": {
          "type": "string"
        },
        "testEnvironments": {
          "type": "array",
          "items": {
            "type": "string"
          }
        }
      },
      "additionalProperties": false
```

```
    },
    "tests": {
      "type": "array",
      "items": {
          "$ref": "#/definitions/Test"
      },
      "minItems": 1,
    }
  },
  "additionalProperties": false,

  "definitions": {

    "Test": {
      "type": "object",
      "properties": {
        "testKey": {
          "type": "string"
        },
        "testInfo": {
          "$ref": "#/definitions/TestInfo"
        },
        "start": {
          "type": "string",
          "format": "date-time"
        },
        "finish": {
          "type": "string",
          "format": "date-time"
        },
        "comment": {
          "type": "string"
        },
        "executedBy": {
          "type": "string"
        },
        "assignee": {
          "type": "string"
        },
        "status": {
          "type": "string"
        },
        "steps": {
          "type": "array",
          "items": {
            "$ref": "#/definitions/ManualTestStepResult"
          }
        },
        "examples": {
          "type": "array",
          "items": {
            "type": "string",
            "enum": ["TODO", "FAILED", "PASSED", "EXECUTING"]
          }
        },
        "iterations": {
          "type": "array",
          "items": {
            "$ref": "#/definitions/IterationResult"
          }
        },
        "defects": {
          "type": "array",
          "items": {
            "type": "string"
          }
        },
        "evidence": {
          "type": "array",
          "items": {
            "$ref": "#/definitions/EvidenceItem"
```

```
        }
      },
      "evidences": { // DEPRECATED
        "type": "array",
        "items": {
          "$ref": "#/definitions/EvidenceItem"
        }
      },
      "customFields": {
        "$ref": "#/definitions/CustomField"
      }
    },
    "required": ["status"],
    "dependencies": {
      "evidence": {
        "not": { "required": ["evidences"] }
      },
      "evidences": {
        "not": { "required": ["evidence"] }
      },
      "steps": {
        "allOf": [
          {
            "not": { "required": ["examples"] }
          },
          {
            "not": { "required": ["iterations"] }
          }
        ]
      },
      "examples": {
        "allOf": [
          {
            "not": { "required": ["steps"] }
          },
          {
            "not": { "required": ["iterations"] }
          }
        ]
      },
      "iterations": {
        "allOf": [
          {
            "not": { "required": ["steps"] }
          },
          {
            "not": { "required": ["examples"] }
          }
        ]
      }
    },
    "additionalProperties": false
  },

  "IterationResult": {
    "type": "object",
    "properties": {
      "parameters": {
        "type": "array",
        "items": {
          "type": "object",
          "properties": {
            "name": {
              "type": "string"
            },
            "value": {
              "type": "string"
            }
          },
          "additionalProperties": false
        }
```

```
    },
    "status": {
      "type": "string"
    },
    "steps": {
      "type": "array",
      "items": {
        "$ref": "#/definitions/ManualTestStepResult"
      }
    }
  },
  "required": ["status"],
  "additionalProperties": false
},

"ManualTestStepResult": {
  "type": "object",
  "properties": {
    "status": {
      "type": "string"
    },
    "comment": {
      "type": "string"
    },
    "evidences": {
      "type": "array",
      "items": {
        "$ref": "#/definitions/EvidenceItem"
      }
    },
    "defects": {
      "type": "array",
      "items": {
        "type": "string"
      }
    },
    "actualResult": {
      "type": "string"
    }
  },
  "required": ["status"],
  "additionalProperties": false
},

"TestInfo": {
  "type": "object",
  "properties": {
    "summary": {
      "type": "string"
    },
    "description": {
      "type": "string"
    },
    "projectKey": {
      "type": "string"
    },
    "requirementKeys":  {
      "type": "array",
      "items": {
        "type": "string"
      }
    },
    "labels": {
      "type": "array",
      "items": {
        "type": "string"
      }
    },
    "testType": {
      "type": "string"
    },
```

```
      steps: {
        type: "array",
        items: {
          type: "object",
          properties: {
            action: {
              type: "string"
            },
            data: {
              type: "string"
            },
            result: {
              type: "string"
            }
          },
          required: ["action"],
          additionalProperties: false
        }
      },
      "scenario": {
        "type": "string"
      },
      "scenarioType": {
        "type": "string"
      },
      "definition": {
        "type": "string"
      }
    },
    "dependencies": {
      "steps": {
        "allOf": [
          {
            "not": { "required": ["scenario"] }
          },
          {
            "not": { "required": ["definition"] }
          }
        ]
      },
      "scenario": {
        "allOf": [
          {
            "not": { "required": ["steps"] }
          },
          {
            "not": { "required": ["definition"] }
          }
        ]
      },
      "definition": {
        "allOf": [
          {
            "not": { "required": ["steps"] }
          },
          {
            "not": { "required": ["scenario"] }
          }
        ]
      }
    },
    "required": ["summary", "projectKey", "testType"],
    "additionalProperties": false
  },

  "EvidenceItem": {
    "type": "object",
    "properties": {
      "data": {
        "type": "string"
      },
```

```
        "filename": {
          "type": "string"
        },
        "contentType": {
          "type": "string"
        }
      },
      "required": ["data", "filename"],
      "additionalProperties": false
    },

    "CustomField": {
      "type": "array",
      "items": {
        "type": "object",
        "properties": {
          "id": {
            "type": "string"
          },
          "value": {}
        },
        "required": ["id", "value"],
        "additionalProperties": false
      }
    }
  }
}
```

Example 1: Importing gherkin and other test results

In this example, we are importing execution results for three existing test issues in Jira. The last issue DEMO-9 must be a BDD Test with a Gherkin definition because the results contain examples. The remaining issues can be of any test type.

```
{
    "tests" : [
        {
            "testKey" : "DEMO-7",
            "start" : "2013-05-03T11:47:35+01:00",
            "finish" : "2013-05-03T11:50:56+01:00",
            "comment" : "Test was OK but the performance is very poor",
            "status" : "PASS"
        },
        {
            "testKey" : "DEMO-8",
            "start" : "2013-05-03T12:14:12+01:00",
            "finish" : "2013-05-03T12:15:23+01:00",
            "comment" : "Performance was better this time, in the context of test set DEMO-10.",
            "status" : "PASS"
        },
        {
            "testKey" : "DEMO-9",
            "start" : "2013-05-03T12:19:23+01:00",
            "finish" : "2013-05-03T12:20:01+01:00",
            "comment" : "Error decreasing space shuttle speed.",
            "status" : "FAIL",
                        "examples" : [
                                "PASS",
                                "PASS",
                                "PASS",
                                "PASS",
                                "PASS",
                                "FAIL"
                        ]
        }
    ]
}
```

Example 2: Generic Test

This is a simple example of a JSON file with execution results for a generic test.

```
{
    "tests" : [
        {
            "testKey" : "ABC-129",
            "start" : "2014-08-30T11:47:35+01:00",
            "finish" : "2014-08-30T11:50:56+01:00",
            "comment" : "Successful execution",
            "status" : "PASS"
        }
    ]
}
```

Example 3: Importing manual test results with steps

This is a simple example of a JSON file with execution results for a manual test.

```
{
    "tests" : [
        {
            "testKey" : "DEMO-57",
            "start" : "2014-08-30T12:19:23+01:00",
            "finish" : "2014-08-30T12:20:01+01:00",
            "comment" : "Error executing step 2!",
            "status" : "FAILED",
            "steps": [
                {
                    "status": "PASSED",
                    "actualResult": "Step 1: OK"
                },
                {
                    "status": "FAILED",
                    "actualResult": "Step 2 *Failed* with an unexpected error message",
                    "evidences" : [
                        {
                            "data": "(... base 64 encoded ...)",
                            "filename": "screenshot1.jpg",
                            "contentType": "image/jpeg"
                        }
                    ]
                }
            ]
        }
    ]
}
```

Example 4: Importing data-driven manual test results with auto-provisioning of tests

This is an example of a JSON file with a single test result.

This is a data-driven manual test with two iterations. For each iteration, we provide the parameters and the step results.

Xray will also create or update the test in Jira with the specification contained on the **"testInfo"** object.

```
{
    "tests": [
        {
            "start" : "2021-08-30T11:47:35+01:00",
            "finish" : "2021-08-30T11:50:56+01:00",
            "comment" : "Successful execution",
            "status" : "PASSED",
            "evidence" : [
                {
                    "data":
"iVBORw0KGgoAAAANSUhEUgAABkIAAAO9CAYAAADezXv6AAAAAXNSR0IArs4c6QAAAARnQU1BAACxjwv8YQUAAAAJcEhZcwAAEn(...base64
```

file enconding)",
                    "filename": "image21.jpg",
                    "contentType": "image/jpeg"
                }
            ],
            "testInfo": {
                "summary": "Strong password validation",
                "type": "Manual",
                "projectKey": "STORE",
                "steps": [
                    {
                        "action": "Open the Change Password screen by selecting option \"My Profile > Password\"",
                        "data": "",
                        "result": ""
                    },
                    {
                        "action": "Fill the password fields with data",
                        "data": "Current Password: ${Password}\nNew Password: ${Password}\nConfirm New Password: ${Password}",
                        "result": "The new password is: ${Valid}\nError:\n${Message}"
                    }
                ]
            },
            "iterations": [
              {
                "parameters": [
                    {
                        "name": "Password",
                        "value": "2635ftvu23v7t!09"
                    },
                    {
                        "name": "Valid",
                        "value": "Valid"
                    },
                    {
                        "name": "Message",
                        "value": ""
                    }
                ],
                "status": "PASSED",
                "steps": [
                    {
                        "actualResult": "",
                        "status": "PASSED"
                    },
                    {
                        "actualResult": "Password changed successfully",
                        "status": "PASSED"
                    }
                ]
              },
              {
                "parameters": [
                    {
                        "name": "Password",
                        "value": "123123"
                    },
                    {
                        "name": "Valid",
                        "value": "Not Valid"
                    },
                    {
                        "name": "Message",
                        "value": "Password is too simple."
                    }
                ],
                "status": "FAILED",
                "steps": [
                    {
                        "actualResult": "",

```
                        "status": "PASSED"
                    },
                    {
                        "actualResult": "Password too simple!",
                        "status": "FAILED"
                    }
                ]
            }
        ]
    }
    ]
}
```

Importing Multiple Execution results

Xray supports importing multiple results for the same Test issues in the same execution. These results often indicate different contexts/environments where the same Test must be executed. Xray will group all executions of the same Test in a single Test Run and present all execution information, including the different contexts in the Execution page.

When importing execution results using the Cucumber JSON output formatter, multiple executions for the same scenario/Test issue will be merged into a single Test Run.

When importing execution results using the Xray JSON format, the **results** JSON element must contain all execution results for the same Test Run.

Xamarin Test Cloud

If you are using the Xamarin Test Cloud for executing **Cucumber** mobile Tests in different combinations of mobile devices and operating systems, you can import the results to Jira by making a compressed zip file containing the multiple Cucumber JSON files.

Learn more

Please see Testing using Calabash and Xamarin Test Cloud in Ruby and the "bundle" endpoint in the REST API.