Using Xray JSON format to import execution results

Overview

Xray provides a proprietary JSON format for importing execution results into Jira/Xray.

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:

- 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

JSON format

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 userid 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

"test" 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. Sub-sequent 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
type	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)
scenario	The BDD scenario
definition	The generic test definition

"step def" object - step definition

This object allows you to define the step fields for manual tests. Custom test step fields are also supported.

action	The step action - native field
data	The step data - native field
result	The step expected result - native field
(...)	Any other step custom fields

"iteration" object - Data-driven test results

If you need to import data-driven test results you need to use the iteration 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)
log	The log for the iteration
duration	A duration for the iteration
status	The status for the iteration (PASSED, FAILED, EXECUTING, TODO, custom statuses ...)
steps	An array of step results (for Manual tests) (link)

"parameter" object - parameters within iteration results

name	The parameter name
value	The parameter value

"step result" object - step results

status	The status for the test step (PASSED, FAILED, EXECUTING, TODO, custom statuses ...)
comment	The comment for the step result
actualResult	The actual result field for the step result
evidence	An array of evidence items of the test run (link)
defects	An array of defect issue keys to associate with the test run

"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

"customField" 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"** or **"name"** to find the existing test run custom field in the project settings.

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

Xray JSON Schema

The JSON results file must comply to 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"]
        }
      },
      "results": { // DEPRECATED
        "type": "array",
        "items": {
          "$ref": "#/definitions/IterationResult"
        }
      },
      "iterations": {
        "type": "array",
        "items": {
          "$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": ["results"] }
        },
        {
          "not": { "required": ["iterations"] }
        }
      ]
    },
    "examples": {
      "allOf": [
        {
          "not": { "required": ["steps"] }
        },
        {
          "not": { "required": ["results"] }
        },
        {
          "not": { "required": ["iterations"] }
        }
      ]
    },
    "results": {
      "allOf": [
        {
          "not": { "required": ["steps"] }
        },
        {
          "not": { "required": ["examples"] }
        },
        {
          "not": { "required": ["iterations"] }
        }
      ]
    },
    "iterations": {
      "allOf": [
        {
          "not": { "required": ["steps"] }
        },
        {
          "not": { "required": ["examples"] }
        },
        {
          "not": { "required": ["results"] }
        }
      ]
    }
  },
  "additionalProperties": false
},
```

```
"IterationResult": {
  "type": "object",
  "properties": {
    "name": {
      "type": "string"
    },
    "parameters": {
      "type": "array",
      "items": {
        "type": "object",
        "properties": {
          "name": {
            "type": "string"
          },
          "value": {
            "type": "string"
          }
        },
        "required": ["name"],
        "additionalProperties": false
      }
    },
    "log": {
      "type": "string"
    },
    "duration": {
      "type": "string"
    },
    "status": {
      "type": "string"
    },
    "steps": {
      "type": "array",
      "items": {
        "$ref": "#/definitions/ManualTestStepResult"
      }
    }
  },
  "required": ["status"],
  "additionalProperties": false
},

"ManualTestStepResult": {
  "type": "object",
  "properties": {
    "status": {
      "type": "string"
    },
    "comment": {
      "type": "string"
    },
    "evidence": {
      "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"
    },
    "projectKey": {
      "type": "string"
    },
    "requirementKeys":  {
      "type": "array",
      "items": {
        "type": "string"
      }
    },
    "labels": {
      "type": "array",
      "items": {
        "type": "string"
      }
    },
    "type": {
      "type": "string"
    },
    steps: {
      type: "array",
      items: {
              type: "object",
                    properties: {
                      action: {
            type: "string"
          },
          data: {
            type: "string"
          },
          result: {
            type: "string"
          }
        },
        // custom fields
        patternProperties: {
          ".+": {}
        },
        required: ["action"],
              additionalProperties: false
      }
    },
    "scenario": {
      "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", "type"],
    "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"
        },
        "name": {
          "type": "string"
        },
        "value": {}
      },
      "anyOf": [
        {
          "required": ["id", "value"]
        },
        {
          "required": ["name", "value"]
        }
      ],
      "additionalProperties": false
    }
  }
}
```

Examples

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" : "PASSED"
        },
        {
            "testKey" : "DEMO-8",
            "start" : "2013-05-03T12:14:12+01:00",
            "finish" : "2013-05-03T12:15:23+01:00",
            "status" : "PASSED"
        },
        {
            "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" : "FAILED",
                        "examples" : [
                                "PASSED",
                                "PASSED",
                                "PASSED",
                                "PASSED",
                                "PASSED",
                                "FAILED"
                        ]
        }
    ]
}
```

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"
                        }
                    ]
                }
                        ]
        }
    ]
}
```

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": [
              {
                "name": "Iteration 1",
                "parameters": [
                    {
                        "name": "Password",
                        "value": "2635ftvu23v7t!09"
                    },
                    {
                        "name": "Valid",
                        "value": "Valid"
                    },
                    {
                        "name": "Message",
                        "value": ""
                    }
                ],
                "log": "Password changed successfully",
                "status": "PASSED",
                "steps": [
                    {
                        "actualResult": "",
                        "status": "PASSED"
                    },
                    {
                        "actualResult": "Password changed successfully",
                        "status": "PASSED"
                    }
                ]
              },
              {
                "name": "Iteration 2",
                "parameters": [
                    {
                        "name": "Password",
                        "value": "123123"
                    },
                    {
                        "name": "Valid",
                        "value": "Not Valid"
                    },
                    {
                        "name": "Message",
                        "value": "Password is too simple."
                    }
                ],
                "log": "Password validation check failed. Password too simple!",
                "status": "FAILED",
                "steps": [
                    {
                        "actualResult": "",
                        "status": "PASSED"
                    },
                    {
                        "actualResult": "Password too simple!",
                        "status": "FAILED"
                    }
                ]
              }
            ]
        }
    ]
}
```