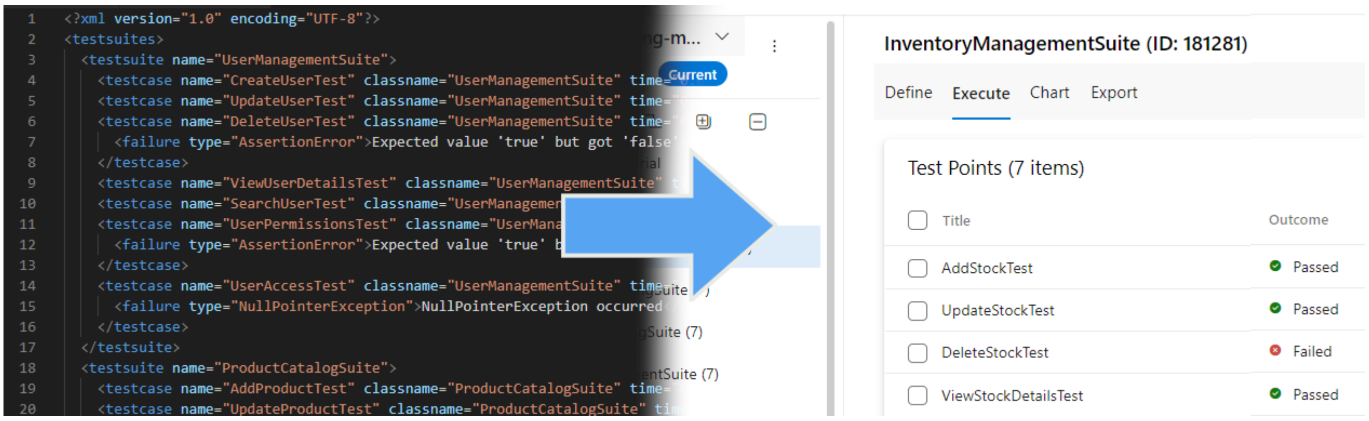


```
<?xml version="1.0" encoding="UTF-8"?>
<testsuites>
  <testsuite name="UserManagementSuite">
    <testcase name="CreateUserTest" classname="UserManagementSuite" time=
    <testcase name="UpdateUserTest" classname="UserManagementSuite" time=
    <testcase name="DeleteUserTest" classname="UserManagementSuite" time=
      <failure type="AssertionError">Expected value 'true' but got 'false
    </testcase>
    <testcase name="ViewUserDetailsTest" classname="UserManagementSuite"
    <testcase name="SearchUserTest" classname="UserManagemen
    <testcase name="UserPermissionsTest" classname="UserMana
      <failure type="AssertionError">Expected value 'true' b
    </testcase>
    <testcase name="UserAccessTest" classname="UserManagementSuite" tim
      <failure type="NullPointerException">NullPointerException occurred
    </testcase>
  </testsuite>
  <testsuite name="ProductCatalogSuite">
    <testcase name="AddProductTest" classname="ProductCatalogSuite" time=
    <testcase name="UpdateProductTest" classname="ProductCatalogSuite" ti
```

InventoryManagementSuite (ID: 181281)

Test Points (7 items)

Title	Outcome
AddStockTest	Passed
UpdateStockTest	Passed
DeleteStockTest	Failed
ViewStockDetailsTest	Passed

Get the big picture of your product quality in Azure DevOps

Are you working with different test tools and technologies? Do you struggle with getting the big picture of your product quality?

We've got you covered!

Solidify's Test Result Importer will get all of your manual and automatic test results in one place - Azure Test Plans.

As long as your test tool of choice supports reporting in the JUnit/xUnit, TestNG or Gherkin formats, it can be imported to Azure Test Plans using this tool.

Key benefits

- Combine automatic and manual test results into one source for reporting and visualizing of product quality.
- Convert the test data into test suites and test cases in a way that preserves the data hierarchy.
- Automatic generation of test cases and test steps in Azure DevOps.
- Support for both yaml pipelines and classic build and releases.

Easy and secure

Everything created with Test Result Importer stays in your Azure DevOps, which means you don't have to worry about integration, security or data protection in a separate service.

Up and running in 5 minutes:

- Install the extension in your Azure DevOps
- Add the Test Result Importer activity to your pipelines
- Find both your manual and automatic test results in the same source - Azure test Plans
- Create reports from your combined manual and automatic test effort, or visualize the quailty using Azure Dashboards.

Try Solidify's Test Result Importer now

Still not sure? Try our **free 30-day trail**. Click the "Get It Now" button and get started!

Features

The **Solidify Test Result Importer extension** provides a build task that can convert test results files from a variety of test frameworks into an Azure Testplan.

The extension currently supports the following file formats:

- Gherkin (.json)
- JUnit/xUnit (.xml)
- TestNG (.xml)

Additional features:

- Convert the test data into test suites and test cases in a way that preserves the data hierarchy.
- Choose wether to create new test cases or overwrite the existing ones.
- Choose which target project in which to create the test data.
- Supports test steps.
- Supports test parameters (a new test iteration will be created for every parameter set).
- Supports custom work item fields (specify in the task).
- Supports import of additional test steps from a .json file.
- Supports import of files and links as Attachments on the Test Runs or Test Results.

Supported test frameworks

Test Result Importer has support for all test automation frameworks that integrate with the **JUnit**, **TestNG** or **Gherkin** formats, including the following:

- Selenium
- Cypress
- Appium
- NUnit
- Cucumber
- Robot Framework
- TestComplete
- Jasmine
- Karma
- Protractor
- Gauge
- Karate
- PyTest
- Spring Boot
- SoapUI
- JMeter
- WebdriverIO
- XCUITest
- Espresso
- Gatling
- And many more...

Installation and configuration

Installation

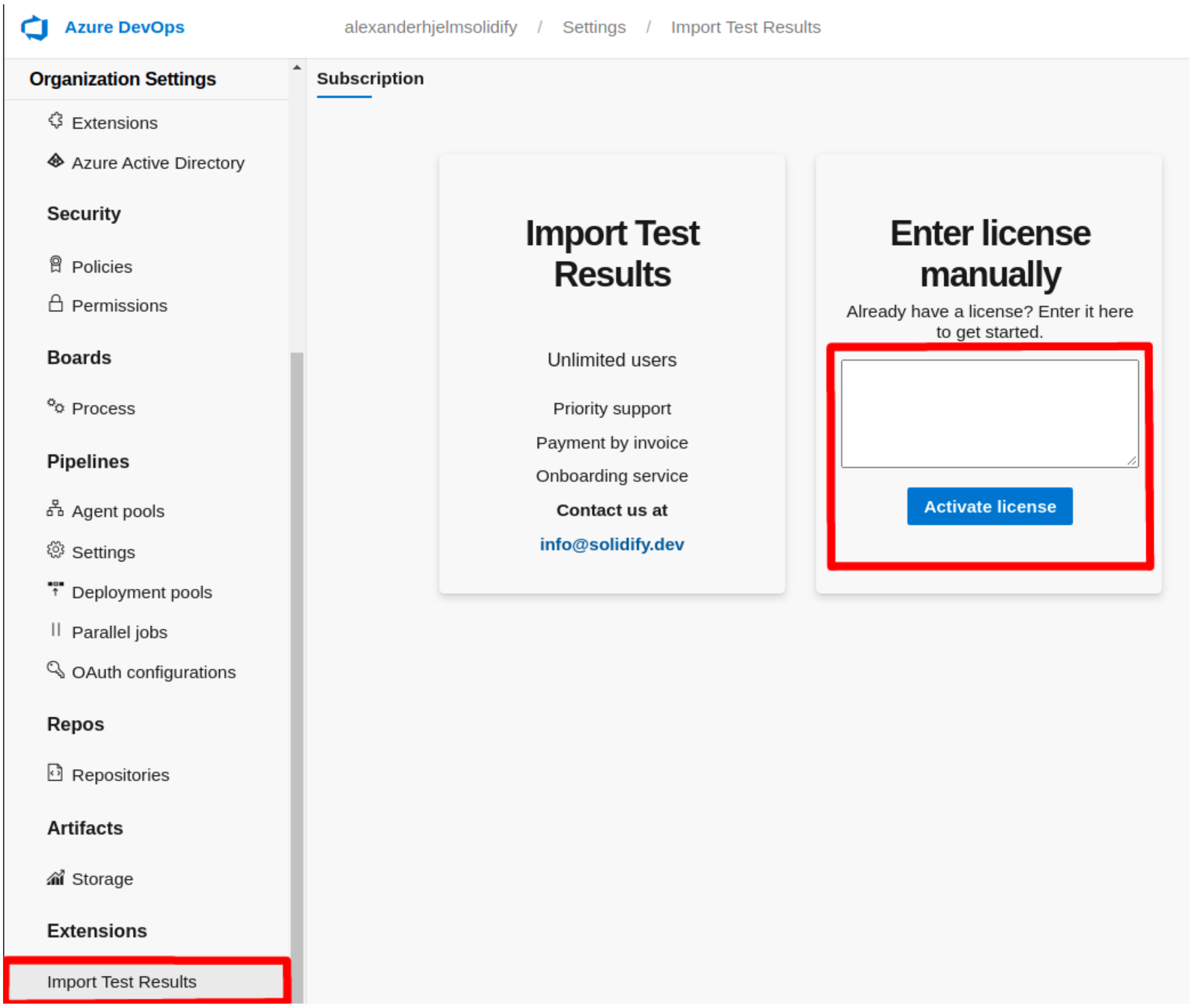
Install the Test Result Importer to your Azure DevOps Organization.

Configure the license

Before you get started, you must import import your .json license file into the extension's storage. If you have been in contact with us before, chances are you have already been given a license file. Otherwise contact us at support.tri@solidify.dev to inquire about purchasing a license.

The validity period of the license is 12 months, after which you must request a new license from us.

You will need to open the configuration page via *Organization Settings* -> *Extensions* -> *Test Result Importer* -> *Enter License Manually*. Paste the contents of the json file in its entirety.



Pipelines or Releases?

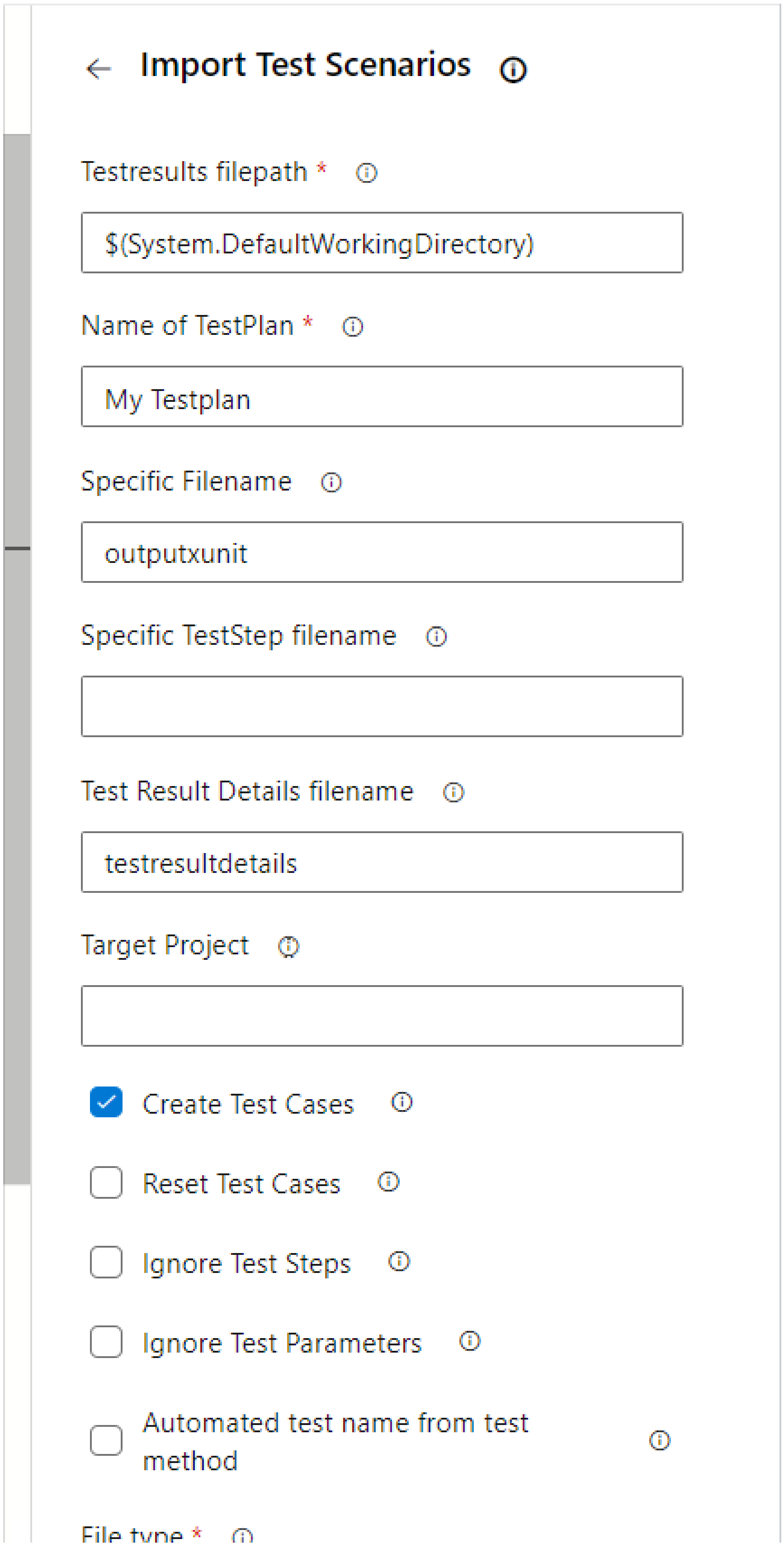
Add the Test Result Importer as a task in either Pipelines or in Releases, both classic task and yaml task are supported.

Configure the task

Required configuration

- **Testresults filepath**: The root path to your testresults file, default: "$(System.DefaultWorkingDirectory)"
- **Specific Filename**: The path to your test results file (.xml/.json)
- **Name of TestPlan**: The name of the test plan to write the data to
- **The file type** (JUnit/Gherkin/TestNG)

This is all the configuration required! However, you can see that there are a lot more options to try out:



Import Test Scenarios

Testresults filepath *

$(System.DefaultWorkingDirectory)

Name of TestPlan *

My Testplan

Specific Filename

outputxunit

Specific TestStep filename

Test Result Details filename

testresultdetails

Target Project

[✓] Create Test Cases

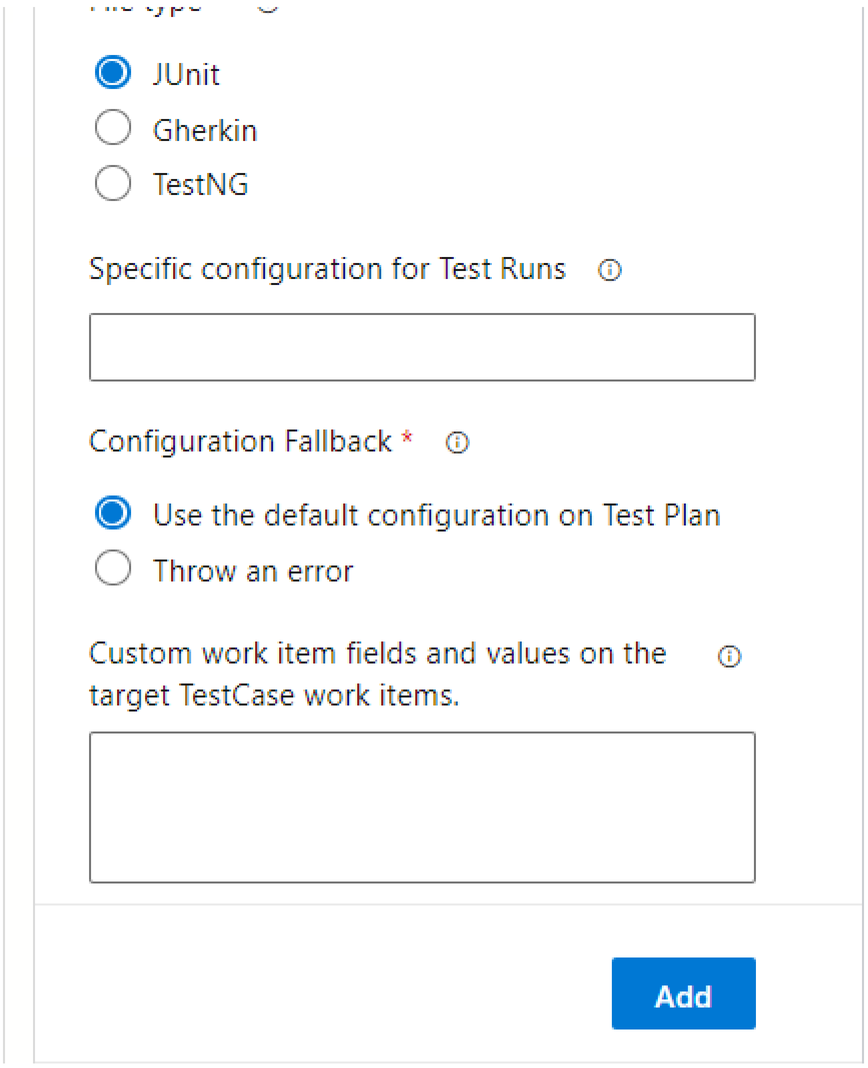
[ ] Reset Test Cases

[ ] Ignore Test Steps

[ ] Ignore Test Parameters

[ ] Automated test name from test method

File type *


File type

[✓ JUnit]
Gherkin
TestNG

Specific configuration for Test Runs

Configuration Fallback *

[✓ Use the default configuration on Test Plan]
Throw an error

Custom work item fields and values on the target TestCase work items.

Optional configuration

- **Target Project**: The project where the chosen Test Plan resides (leave empty to use the same project as the pipeline)
- **Create Test Cases**: Check this if you want to automatically create Test Cases in the selected Test Plan in Azure DevOps
- **Reset Test Cases**: Check this if you want to reset all Test Cases in the chosen Test Plan
- **Ignore Test Steps**: Check this if you want to skip writing the Test Case test steps
- **Ignore Test Parameters**: Check this if you want to skip writing the Test Case parameters
- **File type**: Which type of file to parse (JUnit/Gherkin/TestNG)
- **Specific TestStep filename**: The path to the .json file containing the extra test steps.
- **Test Result Details filename**: The name of the .json that contains the testresult details, including attachments, URLs, work item links and test case summaries.
- **Custom work item fields and values on the target TestCase work items**: Specify custom work item fields to add on every created test case. One field/value-pair per line. example:

```
System.Title=My title!
System.Description=My description!
```

Build Service Account, required permissions

The Build Service Account will be either one fo the following depending on your project configuration:

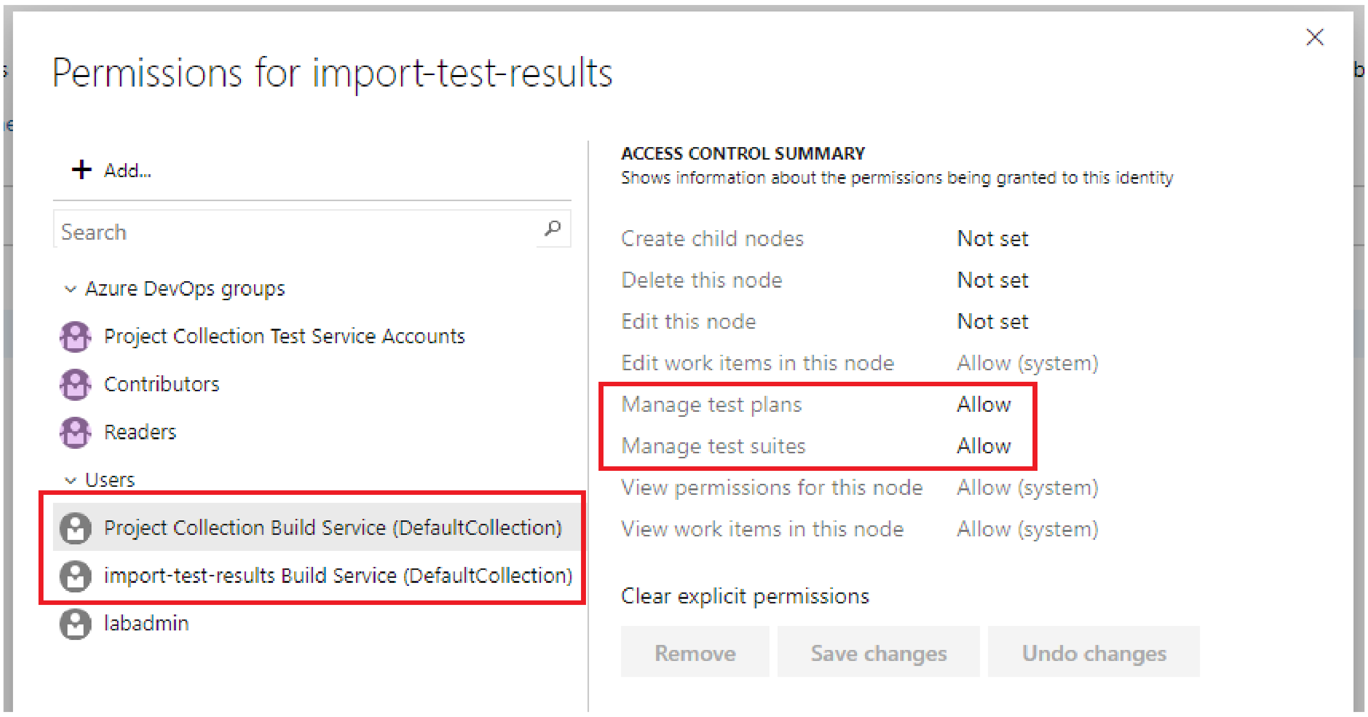
- `Project Collection Build Service ([Collection name])`
- `[Project name] Build Service ([Collection name])`

The Build Service Account will require the following permissions under **Project Settings** -> **Permissions**:

- Create Test Runs
- Delete Test Runs
- Manage Test Plans

Additionally, you must configure the permissions of the **area path** where the Test Plan is located. Under **Project Settings** -> **Project configuration** -> **Areas** -> **[Select the area path of the Test Plan]** -> **Security**: You must add the Build Service Account and **Allow** the following permissions:

- Manage test plans
- Manage test suites



Permissions for import-test-results

+ Add...
Search
Azure DevOps groups
Project Collection Test Service Accounts
Contributors
Readers
Users
Project Collection Build Service (DefaultCollection)
import-test-results Build Service (DefaultCollection)
labadmin

ACCESS CONTROL SUMMARY
Shows information about the permissions being granted to this identity

Create child nodes	Not set
Delete this node	Not set
Edit this node	Not set
Edit work items in this node	Allow (system)
Manage test plans	Allow
Manage test suites	Allow
View permissions for this node	Allow (system)
View work items in this node	Allow (system)

If the area permissions are configured incorrectly, you will see the following error message in the pipeline log:

```
Task failed, error message: Error: You do not have the appropriate permissions to
manage test suites under this area path.: import-test-results
```

Import Extra test steps from .json file

In order to import extra test steps into your cases, you must first specify the **Specific TestStep filename** task parameter. Specify the path to your .json file in the repository which contains the extra test steps.

The extra test steps file must follow this schema:

```
{
    "testcases": [
        {
            "id": <number>,
            "name": <string>,
            "teststeps": [
                {
                    "id": <number>,
                    "name": <string>
                }
            ]
        }
    ]
}
```

Here:

- `testcases` contains references to the test cases for which we want to add test steps.
    - `id` must be a unique positive integer (e.g. 1, 2, 3, 4...).
    - `name` must match the name of the TestCase work item.
    - `teststeps` contains the test steps that we want to add to the current TestCase.
        - `id` is an integer that defines the order of the test steps (1, 2, 3, 4...).
        - `name` is the text of the test step that will appear in the work item.

Here is a complete example of the **test steps .json** file:

```
{
    "testcases": [
        {
            "id": 1,
            "name": "Verify number of frames",
            "teststeps": [
                {
                    "id": 1,
                    "name": "Test Setup"
                },
                {
                    "id": 2,
                    "name": "Create session"
                },
                {
                    "id": 3,
                    "name": "TEST = Get number of received frames from session"
                },
                {
                    "id": 4,
                    "name": "Test Teardown"
                }
            ]
        },
        {
            "id": 2,
            "name": "Verify can receive messages",
            "teststeps": [
                {
                    "id": 1,
                    "name": "Test Setup"
                },
                {
                    "id": 2,
                    "name": "Create session"
                },
                {
                    "id": 3,
                    "name": "Verify that session can receive messages"
                },
                {
                    "id": 4,
                    "name": "Test Teardown"
                }
            ]
        }
    ]
}
```

Edit details for specific test case/test run result

Additional details about the individual test cases/test run results can be supplied in the file `testresultdetails.json` in the working directory (the git repository root). If no such file is found in the repo, this step will be skipped.

Using the `testresultdetails.json` file, you can:

- Import files or links as Test Run/Test Result attachments
- Specify individual work item links (useful to track which requirements are verified by what test cases)
- Specify test case descriptions (summaries).

`testresultdetails.json` must follow this schema:

```
    {
        "testrun": {
            "urls": [
                {
                    "id": <number>,
                    "name": <string>,
                    "urlstring": <urlstring>
                }
            ],
            "attachments": [
                {
                    "id": <number>,
                    "name": <string>,
                    "filename": <urlstring>
                }
            ]
        },
        "testcases": [
            {
                "id": <number>,
                "name": <string>,
                "summary": <string>,
                "workItemLinks": [
                    {
                        "id": <number>,
                        "linkType": <string>
                    }
                ]
                "urls": [
                    {
                        "id": <number>,
                        "name": <string>,
                        "urlstring": <urlstring>
                    }
                ],
                "attachments": [
                    {
                        "id": <number>,
                        "name": <string>,
                        "filename": <urlstring>
                    }
                ]
            }
        ]
    }
```

Here:

- `testrun` contains the urls and attachments to be added to the Test Run.
- `testcases` contains the **urls**, **attachments**, **work item links** and **descriptions** to be added to the individual test case results. The `id` property must be a unique positive integer (e.g. 1, 2, 3, 4...). The `name` must match the test case name.

Here is a complete example of the `testresultdetails.json` file:

```

{
    "testrun": {
        "urls": [
            { "id": 1, "name": "aftonbladet", "urlstring":
"http://www.aftonbladet.se" },
            { "id": 2, "name": "expressen", "urlstring": "http://www.expressen.se"
}
        ],
        "attachments": []
    },
    "testcases": [
        {
            "id": 1,
            "name": "Verify user can add THU Line",
            "summary": "This is my summary!",
            "workItemLinks": [
                {
                    "id": 57841,
                    "linkType": "Hierarchy-Forward"
                }
            ],
            "urls": [
                { "id": 1, "name": "aftonbladet", "urlstring":
"http://www.aftonbladet.se" },
                { "id": 2, "name": "expressen", "urlstring":
"http://www.expressen.se" }
            ],
            "attachments": [
                { "id": 1, "name": "file_a.md", "filename": "testdata\\file_a.md"
},
                { "id": 2, "name": "file_b.md", "filename": "testdata\\file_b.md"
}
            ]
        },
        {
            "id": 2,
            "name": "Verify added Pickup Reference is available in confirmed TBR",
            "urls": [
                { "id": 1, "name": "aftonbladet","urlstring":
"http://www.aftonbladet.se" },
                { "id": 2, "name": "aftonbladet","urlstring":
"http://www.expressen.se" }
            ],
            "attachments": [
                { "id": 1, "name": "file_a.md", "filename": "testdata\\file_a.md"
},
                { "id": 2, "name": "file_b.md", "filename": "testdata\\file_b.md"
}
            ]
        }
    ]
}
```

Set AutomatedTest properties on the Test Cases

Add the following lines to the **Custom work item fields and values on the target TestCase work items** property:

```
Microsoft.VSTS.TCM.AutomatedTestStorage=myTestStorageFile.dll
Microsoft.VSTS.TCM.AutomatedTestId=527a1270-8d28-41d3-ba70-933a59c63ab1
Microsoft.VSTS.TCM.AutomatedTestType=Unit Test
Microsoft.VSTS.TCM.AutomationStatus=Automated
```

Enable access to the OAuth token

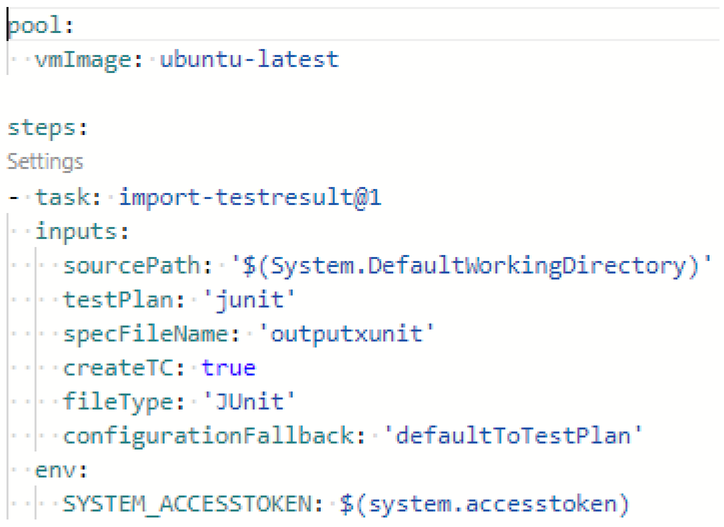
For the pipeline to run, you must give it access to the Oauth token, otherwise you will receive errors like the following.

```
SYSTEM_ACCESSTOKEN env var not set
```

Simply go to: **Edit Pipeline** -> **Agent Job 1** -> **Additional Options**, and ensure the "Allow scripts to access the OAuth token" checkbox is ticked. Then save the pipeline and run again.

Configuration example: yaml

Here is a screenshot detailing how to configure the task in yaml:



```yaml
pool:
  vmImage: ubuntu-latest

steps:
Settings
- task: import-testresult@1
  inputs:
    sourcePath: '$(System.DefaultWorkingDirectory)'
    testPlan: 'junit'
    specFileName: 'outputxunit'
    createTC: true
    fileType: 'JUnit'
    configurationFallback: 'defaultToTestPlan'
  env:
    SYSTEM_ACCESSTOKEN: $(system.accesstoken)
```

Configuration: Overriding Automated Test Case name

For JUnit and TestNG, the Automated Test Case property on the Testcase work item is inferred automatically from the testcase/class name. If you wish to override it, or also set it for Gherkin, the following must go into the **Custom work item fields and values on the target TestCase work items**-textbox at the bottom of the config:

```
Microsoft.VSTS.TCM.AutomatedTestName=ExampleTestName
Microsoft.VSTS.TCM.AutomatedTestStorage=example.dll
Microsoft.VSTS.TCM.AutomatedTestId=527a1270-8d28-41d3-ba70-933a59c63ab1
Microsoft.VSTS.TCM.AutomatedTestType=Unit Test
Microsoft.VSTS.TCM.AutomationStatus=Automated
```

Replace "ExampleTestName" with whatever you want the AutomatedTestName field to say.

Do you have questions, issues or a feature request?

Please get in touch using the Q&A section or our contact form: https://solidify.dev/contact