pure::variants Model Test Framework Manual

pure-systems GmbH

Version 6.0.5.685 for pure::variants 6.0

Copyright © 2003-2024 pure-systems GmbH

2024

Table of Contents

1. Introduction 1
1.1. Software Requirements 1
1.2. Installation 1
1.3. About this manual 1
2. Using 2
2.1. Creating a PVUnit Test Case 2
2.1.1. Create new empty PVUnit Test Case 2
2.1.2. Create new PVUnit Test Case 3
2.2. Modifying a Test Case 5
2.2.1. General Tab 5
2.2.2. Modifications Tab 6
2.2.3. Validations Tab 7
2.2.4. PVSCL Tab 8
2.2.5. Transformation Tab 9
2.2.6. Navigation 10
2.3. Running Test Cases 11
2.3.1. Running Test Cases from Projects View 11
2.3.2. Running Test Cases from Outline View 12
2.3.3. Test Run Result View 13
3. External Build Support (Ant Tasks) 13
3.1. pv.pvunittest 13
4. Known restrictions 14

1. Introduction

The pure::variants Model Test Framework enables the pure::variants user to easily test pure::variants variant description models.

1.1. Software Requirements

The pure::variants Model Test Framework is an extension for pure::variants and is available on all supported platforms. There are no further software requirements.

1.2. Installation

Please consult section **pure::variants Connectors** in the **pure::variants Setup Guide** for detailed information on how to install the connector (menu **Help** -> **Help Contents** and then **pure::variants Setup Guide** -> **pure::variants Connectors**).

1.3. About this manual

The reader is expected to have basic knowledge about and experiences with pure::variants and software testing. The pure::variants manual is available in online help as well as in printable PDF format here.

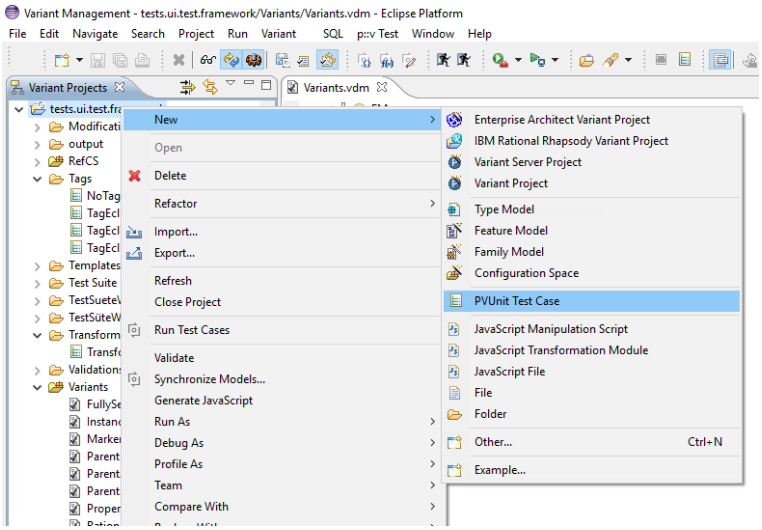
2. Using

2.1. Creating a PVUnit Test Case

2.1.1. Create new empty PVUnit Test Case

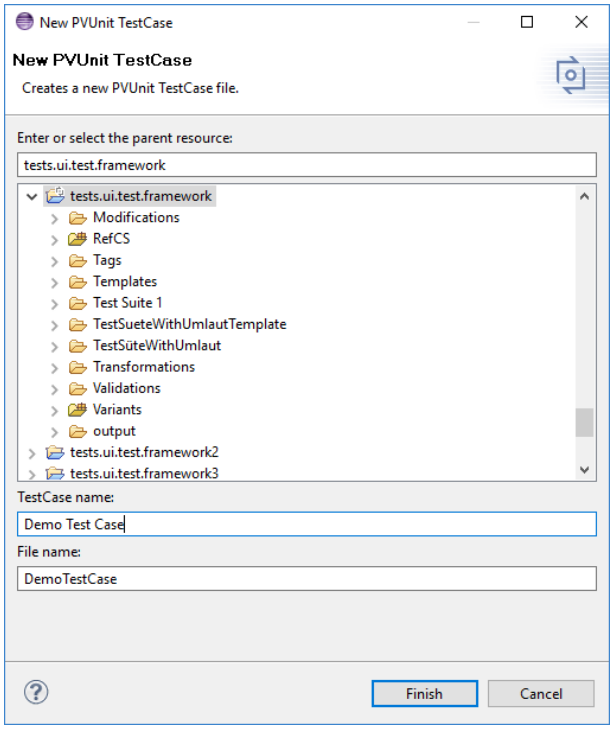
To create a new empty PVUnit Test Case the *PVUnit Test Case* entry from the *New* menu in the context menu of the *Variant Projects* view is used.

Figure 1. Create new empty PVUnit Test Case



This action opens a wizard, where the user specifies a name and a file name for the new test case. After finishing the wizard the new test case is opened in the PVUnit Test Case Editor. See Section 2.2, “Modifying a Test Case”

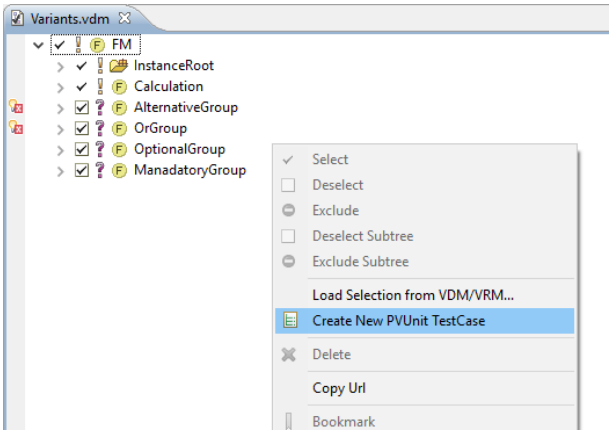
Figure 2. Create new empty PVUnit Test Case



2.1.2. Create new PVUnit Test Case

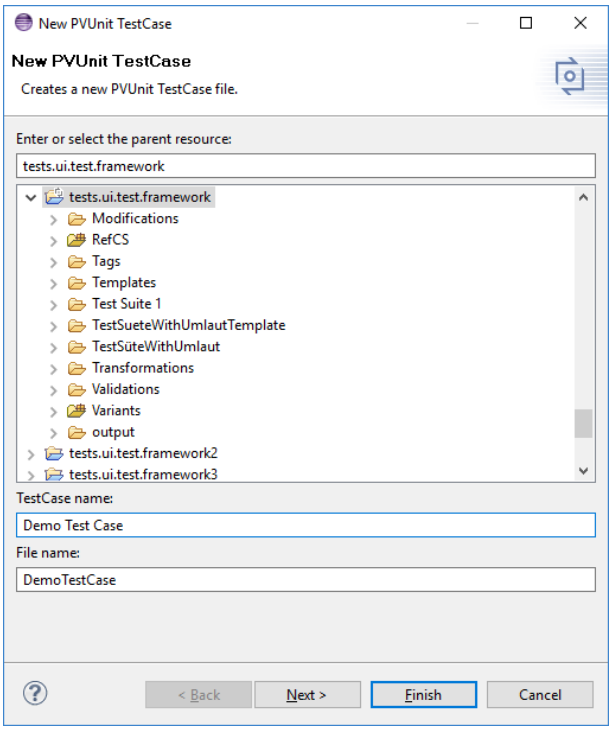
There is a second possibility to create a new Test Case. To create a new PVUnit Test Case the variant model under test is opened. In the variant model editor either the tool bar item or the *Create New PVUnit Test Case* entry in the context menu of the variant model editor is used.

Figure 3. Create new PVUnit Test Case



This action opens a wizard, which is similar to the wizard for the empty test case. On the first page the user specifies a name and a file name for the new test case.

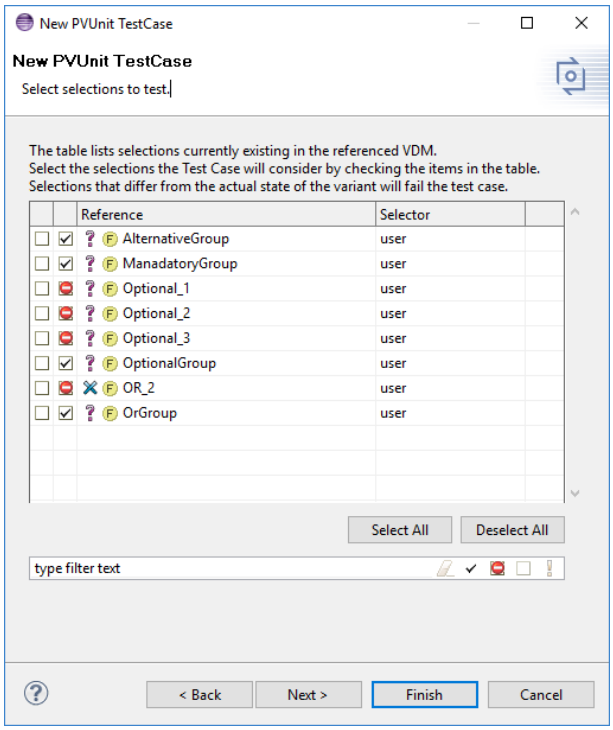
Figure 4. Create new PVUnit Test Case - Page 1



The second page enables the user to select selections, which shall be checked during test case execution. The page lists all selections from the variant model used to create this test case. By default deselected elements are filtered from the table along with mandatory elements, since in most cases these elements are not of concern. The default filter can be modified with the filter control below the table.

The user selects selections, which will be checked by the test by checking the box in the first column.

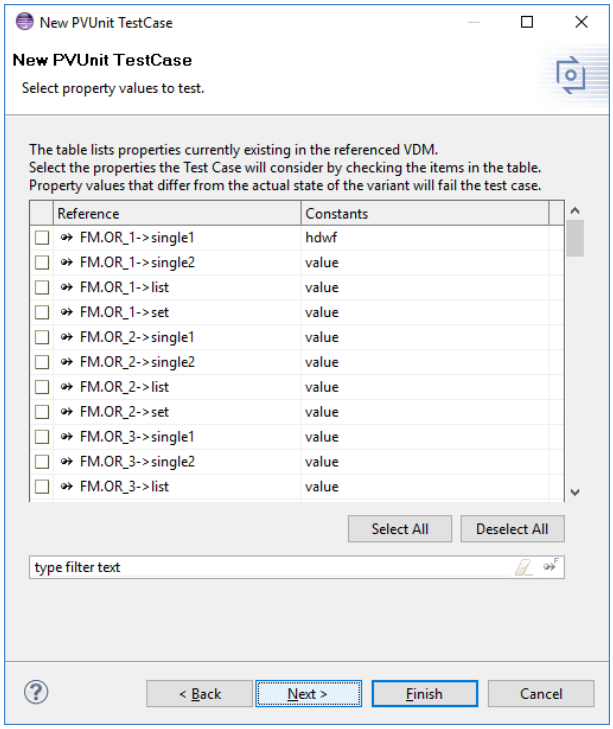
Figure 5. Create new PVUnit Test Case - Page 2



The third page lists all properties from the variant model. This includes fixed attributes. By default fixed attributes are filtered from the table, since in most cases these attributes are not of concern. The default filter can be modified with the filter control below the table.

The user selects attributes, which will be checked by the test by checking the box in the first column.

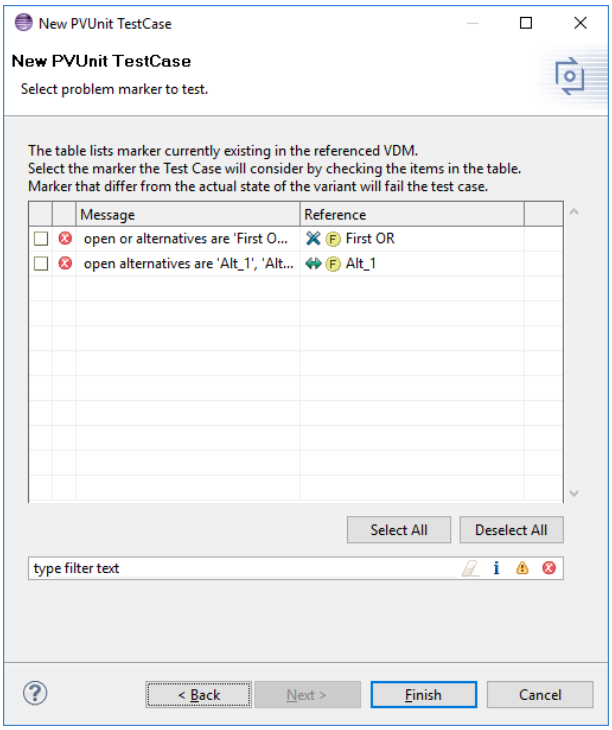
Figure 6. Create new PVUnit Test Case - Page 3



The fourth page lists all evaluation marker from the variant model. The filter can be modified with the filter control below the table. The user selects marker, which will be checked by the test by checking the box in the first column.

After finishing the wizard the new test case is opened in the PVUnit Test Case Editor. See Section 2.2, “Modifying a Test Case”

Figure 7. Create new PVUnit Test Case - Page 4



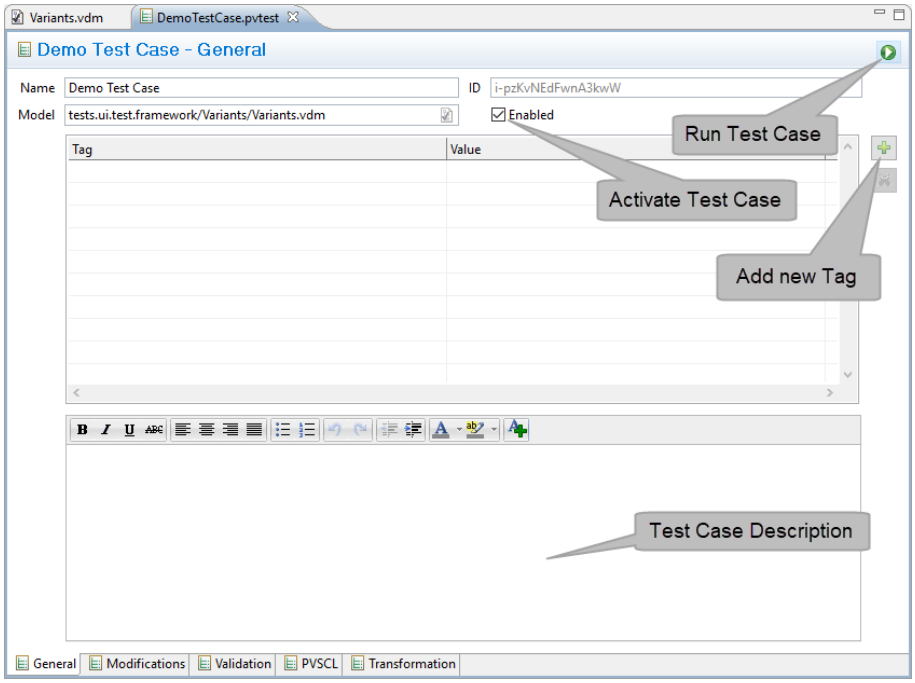
2.2. Modifying a Test Case

For modifying a test case the PVUnit Test Case Editor is provided. This editor consists of 5 tabs. Each tab allows the user to modify a part of the test data. The Editor aids the user by checking input data for consistency. If something is not as expected the editor shows the problems by adding marker to the controls with problems.

2.2.1. General Tab

The General tab enables the user to modify the general test data. On this tab the name of the test case can be modified, the variant model to test can be specified, a description for the test can be defined in the lower part and it is possible to define multiple tags. The *Enabled* checkbox defines, if a testcase is active. If the checkbox is deselected the test case will be ignored during execution.

Figure 8. General Tab



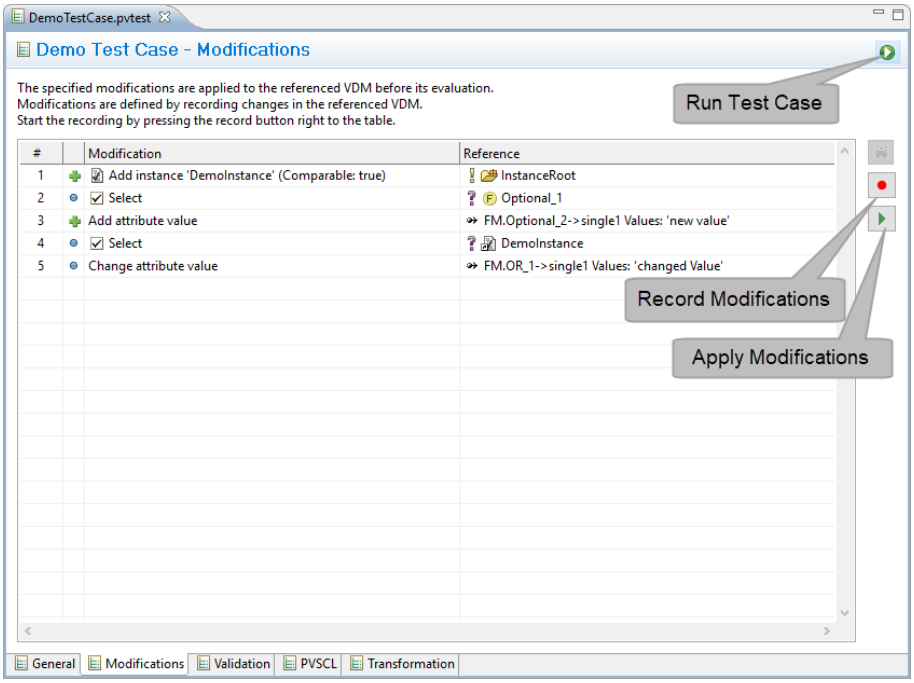
2.2.2. Modifications Tab

The modification tab lists all modifications, which will be performed before the variant model is evaluated. Possible modifications are:

- Performing selections, exclusions, deselections
- Adding, Changing, Removing attribute values
- Adding, Renaming, Cloning, Removing instances
- Adding, Removing inherited variant models
- Adding, Removing, Changing selection rationals

To define modifications the modification recorder is used. The recorder is started by the record button. With pressing the button the editor opens the referenced variant model in a variant model editor and applies all modifications from the table. The user now can perform all necessary modifications in the variant model editor. If the modifications are done, the user either returns to the PVUnit Test Case editor and uses the stop button or uses the stop button from the tool bar in the variant model editor.

Figure 9. Modifications Tab



2.2.3. Validations Tab

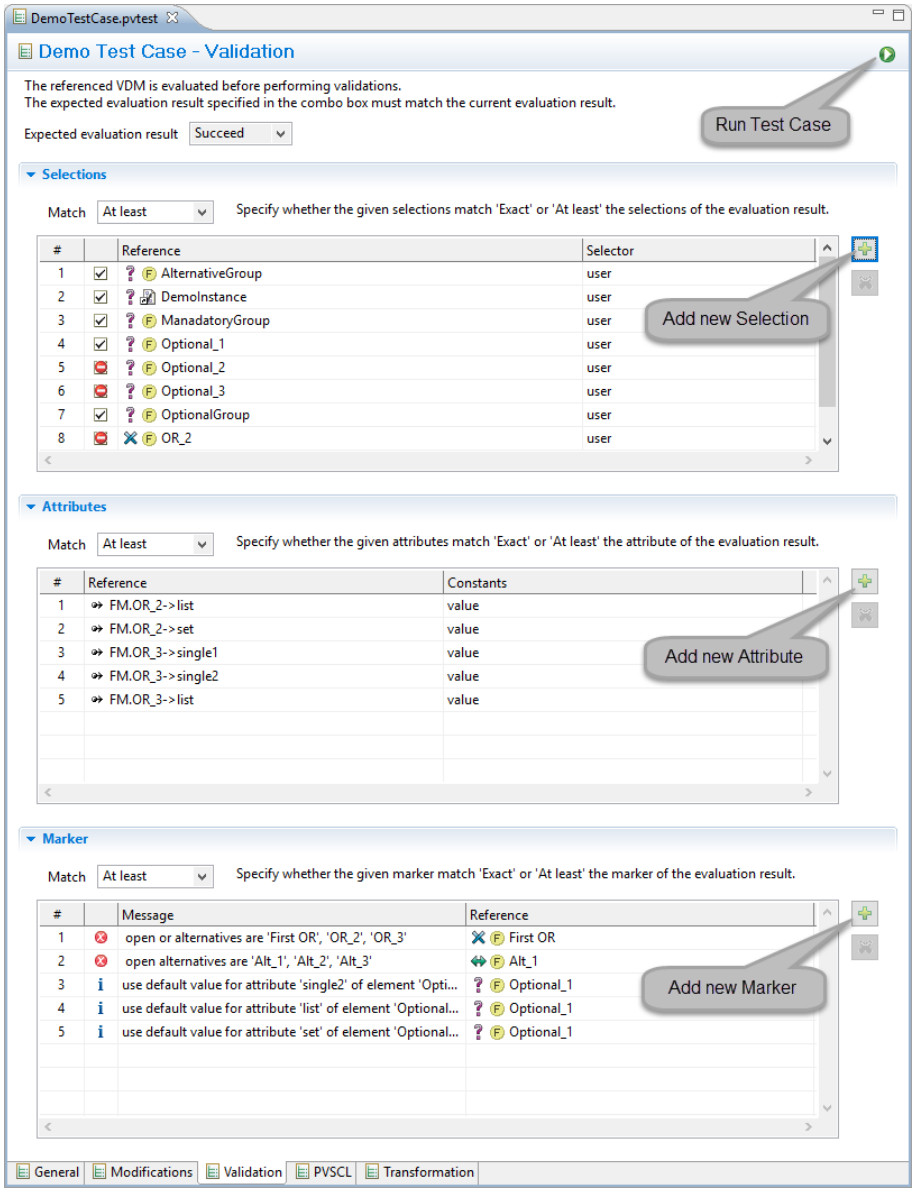
The third tab of the PVUnit Test Case Editor is used to define items, which are checked by the test case. The tab starts with defining the expected evaluation result. During test case execution the referenced variant model is evaluated. The expected result has to match the result of that evaluation.

If the evaluation result matches, selections, attributes and marker defined in this tab are checked with the current state of the variant model. Selections, attributes, marker defined in this tables have to be found in the evaluated variant model. If they are missing or do not match the data from the variant model the Test Case fails.

To define new selections, attribute or marker use the plus buttons next to the tables. A dialog comes up, which is similar to the new PVUnit Test Case wizard pages. The user selects the items to test by checking them in the first table column and finish the dialog with the *OK* button.

Above each table is a *Match* combo box. If this is set to *At least* the elements in the corresponding table are checked with the referenced variant. It is allowed to have more elements in the variant model. If this combo box is set to *Exact* then each element existing in the variant model must be specified in the table otherwise the test case will fail.

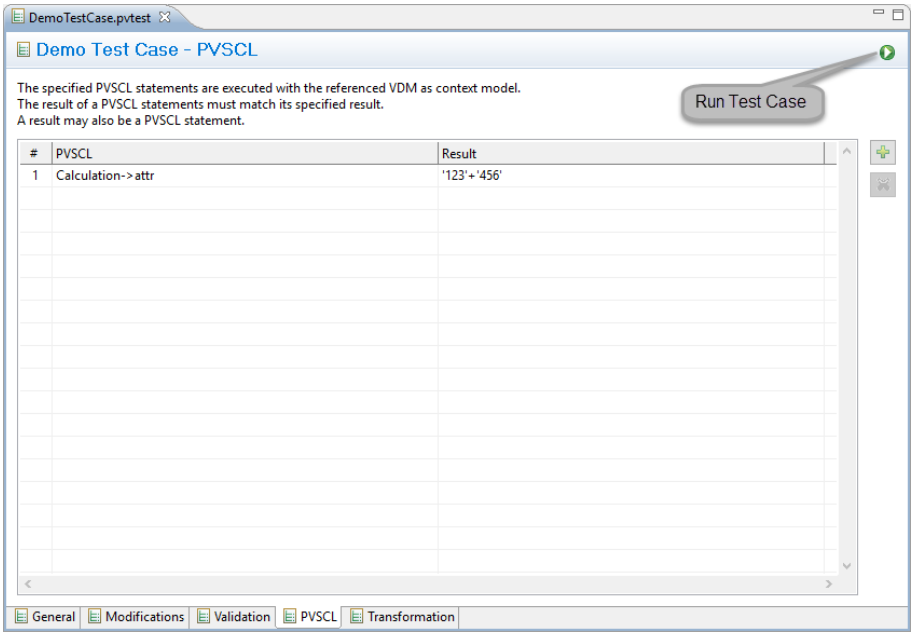
Figure 10. Validations Tab



2.2.4. PVSCL Tab

The PVSCL tab is used to perform more complex checks. Each PVSCL statement in the table is evaluated with the referenced variant model. The result of the script is compared with the defined result. The result itself can be a PVSCL statement. Expected and calculated result have to match otherwise the test case will fail.

Figure 11. PVSCL Tab



2.2.5. Transformation Tab

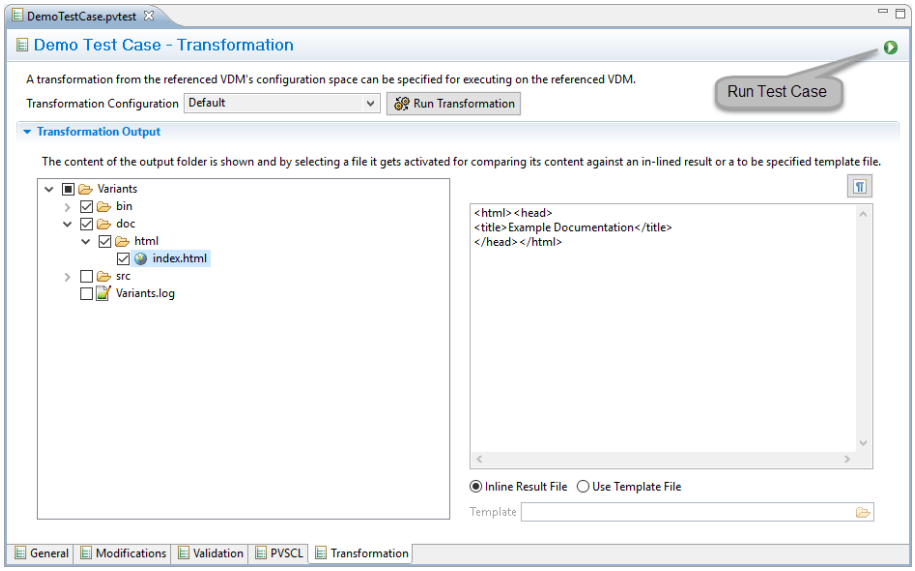
With the last tab of the editor it is possible to use the test case to test transformation results. The combo box at the to is used to select a valid transformation configuration. This configuration is performed by the test case before checking the transformation output.

To define which files of the transformation output are checked first the output needs to be generated once. The *Run Transformation* button is used to perform the transformation. After the transformation is finished the output folder is shown on the left part of the tab page. The user selects the files to check in the tree. The content of the files is saved into the test case. If changes are necessary in the result files, they can be performed on the right side in the text area.

If parts of the result file shall be ignored, the parts are marked in the right text area and *Ignore selected block in result file check* from the context menu is used to ignore the marked parts.

The lower right parent enables the user to define template files instead of inlining the result file content to the test case.

Figure 12. Transformation Tab

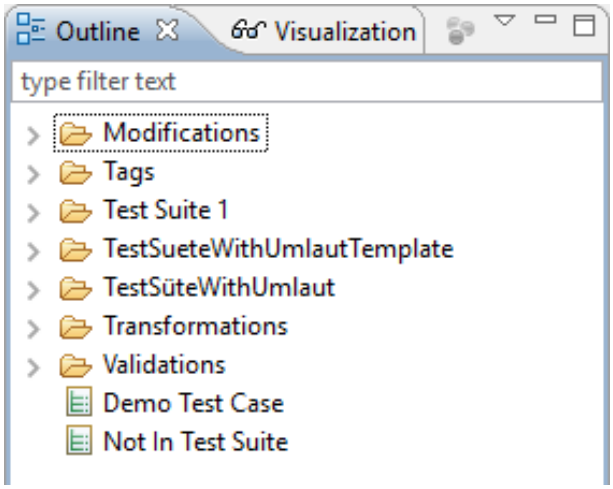


2.2.6. Navigation

Navigation between test cases in the same eclipse project is guided by the outline view. The outline view shows all test cases and suites from the same eclipse project. Test cases can be opened by double clicking the entry in the outline view.

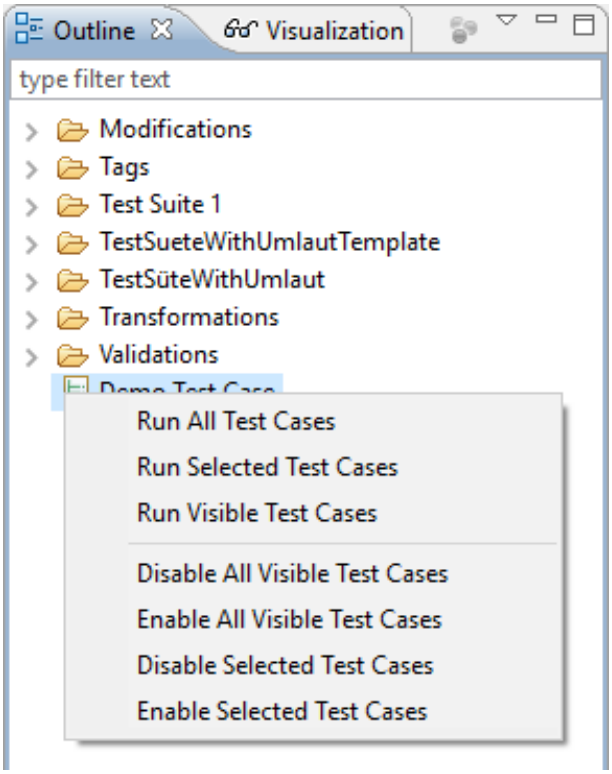
The view allows filtering of the test cases as well. The filter control uses Camel Case filtering on test case names and on names and values of tags defined by that test cases.

Figure 13. Navigation



The *Outline* view provides some useful actions for enabling or disabling multiple test cases at once. Actions working on *All Visible Test Cases* do consider test cases shown in the view only.

Figure 14. Run Actions in Outline View



2.3. Running Test Cases

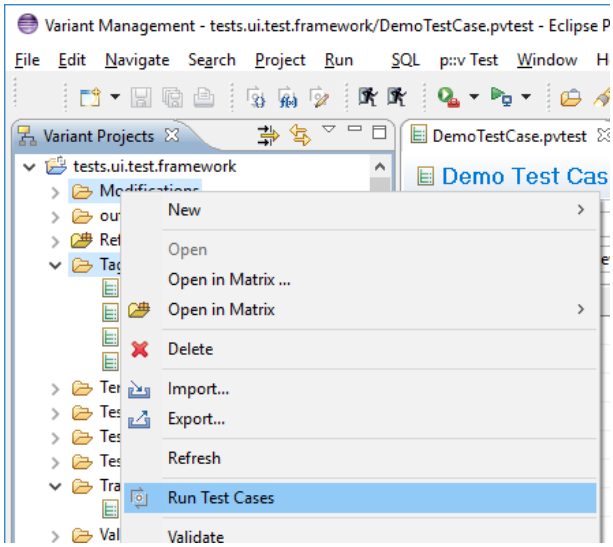
To run test cases 4 possibilities exist.

- A single test case can be run in the test case editor using the *Run Test Case* button. Using this button always performs the test case, even if the test case is disabled
- Run multiple test cases and suites from the *Variant Projects* view
- Run multiple test cases and suites from the *Outline* view
- Rerun multiple test cases and suites from the *Test Run Result* view

2.3.1. Running Test Cases from Projects View

To run test cases and suites from the *Variants Projects* view multiple test cases and/ or test suites are selected in the *Variant Projects* view. The run is started with the *Run Test Cases* action from the context menu.

Figure 15. Run Action in Project View

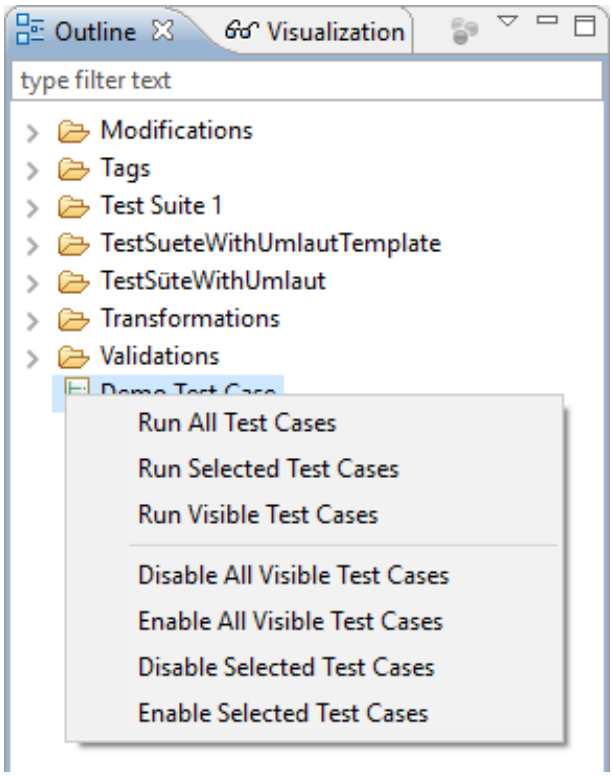


2.3.2. Running Test Cases from Outline View

To run test cases and suites from the *Outline* view multiple test cases and/ or test suites are selected in the *Outline* view. The run can be started with one of three actions from the view menu or context menu of the view.

- *Run All Test Cases* runs all test cases from the current eclipse project, this action ignores the selection in the *Outline* view.
- *Run Selected Test Cases* runs the selected test cases.
- *Run Visible Test Cases* runs all test cases, that are visible in the *Outline* view. This is useful, if a filter was applied to the view and all test case shall be run, that match the filter.

Figure 16. Run Actions in Outline View



2.3.3. Test Run Result View

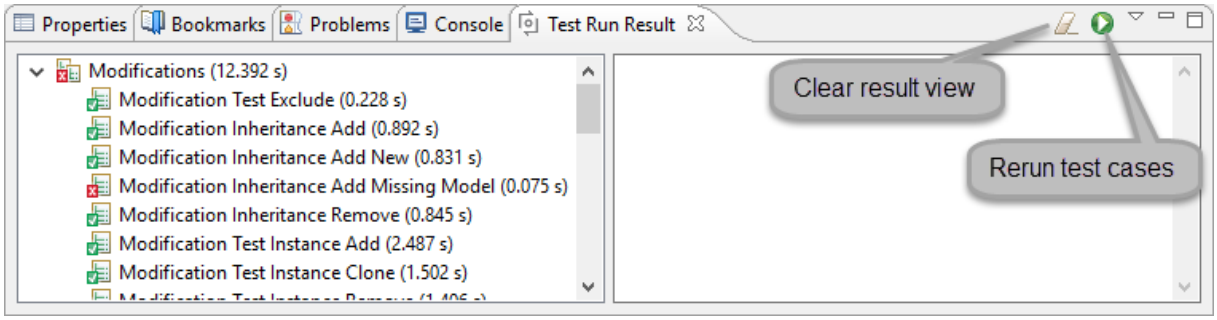
The Test Run Result view informs about the current state during test case execution and presents the result after the run is finished. The result state is presented on the left side of the view. For failed test cases a detailed message is presented on the right text area, after selecting the test case.

With the two tool bar items, the last run can be performed again and the last result can be cleared from the view.

Note

Double clicking on a test case opens the corresponding test case editor. If the test case is failed the editor navigates to the problematic item automatically.

Figure 17. Result View



The last result can be exported in a JUnit compatible result. This export is triggered in the test run result view menu. A JUnit compatible result can be imported to that view as well.

3. External Build Support (Ant Tasks)

pure::variants provides some useful Ant task. They can be used with buildfiles inside Eclipse or in headless mode.

3.1. pv.pvunittest

The `pv.pvunittest` performs the given test cases.

Example:

```
<pv.pvunittest junit_result="report/junit.xml">
  <test path="project/suite/testcase.pvtest" tag="Eclipse" value="3.8.2"/>
  <test path="project/suite" tag="Eclipse" value="3.8.2"/>
</pv.pvunittest>
```

This task has the following attributes:

- **junit_result** is the path to the generated junit result xml file, which will be filled with the actual results of the test run.
- **path** is the absolut or workspace relative path the the test suites or case, which will be performed.
- **tag** is optional. If given the test cases will be filtered, so only test which have the specified tag will be performed.
- **value** is optional. IF given the test cases will be filtered, so only tests which have the specified tag value will be performed.

Note

If both tag and value are given, the filtering is combined, so only test will be performed. which have a specified tag with the specified value.

4. Known restrictions

- pure::variants server projects are not supported. Neither for defining test cases nor for executing test cases.