Table Of Contents

Regression Tests

[Quickstart 2](#_Toc323733302)

[Checklist 2](#_Toc323733303)

[Detailed Description 2](#_Toc323733304)

[Workflow setup 2](#_Toc323733305)

[Example flow (no learner) 3](#_Toc323733306)

[Example flow (learner/predictor) 3](#_Toc323733307)

[Disturber node 3](#_Toc323733308)

[Providing Input Files 4](#_Toc323733309)

[Required Messages and Failing Nodes 4](#_Toc323733310)

[Upload 5](#_Toc323733311)

[Running regression tests 6](#_Toc323733312)

[Testing dialog 6](#_Toc323733313)

[Running from the Command Line 6](#_Toc323733314)

[Notes 8](#_Toc323733315)

# Quickstart

If you just need to refresh your memory: here’s the checklist. Further down is a more comprehensive tour.

## Checklist

* Create your flow, use a reader/writer combination to store the golden table, use the Differ node to compare test results.
* Remove all writers from the flow.
* Reset all nodes that should be tested.
* Add a Testflow Configuration node to your workflow, open its dialog, and
  + Set the workflow owner
  + Optionally add required log messages (errors, warnings, infos)
  + Optionally set required node messages (error, warnings) and if a node is supposed to fail
* Run the test locally on your computer and make sure it passes.
* Upload the workflow to the testflow server

# Detailed Description

Regression tests are little workflows to test one or several nodes.

You need to install the “Test-Node extension for KNIME Workbench” (or get the plug-in org.knime.testing from Subversion).

## Workflow setup

During test execution all nodes of a flow that are not already executed are run. Meaning the regression test flow – at the time of upload – should be in a state where all nodes that should *not* be tested must be *executed* and those that should be tested must be *executable*.

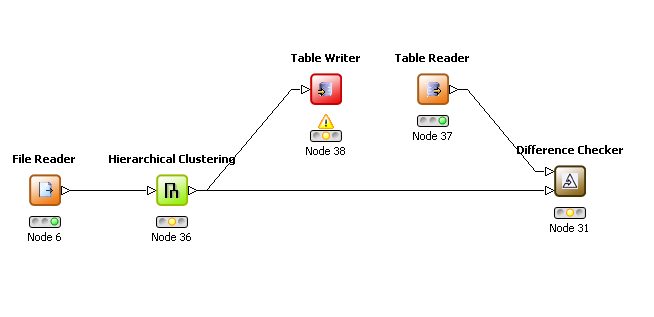
A test succeeds, if at the end all nodes are executed and no unexpected error occurred and no unexpected warning icon appears at a node.

Now, in order to test the output of a node add the *Difference Checker* node to the flow (category *Testing*). It has two inputs and compares the two incoming tables during execute. There are different ways of comparing tables (adjustable in the node’s dialog):

1. Check equivalence; option *TableDiffer*.
2. Check if the input table is empty; option *EmptyTableTest*
3. Compare the outputs from two *Scorer* nodes with tolerance. This way classifiers fail only when they create significantly different errors during classification; see below. Option *LearnerScoreComparator*.

Connect the node to test to the “Test Table” input of the *Difference Checker*, the “Golden Table” input of the node needs to get the correct reference table – the golden table. This golden table can be saved with a TableWriter/Reader combination at testflow creation time, can manually entered via a *Table Creator* node, or can be the output of an executed instance of the same node. If you are using writer nodes to create the golden tables, you **must delete them** before the flow is uploaded (otherwise they will be re-executed every time, writing to that (possibly not existing) temporary directory).

### Example flow (no learner)



File Reader (executed)

**Node under test.**

Must be executable.

While preparing the test, save a correct (golden) table in a temporary location. Delete the node before upload.

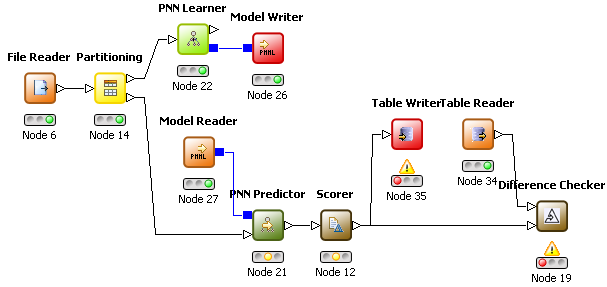
Read the correct (golden) table from the temporary location.

Must be executed.

Configure to test the two input tables for equality.

Must be executable.

### Example flow (learner/predictor)



Executed pre-processing nodes

Writers must be deleted!

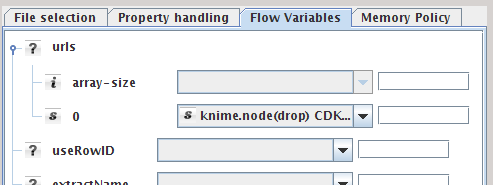
**Node under test**

### Disturber node

Use the “Disturber Node” to test the robustness of your code. This node has one input, three outputs: An output providing the unaltered input table, another delivering an empty table, and the third delivers the input table with 10% of the cells replaced by missing values. You could copy the nodes of your flow three times and connect each copy to an output of the disturber. All nodes should be able to gracefully handle empty tables and missing values somehow.

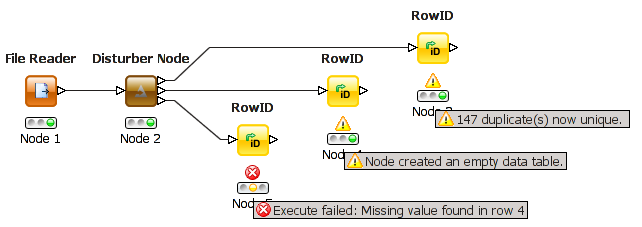
### Providing Input Files

Usually your testflow needs some kind of input that is read by a reader node. Since the read file usually does not exist on other systems, these files should be placed inside the *drop-*folder of the reader node. Create a folder with that name inside the node directory (e.g. File Reader (#1)/drop) and put the input file into it. After closing and re-opening your workflow, you can select the file in the *Flow Variables* tab in the node’s dialog.

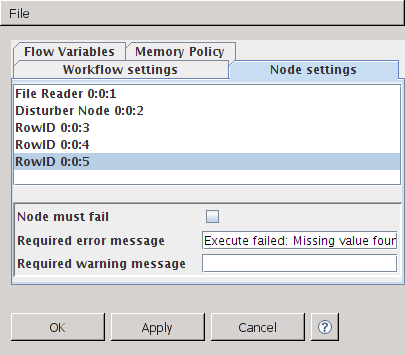
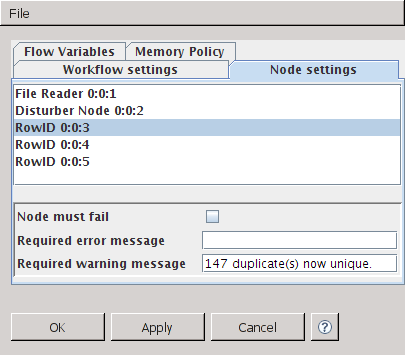


### Required Messages and Failing Nodes

To ensure that a warning and/or an error appears on a node in the workflow, or to check for messages in the log file/console you can use the Testflow Configuration node. In the dialog enter the messages in the corresponding places. Suppose you have a workflow that has the following status after it has been executed:



In the Testflow Configuration node enter the warning and error messages on the nodes as follows:

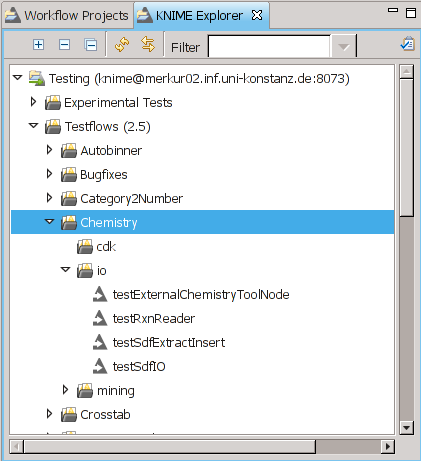


This message fields are pre-filled with the current error message of the selected node (if there is any), indicated by the gray text color. If you want to really use this message click in the text field and possibly edit it. Once the text color is black, the message is used.

If you want to assure that the log file (or console) contains certain error, warning, or info messages you can enter them in the “Workflow settings” tab of the dialog.

### Upload

Make sure, you’ve deleted the golden table writers from the flow. Before you upload the test flow make sure that it contains a *Testflow Configuration* node that has at least the workflow owner set in its dialog. Otherwise the test won’t be run and an error message occurs. Upload the flow into an appropriate workflow group on your testflow server. All workflows in this or any sub-groups will be executed.

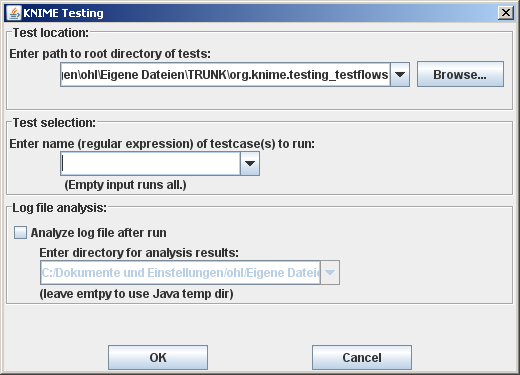
****

## Running regression tests

In order to run the tests locally, create a new R**un configuration** for a new **Eclipse Application**. As program to run choose “**Run an application**”, and select **org.knime.testing.KNIME\_TESTING\_APPLICATION** from the drop-down list. In the “Arguments” tab, you may want to specify “-ea” as VM Argument (to enable assertions) and “-Xmx1G” (to grant enough memory). If you don’t clear the workspace, a history for the following parameters is maintained.

If you run this configuration, a dialog opens asking for the tests to run.

### Testing dialog



Tests in this dir and any subdir are executed (if they match the pattern).

Only tests matching this regular expression are run.

Example: ‘.\*Kmeans’ runs all tests ending with Kmeans

If checked, the log file is analyzed. Parts generated by failing tests are extracted into separate files. A summary file is created.

Enter a directory containing test flows. All tests in that directory or any subdirectory will be executed (if they match the pattern). (If the directory is in your workspace and ends with “\_testflows” it will be set as default).

Enter a pattern (regular expression). All tests whose name match the pattern will be executed (".\*" runs all tests, ".\*Kmeans" e. g. runs the "testKmeans"). Analyzing the log file is only useful if you run many tests (it splits KNIME’s log file into parts, one for each failing test).

### Running from the Command Line

The workflow tests can also be run with a graphical interface from the command line. This is achieved by calling KNIME with the following arguments:

**knime -application org.knime.testing.TestflowRunner –nosplash -consoleLog**

This starts the Eclipse framework (activating all installed plug-ins – without showing the splash screen) and runs the testing application. You can pass the following arguments to the application (after the above parameters):

* The **-root** argument specifies the root directory of where the tests are located. The application traverses this directory and all sub-directories and runs all workflows it finds as a testcase.
* The **–server** argument specifies the URI of a workflow group on a KNIME server where the testing workflows are stored. The format of the URI is **knimefs://<user>:<password>@host[:port]/workflowGroup1**

Either **–root** or **–sever** must be provided.

* With the **-pattern** argument a subset of tests can be selected. Only testcases whose directory name (the directory the **workflow.knime** file is located in) matches the specified pattern (which is a regular expression) are run. Specifying “**.\***” matches all tests.
* The optional argument **–analyze** causes the application to analyze the log file after all tests are run.
* If you want to process the test results further (e.g. in Jenkins), an XML file with the results is created when you specify **–xmlResult** together with a file name.

The framework runs all tests (without saving them) and prints a short summary at the end. Tests that issue error (or fatal) messages during the run are considered failed (unless specified differently in the *Testflow Configuration* node). If a workflow has nodes that are not executed (after the test has run), it is also considered failed (unless specified differently in the *Testflow Configuration* node). Also, if the node has a warning message set (that would cause the little warning icon shop up in the KNIME GUI), which is not specified in *Testflow Configuration* node, the test fails. The log file contains all messages (and errors) occurring during the run.

If **–analyze** is specified, the framework looks at the log file after all tests finish. First, it creates a copy of it (actually only from the part of the last application run – named **\_KNIMELastRunLogCopy+timestamp**), and then it extracts one file for each failing test from the log file (named **<testflowname>+timestamp**). That file contains the part of the log that was produced by this failing test. The first line in that file is the owner of the test (taken from the **owner** file in the directory of the workflow).

It also creates a summary file (named **\_Summary\_+timestamp**) containing a similar summary to the one printed at the end of the test runs.

If testflows without an ownerare found, they are not run. Instead a file called **\_OwnerlessTests+timestamp** is created listing all tests without owner file. The first line in this file is the “regression owner”, which is hard coded in the source.

Also the first lines of the log file are analyzed: If an Exception occurs during Eclipse workbench initialization, a file **\_workbenchInitializing\_+timestamp** is created containing the lines up to the start of the first testflow.

All files created by the **–analyze** option are stored in a new directory called **RegRunAnalyze\_+timestamp** at the specified location.

The timestamp is extracted from the log file. It is the time the application was started.

You can specify a **–data <workspace\_location>** argument – even though the testframe work is not writing to the workspace. Other plug-ins may though. And the .metadata directory will be placed there too, I guess.

# Notes

* If a node should be tested with different settings or different input data you need to create one test for each set of settings/input data. You can either create several testflows with identical structure but different settings or add several independent branches into one workflow.
* It is possible to test multiple nodes in one regression test (as all nodes that are executable will be executed) – it’s not advisable though. If one node is broken, there would be many tests failing and it would be hard to figure out which node is actually causing the pain. It’s just harder to debug. Create one test per node. The more the merrier.
* During the regression run the node’s views will be opened before the node gets executed. This is to test the updateModel method. So, there are windows popping up – and automatically closing during the run.

© Copyright 2003 – 2012, Chair for Bioinformatics and Information Mining (Prof. M. Berthold) and KNIME GmbH, Konstanz, Germany.