

Define Test Suites with Annotations

Annotations are single-line comments and belong in the package header. Put a newline between package and procedure annotations.

Package Annotations

--%suite(description)	Package is a test-suite (<i>optional</i> description). <i>Mandatory</i>
--%suitepath(org.utplsql)	Groups suites in hierarchical namespaces
--%rollback(auto manual)	Automatic (<i>default</i>) / manual transaction control
--%context(description)	Starts/ends sub-suite in suite. Can be nested.
--%endcontext	
--%disabled	Tests of suite/context won't be executed
--%beforeall(procedure [, ...])	Procedure(s) to run before/after all tests in the suite/context
--%afterall...	
--%beforeeach(procedure [, ...])	Procedure(s) to run before/after each test in the suite/context
--%aftereach...	
--%tags(myTag[, ...])	Label a test/context/suite

Procedure Annotations

--%test(description)	Procedure is a test (with optional description)
--%disabled	Test won't be executed
--%throws(exception [, ...])	Test expects exception(s) to be thrown
--%beforeall, --%afterall	Procedure to run before/after all/each tests in the suite/context
--%beforeeach, --%aftereach	
--%beforetest(procedure [, ...])	Procedure(s) to be run before/after annotated test
--%aftertest(procedure [, ...])	
--%tags(myTag[, ...])	Label test

Run Tests

<code>exec ut.run();</code>	Run all tests to the default reporter (DBMS_OUTPUT)
<code>select * from table(ut.run());</code>	Run all tests and display output as result set

Consider using [utplsql-cli](#) to easily run tests

Run specific tests

<code>ut.run('my_suite_pkg')</code>	Run only the specified suite by name (current schema)
<code>ut.run('HR')</code>	Run all tests in specific schema
<code>ut.run('my_suite.test1, my_suite.test2')</code>	Run list of items
<code>ut.run(':my_suite.path')</code>	Run specific suite path

Run Parameters

<code>a_force_manual_rollback</code>	True/False. Run specified test and don't rollback changes (leaves open transaction)
<code>a_random_test_order</code>	True/False. Run tests in random order
<code>a_color_console</code>	True/False. Color-code output (for Windows see ANSICON)
<code>a_client_character_set</code>	Define report's character-set
<code>a_tags</code>	Run tests with any specified tags . Use <code>a_tags=>' -tag1'</code> to exclude a tag

Reporting: Run alternative Reporter

```
exec ut.run(ut_junit_reporter());
```

Reporters

<code>UT_DOCUMENTATION_REPORTER</code>	Textual pretty-print of test results. <i>Default</i>
<code>UT_COVERAGE_HTML_REPORTER</code>	HTML Coverage Report, includes SourceCode
<code>UT_JUNIT_REPORTER</code>	Test results conforming to JUnit 4 and above
<code>UT_REALTIME_REPORTER</code>	Test results and additional information as XML. Allows to show progress information
<code>UT_SONAR_TEST_REPORTER</code>	JSON Test results designed for SonarQube

See [full list](#) of built-in reporters. See documentation for [coverage](#) and [reporters](#).



Expectation Syntax

Base expectation

```
ut.expect( actual_value ).to_( matcher );
```

Negated expectation

```
ut.expect( actual_value ).not_to_( matcher );
```

Shortcut syntax (recommended)

```
ut.expect( actual_value ).to_matcher;
```

```
ut.expect( actual_value ).not_to_matcher;
```

Provide additional expectation description

```
ut.expect(1, 'Additional text').to_equal(3);
```

Simple Matchers

```
ut.expect(1=1).to_be_true();
```

```
ut.expect(1 is null).to_be_false();
```

```
ut.expect(null).to_be_null();
```

```
ut.expect(to_clob('ABC')).to_be_not_null();
```

```
ut.expect( 3 ).to_be_between( 1, 3 );
```

```
ut.expect( 3 ).to_be_greater_or_equal( 2 );
```

```
ut.expect( 2 ).to_be_greater_than( 1 );
```

```
ut.expect( 3 ).to_be_less_or_equal( 3 );
```

```
ut.expect( 3 ).to_be_less_than( 4 );
```

Equality Matchers

```
ut.expect( 'a dog' ).to_equal('a dog',
    a_nulls_are_equal => false );
```

a_nulls_are_equal is **true** by default

equal on objects

```
ut.expect( anydata.convertObject( l_expect )
    .to_equal( anydata.convertObject( l_actual ) );
```

equal on collections

```
ut.expect( anydata.convertCollection( l_expect )
    .to_equal( anydata.convertCollection( l_actual ) );
```

It's not possible to compare rowtypes and records directly. You have to wrap them in a table of record, select from them and use cursor comparison

Advanced Matchers

```
be_like ut.expect( 'Lorem ipsum' )
        .to_be_like( '%re%su' );
```

```
be_like ut.expect( 'Lorem ipsum' )
with escape .to_be_like(
    a_mask => '%rem\_%',
    a_escape_char => '\' );
```

see [Oracle like operator](#)

```
match ut.expect('some value')
        .to_match('^some.*');
```

```
match ut.expect( '123-456-ABcd ' )
with options .to_match(
    a_pattern=>'\d{3}-\d{3}-[a-z]',
    a_modifiers=>'i');
```

See [regexp_like function](#)

Cursor Comparison

Basic example

```
open l_expected for select * from all_objects;
open l_actual for select * from all_objects;
ut.expect( l_actual )
    .to_equal( l_expected )
    .join_by( 'COLUMN_NAME' )
    .exclude( ut_varchar2_list( 'ID', 'SOMECOLUMN' ) )
```

Special Matchers

```
ut.expect( l_actual ).to_have_count( 5 );
ut.expect( l_actual ).to_be_empty();
```

Helpers

Type: List of strings

```
ut_varchar2_list( 'String 1', 'String 2', ... )
```

Type: List of reporters

```
ut_reporters( ut_documentation_reporter(), ut_junit_reporter() )
```

Select all tests

```
select * from table(
    ut_runner.get_suites_info( USER, 'PACKAGE_NAME' )
    where item_type = 'UT_TEST');
```

