keywords-outline

Hugo

September 27, 2020

Contents

-*- org -*-

1 package Heterarch::Test::Element;

All the test elements described below derive from this class. No keywords.

2 package Heterarch::Test::Module;

A file with test specifications is called a test module. A test module groups related tests. For example core functions of an algorithm can be tested in one test module, edge cases of the same algorithm can be tested in a second test module. During test execution the file with test specifications is represented with this class (a package implements a class).

A test module recognizes optional `preparation` and `reparation` clauses to prepare the environment before tests are executed, and restore the environment to its original state after test execution, respectively. A test module contains a list of command definitions.

2.1 keywords

The keywords recognized in a test module are:

2.1.1 `harnessing`

1. `preparation`
   - (a) `preparer`
     - i. `class`
     - ii. `applicators`
       - A. `method`
       - B. `arguments`
     - iii. `system_commands`
2. `reparation`

    (a) `class`
    (b) `applicators`
        i. `method`
        ii. `arguments`
    (c) `system_commands`

2.1.2 `command_definitions`

The list of command definitions of the module.

3 package Heterarch::Test::CommandDefinition;

Can have preparation and reparation clauses.

Contains a command specification that specifies how to connect with the application to be tested. After this connection is successfully established, the command tests are executed one by one and results recorded.

Contains a possible empty list of command tests. If the list of command tests is empty, the only test in the command definition is connection establishment with the application to be tested.

3.1 keywords

The keywords recognized in a command definition are:

3.1.1 `harnessing`

1. `preparation`
    (a) `preparer`
        i. `class`
        ii. `applicators`
            A. `method`
            B. `arguments`
        iii. `system_commands`
2. `reparation`
    (a) `class`
    (b) `applicators`

        i. `method`
        ii. `arguments`
    (c) `system_commands`

3.1.2 `command_tests`

The list of command tests for the command definition.

3.1.3 `class`

3.1.4 `command`

3.2 "Heterarch::Test::CommandDefinition";

default, only possible for command definitions that have no command tests.

3.3 "Heterarch::Test::CommandDefinition::PerlClass";

```
if ($command_definition->{class})
```

The perl class referenced by the class keyword is instantiated using a constructor without arguments. The instance of the class is then called using the methods named in the write keyword of the command tests. The result is checked with the method named in the read keyword.

3.3.1 keywords

1. `class`

3.4 "Heterarch::Test::CommandDefinition::PerlCode";

```
elsif (ref $command_definition->{command} eq 'CODE')
```

The perl code found inside the command keyword is run before command tests are executed.

3.4.1 keywords

1. `command` (perl code)

3.5 "Heterarch::Test::CommandDefinition::Interactive";

`else`

If no class keyword is found and if the command keyword is not a code reference to perl code, the command keyword is used as the name of a system command. This system command is executed to establish a connection with the application to be tested.

4 package Heterarch::Test::CommandTest;

4.1 "Heterarch::Test::CommandTest";

Default class, normally not used.

4.2 "Heterarch::Test::CommandTest::CommandObject";

`if (ref $command_test->{write} eq 'ARRAY')`

If the `command_test` has a `write` clause with a list, it is used as a list of methods that are invoked on the perl object previoulsy instantiated. The result is then verified with the `read` clause that is used as literal text.

4.2.1 keywords

1. `write`
   (a) `method`
   (b) `arguments`
2. = read=

4.3 # "Heterarch::Test::CommandTest::PerlCode";

`# elsif (ref $command eq 'CODE')`

4.4 "Heterarch::Test::CommandTest::Interactive";

`else`

This class is used to test the application interactively. Other classes derive from it to implement specific testing capabilities. If the specific tests fail, this class may still decide to do further testing (for instance using numerical comparisons rather than textual comparisons).

4.4.1 keywords

1. `read` What is expected for interactive output from the application. The value of this clause can be literal text, an array or a hash / dictionary with keywords that are explained in the following sections.
2. `shell` Runs the given system shell command for a maximum of `timeout` seconds.
3. `timeout` The maximum time to wait before reporting a failure that the application did not generate the expected output.
4. `wait` A time specified in seconds the tester will wait after applying the `write` clause, and before applying the `read` clause.
5. `write` The text to write to the application.

4.4.2 "Heterarch::Test::CommandTest::Interactive::ShellTester";

```
else
    my $tester = $command_test->{tester};
    if ($tester)
```

1. keywords
   (a) `tester`
      i. `shell` The shell command is run and its output is used as if it was produced by the application being tested.

4.4.3 "Heterarch::Test::CommandTest::Interactive::Literal";

```
else
    my $read = $command_test->{read};
    elsif (defined $read)
if (!ref $read)
```

Literal comparison between what is expected and what is produced by the application.

1. keywords

   (a) `white_space` If the value of the `white_space` clause is equal to `convert seen 0a to 0d 0a newlines`, newlines in the expected literal text are converted from `0x0a` to `0x0d 0x0a` newlines.

   The implementation of the tester should be changed so that the value of this key corresponds with the implementation.

4.4.4 "Heterarch::Test::CommandTest::Interactive::Regex";

```
else
    my $read = $command_test->{read};
    elsif (defined $read)
elsif (ref $read eq 'ARRAY')
```

The `read` clause is an array with as its first element the value `-re`. The second value is used as a regular expression to compare with the output that the application has produced.

4.4.5 "Heterarch::Test::CommandTest::Interactive::Alternatives";

```
else
    my $read = $command_test->{read};
    elsif (defined $read)
else
    if ($read->{alternatives})
```

The `read` clause has a keyword `alternatives` that is used to compose a regular expression consisting of the different alternatives in the expected output. This expressions is then compared with the output that the application has produced.

1. keywords

   (a) `alternatives`

4.4.6 "Heterarch::Test::CommandTest::Interactive::File";

```
else
    my $read = $command_test->{read};
    elsif (defined $read)
else
    elsif ($read->{application_output_file})
```

The `read` clause has a the keywords `application_output_file` and either `expected_output_file` or `expected_output`. The keyword `application_output_file` defines the name of the file that will be produced by the application during the test. If the keyword `expected_output_file` is present, it is used as a reference to a file whose contents will be compared against the contents of the application output file. Otherwise the contents of the clause `expected_output` is used and compared against the contents of the application output file.

1. keywords
   (a) `application_output_file`
   (b) `expected_output`
   (c) `expected_output_file`

4.4.7 "Heterarch::Test::CommandTest::Interactive::Shell";

```
else
    my $read = $command_test->{read};
    elsif (defined $read)
else
    elsif ($read->{shell})
```

1. keywords
   (a) `shell` The command in the `shell` clause is executed as a system shell command and its output is compared against the output produced by the application.