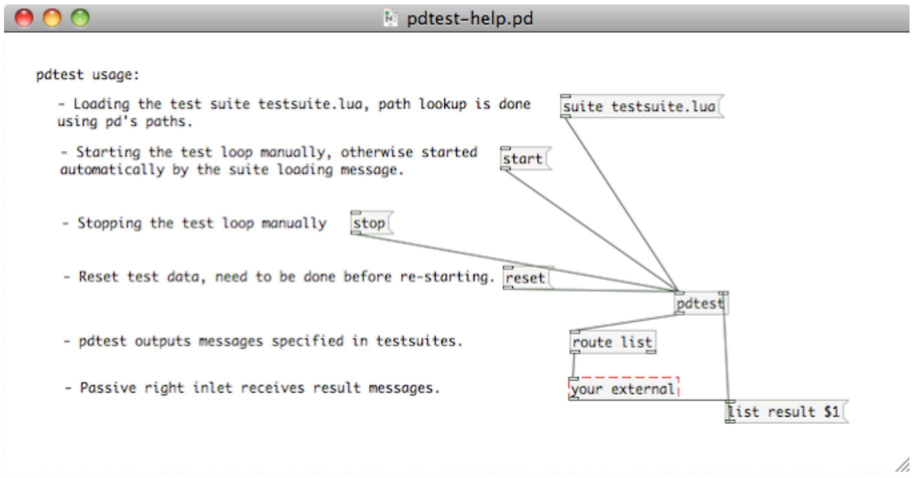
Pure Data testing external

Async Functional testing for Pure Data using Lua scripting.

PdTest is intended at developers of Pure Data externals that needs functional testing for their project. Functional tests are organised as test suites, grouping test cases that coordinates the many individual tests. The pdtest external takes no creation arguments and features 2 inlets, the active left for loading test suites, starting, stopping and resetting the test loop while the passive right inlet receives result messages. Test are built using standard Lua scripts, with provided helper functions for declaring the test structure. Refer to **Lua documentation** to learn more about test scripts syntax.



pdtest usage:

- Loading the test suite testsuite.lua, path lookup is done using pd's paths.
- Starting the test loop manually, otherwise started automatically by the suite loading message.
- Stopping the test loop manually
- Reset test data, need to be done before re-starting.
- pdtest outputs messages specified in testsuites.
- Passive right inlet receives result messages.

Test Suite Syntax:

Building a test suite is easy when you know its basics:

The pdtest namespace

```
pdtest      -- the pdtest namespace object
pdtest.suite("suite name")                          -- suite creation function
pdtest.message({"YOUR","MESSAGE","LIST"})           -- sends lua table as test message through outlet
pdtest.raw_message({"YOUR","RAW","MESSAGE","LIST"}) -- sends lua table as raw message through outlet,
                                                    -- raw message are not monitored for testing
pdtest.post("string to post")                       -- post to pd console
pdtest.error("string to signal")                    -- sends error to pd console
```

Suites functions

```
mysuite = pdtest.suite("my test suite")
mysuite.setup(function()                            -- set a setup function to be called
    pdtest.raw_message({"SOME","SETUP","MESSAGE"})  -- before all of this suite tests
end)
mysuite.teardown(function()                         -- set a cleanup function to be called
    pdtest.raw_message({"SOME","CLEANUP","MESSAGE"}) -- after all of this suite tests
end)
mysuite.case("my test case")                        -- define a new test case
```

Cases functions

```
mycase = mysuite.case("my test case")
mycase.setup(function()                             -- set a setup function to be called
    pdtest.raw_message({"SOME","SETUP","MESSAGE"})  -- before all of this case tests
end)
mycase.teardown(function()                          -- set a cleanup function to be called
    pdtest.raw_message({"SOME","CLEANUP","MESSAGE"}) -- after all of this case tests
end)
mycase.test({"MY","TEST","MESSAGE"})                -- simple test form, takes a lua table as test message
mycase.test(function()                              -- long test form, function body can contain only one
    pdtest.message({"MY","TEST","MESSAGE"})         -- pdtest.message() call, but unlimited pdtest.raw_message().
end)
```

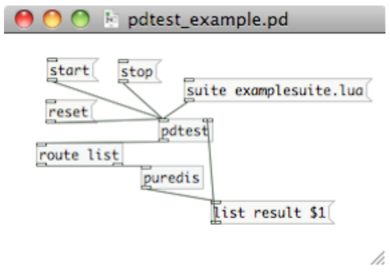
Tests functions

```
mycase.test({"MY","TEST","MESSAGE"}                 -- tests are made by calling condition methods
    ).should:equal(                                 -- like equal() from the .should namespace
        {"TEST","DESIRED","OUTPUT","MESSAGE"})      -- notice the colon for calling condition methods
mycase.test({"MY","TEST","MESSAGE"}                 -- tests can also be reversed by calling them
    ).should.nt:equal(                              -- from the .should.nt namespace
        {"TEST","NOT","DESIRED","OUTPUT","MESSAGE"})
```

Conditions methods

```
.should:equal(                                        -- equal method test for message content equality
    {"TEST","DESIRED","OUTPUT","MESSAGE"})
.should.nt:equal(                                     -- not equal method
    {"TEST","DESIRED","OUTPUT","MESSAGE"})
--
.should:match("%d+%s%a+")                             -- match method matches individual result atoms
                                                      -- against string pattern according to "lua's matching":http://www.lua.org/pil/20.2.html
                                                      -- mechanism.
.should.nt:match("%d+%s%a+")                          -- not match method
--
.should:be_true(function(result)                      -- be_true method test if provided function returns
    if result[1] == "OK" then                         -- true or false
        return true
    else
        return false
    end
end)
.should.nt:be_true(function(result)                   -- not be_true (be_false...)
    if result[1] == "ERROR" then
        return true
    else
        return false
    end
end)
```

Example Tests Suite



examplesuite.lua

```
mysuite = pdtest.suite("TestSuite") -- initializing a new test suite named "TestSuite"
```

```
mysuite.setup(function()                -- called before every test in the suite
  pdtest.raw_message({"command","flushdb"})
end)
```

```
mysuite.teardown(function()             -- called after every test in the suite
  pdtest.raw_message({"command","flushdb"})
end)
```

```
mysuite.case("Server Info"              -- new test case named "Server Info"
  ).test({"command", "INFO"}            -- will output message "list command INFO"
    ).should:match("^redis_version")    -- test will pass if result received on right
                                        -- inlet starts with the string "redis_version"
```

```
mysuite.case("Reality Check"            -- second test case named "Reality Check"
  ).test({"command", "dummy"}           -- will output message "list command dummy"
    ).should:match("ERR"                -- test will pass if result received on right
  ).test({"command", "dbsize"}          -- inlet match the string "ERR"
    ).should.nt:match("ERR")            -- second test will pass if result doesn't match "ERR"
```

```
mycase = mysuite.case("Basic tests")    -- new test case named "Basic tests"
```

```
mycase.setup(                           -- called before every test in the case
  function() pdtest.raw_message({"command","SET","FOO","BAR"})
end)
```

```
mycase.test({"command", "GET", "FOO"}   -- test will pass if result equals "BAR"
  ).should:equal({"BAR"})
```

```
mycase.test(function()                  -- here a test function is passed instead
  pdtest.raw_message({"command","DEL","FOO"})
  pdtest.message({"command","EXISTS","FOO"})
end).should:equal("0")                  -- equal is passed a string, when result list
                                        -- contains only one member
```

```
mycase.test(
  {"command","SETNX","FOO","BAT"}
  ).should:be_true(function(result)     -- test method '.should:be_true' takes a function
    if result[1] == "1" then            -- for which it provides a 'result' argument
      return true                       -- the function must return true or false
    else
      return false
    end
  end)
```

pd console

```
pdtest: loading testfile examplesuite.lua
pdtest: TestSuite -> Server Info < command, INFO >  |> OK
pdtest: TestSuite -> Reality Check < command, dummy >  |> OK
pdtest: TestSuite -> Reality Check < command, dbsize >  |> OK
pdtest: TestSuite -> Basic tests < command, GET, FOO >  |> OK
pdtest: TestSuite -> Basic tests < function >  |> OK
pdtest: TestSuite -> Basic tests < command, SETNX, FOO, BAT >  |> FAILED > 0 is not true
pdtest: !!! Test Completed !!!
pdtest: 1 Suites | 3 Cases | 6 Tests
pdtest: 5 tests passed
pdtest: 1 tests failed
```