# Specs and folders related to regression testing Plang\Doc\TestDocs\

RegressionTestTool.docx: instruction on running the test tool and adding new tests (this doc)

TestingFramework.docx: list of P features to test and how test are organized

Tests.xlsx: list of all tests with features tested and type of error detected (or not)

Sample test folders:  
Plang\Tst\SampleTests

Regression test suite:  
Plang\Tst\RegressionTests

# Regression process for developers

1. Each developer should run regression (“testP.bat RegressionTests”) before pushing the changes.
2. If the regression fails, the developer responsible for the change should figure out why if fails. If the acceptors have to change to accommodate new valid outputs, the acceptors should be reset by running

Tests\...\X> ...\Tools\CheckP\bin\debug\checkP.exe –cfg:”testconfig.txt” –add

where X is the test directory.

1. Run testP again and make sure that all the tests pass.
2. Push the changes AND new acceptors.

For now, step #2 is manual: you have to run checkP for each test under RegressionTests (there are 7 of them for now, and the new tests are being added).

I will automate the reset process ASAP. For now, I would be happy to do step #2 for you.

In case we decide to change testconfig.txt (to add new outputs, change includes, etc.), please let me know and I will do the regression reset.

# Test tool instruction

“Plang\Tst\testP.bat <root folder with tests>” will build a debug drop of PLang and run tests against it. The testing framework does the following:

1. Runs pc.exe and if no error, zinger.exe. In the future, it will invoke runtime if no error is reported in zinger.exe.
2. Concatenates all output written to streams, the return code of the exe, and some files into a single output file.
3. Checks if there is some “acceptor” file that matches the output file exactly.
4. The test passes if the output is accepted, otherwise it fails.

The root folder for regression tests is Plang\Tst\RegressionTests. There are subfolders according to the features listed in the spec “\plang\Doc\TestDocs\TestingFramework.docx”.

For example, feature test subfolders have a number in the name which corresponds to the TestingFramework.docx enumeration in the “P Features to Test” section. At this point, a decision has been made to have subfolders for the top-level features (numbers 1 thru 5), such that we don’t have too many subfolders. More detailed information about specific (sub)-features that a particular test is checking can be found in two places:

* “dsc” directive of the testconfig.txt in the test folder; a full feature number from “TestingFramework.docx” should be placed here, for example:

“dsc: 1.7. Transition to undefined state: error”

* full list of tested features can be found in the spreadsheet “\plang\Doc\TestDocs\Tests.xlsx”

As more tests are created, new subfolders can be added to RegressionTests. This would require an addition to the “P Features to Test” section.

To create a test case X, you need to do the following:

1. Create a new directory X under Plang\Tst\RegressionTests\F\E, where F is the “Feature” subfolder and E is “error type” subfolder (StaticError\DynamicError\Correct).
2. Add the test case to Plang\Doc\TestDocs\Tests.xlsx.
3. Create a file called “X\testconfig.txt”, which will define how to run your test case.
4. Create a file called “X\myfile.p”, which is your test case (the name of myfile.p doesn’t matter)
5. When you run “testP.bat” your test case will be detected and executed.
6. A test case will be executed as if the working directory is X
7. Even if the exe being tested crashes, then this will still be captured without crashing the testing tool.

You can see four examples of test case folders under “Plang\Tst\SampleTests”. In particular, “SampleTests\SamplePcPassZingFail” and “SampleTests\SamplePcZingPass” have sample “testconfig.txt” files for running both pc.exe and zinger.exe.

The contents of “testconfig.txt” is a sequence of directives of the form “directive: data”. The possible directives are:

runPc: zinger.exe to run. Must be exactly one such directive.

runZing: pc.exe to run. Must be exactly one such directive.

runPrt: runtime exe to run. Must be exactly one such directive.

argPc: An arg to pass to pc.exe. If more than one arg directive, then args are passed in order

argZing: An arg to pass to zinger.exe. If more than one arg directive, then args are passed in order

argPrt: An arg to pass to runtime. If more than one arg directive, then args are passed in order

incPc: A file that should be included as output for pc.exe. Can be more than one such directive.

incZinger: A file that should be included as output for zinger.exe. Can be more than one such directive.

incPrt: A file that should be included as output for runtime. Can be more than one such directive.

acc: A directory containing acceptor files (more about this later). Must be exactly one such directive.

del: A file to delete before running test case. Can be more than one such directive.

igp: Ignores output sent to the prompt by run

dsc: A description of this test case

The “testP.bat” tool actually calls a tool called “CheckP.exe” located in “Tst\Tools\CheckP”. To run a single test case in directory X, do:

Tests\...\X> ...\Tools\CheckP\bin\debug\checkP.exe –cfg:”testconfig.txt”

Check will also create a new acceptor file, if you decide the output produced by check is the correct output.

Tests\...\X> ...\Tools\CheckP\bin\debug\checkP.exe –cfg:”testconfig.txt” –add

This creates a file called acc\_i.txt. Initially, i = 0. Each time you run with the –add flag, CheckP finds the index of largest acceptor and then creates a new acceptor with an index one larger. The only reason you need more than one acceptor file would be if the generated output would differ between operating systems / platforms. If this is the case, the tool will compare the output against each of the acceptor files, and if one of them matches, the test passes.

Otherwise, one acceptor should be enough. If you update the test case and the acceptance criterion changes, then you should delete the stale acceptors, and then run checkP.exe with the –add flag.

Additional tips on using the tool:

* if you include a generated output file in the acceptor criterion, make sure you delete the generated file first with a “del: file” directive. Otherwise, a stale file could be hanging around in the directory and accidentally cause a bad test to pass
* make sure the acceptor files and testconfig.txt are checked in
* make sure outputs can be tuned to remove timestamps, time-to-completion, absolute paths. All of these are machine specific and will be captured by the testing tool. One someone else’s machine, these values may be different, cause a good output to fail to the test
* if P executables change, then run testP.bat before running checkP.exe, such that checkP.exe uses updated executables
* if you want to run regression only on a specific feature folder F or on a single test X, run

“testP.bat RegressionTests\F” or “testP.bat RegressionTests\F\E\X”