# 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 with hyperlinks to the existing tests

Tests.xlsx: list of all tests with more information about the tests: features

tested, test result, etc.

Primary Regression test suite: Plang\Tst\RegressionTests

Secondary (optional) samples that implement protocols: Plang\Tst\SamplesProtocols

# Regression process for developers

1. Ideally, each developer should run regression as “testP.bat RegressionTests” before pushing the changes. “SamplesProtocols” tests take longer and are optional for now.
2. If the regression fails, the developer responsible for the change should figure out why it fails.

See section “How to analyze regression results” below for tips on how to do it for all the failed tests at once.

If the acceptors have to change to accommodate new valid outputs, the acceptors should be reset.

**How to reset acceptors:**

First, delete old acceptors (acc\_X.txt) in all test directories.

To reset multiple tests, use

“Plang\Tst> testP.bat <path to the root folder with tests> reset”

For reset in a separate test directory, there’s an alternative and faster way:

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

where X is the test directory.

“CheckP.exe” does not rebuild PLang – hence, it is faster than “testP.bat”.

More details on running CheckP.exe are below.

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

If testconfig.txt need changes (to add new outputs, change includes, etc.), please let Ella know and she will do the regression reset.

# Test tool instruction

“Plang\Tst> testP.bat [<path to the root folder with tests>] [reset]” will build a debug drop of PLang and run tests against it.

If you run testP.bat with no parameters, both regular and long regressions will run.

To reset regression, the parameter <path to the root folder with tests> is mandatory.

The testing framework does the following:

1. Runs P tools specified in the configuration file “testconfig.txt”, starting with pc.exe (if specified). If pc.exe does not report errors, then zinger.exe and/or runtime run, if specified.
2. Concatenates all output written to streams, the return code of the exe, and some additional files defined in “testconfig.txt” into a single output file.
3. If no “reset” flag: checks if there is some “acceptor” file that matches the output file exactly.

The test passes if the output is accepted, otherwise it fails.

With “reset” flag: new acceptor files are generated; this is equivalent to calling CheckP with “-add” flag for all tests under the root folder. In particular, when running “testP.bat” with the “reset” flag, a file called acc\_i.txt is created in each test directory. Initially, i = 0. Each time you run with the –add flag, testP 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 testP.exe with the –reset flag.

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, four feature test subfolders have a number in the name which corresponds to the TestingFramework.docx enumeration in the “P Features to Test” section. In addition, there are two more subfoders “Integration” and “Combined” – see explanation in “\plang\Doc\TestDocs\TestingFramework.docx”.

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” is placed there, for example:

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

* more detailed 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 of “\plang\Doc\TestDocs\TestingFramework.docx”.

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).

There are six “Feature” subfolders:

Feature1SMLevelDecls

Feature2Stmts

Feature3Exprs

Feature4DataTypes

Combined

Integration

For more information on specific features in each category, look in the TestingFramework.docx document, sections “P Features to Test”, “Integration Tests”, “Combined Tests”.

1. Create a file called “X\myfile.p”, which is your test case (the name of myfile.p doesn’t matter).

It is very helpful to have a comment in the beginning of myfile.p which briefly describes the purpose of the test.

1. Create a file called “X\testconfig.txt”, which will define how to run your test case.

For templates of the “testconfig.txt” files, look in similar test directories **in the same “error type” subfolder.** Keep in mind that “testconfig.txt” files are different for different error types (static error, dynamic error, correct).

1. Run “testP.bat” with the root directory X with “reset” flag. Make sure that the result is what you expected.
2. A test case will be executed as if the working directory is X.
3. Even if the exe being tested crashes, then this will still be captured without crashing the testing tool.

The last two steps are optional and can be delegated to Ella:

1. Add the test case to Plang\Doc\TestDocs\Tests.xlsx.
2. Add a hyperlink to the test file to “\plang\Doc\TestDocs\TestingFramework.docx”, to the relevant section.

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

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

runZing: zinger.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.

incZing: 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”

CheckP will also create a new acceptor file, if you decide the output produced by CheckP 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 (or run “Bld\build.bat –d”)
* 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”

# How to analyze regression results

“testP.bat” generates a summary file “failed-tests.txt” that includes all directories with failed tests.

To look at the differences between the acceptor file “acc\_0.txt” and the newly generated output file “check-ouput.log” for all failed tests, a batch file “display-diffs.bat” is generated by “testP.bat”. For now, it supports “kdiff3” tool which you can install from here:

<http://sourceforge.net/projects/kdiff3/files/>

The batch file “display-diffs.bat” assumes that the full path to kdiff3 is added to the environment variable %PATH%, for example, as follows:  
set PATH=%PATH%;C:\Program Files\KDiff3