# 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

# Directory structure for regression tests

The folder structure for “Plang\Tst\RegressionTests” and “Plang\Tst\SampleProtocols” is identical:

At the first level, called “Feature”, the tests are divided according to the feature tested, in the way that is specified in “TestingFramework.docx”. For example, four feature test subfolders have a number in the name which corresponds to the subsection numbers in the “P Features to Test” section. In addition, there are two more subfoders “Integration” and “Combined” – see explanation in “TestingFramework.docx”.

* Root
  + State Machine Level Declarations (subdirectory called “Feature1SMLevelDecls”)
  + Statements (“Feature2Stmts”)
  + Expressions (“Feature3Exprs”)
  + Data types (“Feature4DataTypes”)
  + Integration Tests (“Integration”)
  + Combined Tests (“Combined”)
  + Liveness tests (“Liveness”)

Each “Feature” folder has the following three subfolders (at the “type of error” level):

* Static Error: Static analysis reports an error (as listed in “P COMPILER” section)
* Dynamic Error: Zinger reports an error
* Correct: Zinger does not report an error

At the next level, subfolders for individual tests are located.

For example, here’s a full path to the test “function”:  
\plang\Tst\RegressionTests\Feature1SMLevelDecls\StaticError\function

Each test subfolder contains “test.p” file and up to three subfolders at the “tool” level: Pc, Zing and Prt.

Pc, Zing and Prt folders are the “leaves” in the directory tree, and each of these contain “testconfig\*.txt” and the acceptor(s). The names of the testconfig\*.txt files for Pc, Zing and Prt are testconfigPc.txt, testconfigZing.txt and testconfigPrt.txt, respectively.

Some subdirectories Pc, Zing and Prt might not be present for some tests. For example, for the StaticError tests, the only subfolder would be Pc. Another example: if a test is only intended for testing Zing (Prt), then there should be Pc and Zing (Pc and Prt) subfolders only. Note that Pc directory is mandatory, since P compiler generates files used by Zinger and runtime.

# Regression process for developers

1. Ideally, each developer should run regression as

“testP.bat RegressionTests.txt”

before pushing the changes.

“SampleProtocols” tests take longer and are optional for now.

To run regression on the SampleProtocols directory, run:

“testP.bat SampleProtocols.txt”

1. 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:**

To reset multiple tests, use

“Plang\Tst> testP.bat <path to the text file with test directories> /reset”

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

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

“CheckP.exe” is faster than “testP.bat”, since it does not rebuild Plang.

More details on running CheckP.exe are below.

1. Run testP.exe again and make sure that all the tests pass.
2. Push the changes AND new acceptors (and testconfig\*.txt, if changed).

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

# 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

# Test tool instruction

“Plang\Tst> testP.bat [<path to the text file with test directories>] [/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.

For now, testP.bat only handles positional parameters, so to reset regression, the first parameter < path to the text file with test directories > must be present.

To run regular and long regressions, use files

Tst\RegressionTests.txt

Tst\SampleProtocols.txt

Please note that in the file with test directories, the names of the directories should either be absolute paths, or paths relative to Tst directory. IN the latter case, the directory name should start with either “.\” or with the name (no backslash). For example:  
Correct directory names:

D:\PLanguage\plang\Tst\RegressionTests

.\RegressionTests

RegressionTests

Incorrect directory name:

\RegressionTests

The testing framework does the following:

1. Runs P tool specified in the configuration file “testconfig\*.txt”: pc.exe, zinger.exe or tester.exe (for runtime).
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: acceptor file “acc\_0.txt” is updated; all other acceptor files are deleted; this is equivalent to calling CheckP with “-add” flag for all tests.

1. If there are failing tests, the tool generates two files to help with debugging:

failed-tests.txt : a list of all tests that failed, to be used for running regression again

on the failing tests only (as the first parameter for testP.bat)

display-diffs.bat : a batch file to run in order to display differences in acceptors for all failing

tests

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 the following “Feature” subfolders:

Feature1SMLevelDecls

Feature2Stmts

Feature3Exprs

Feature4DataTypes

Combined

Integration

Liveness

Zinger

For more information on specific features in each category, look at 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 one, two or three tool subfolders (depending on the test): Pc or Pc + Zing or Pc + Prt or Pc + Prt + Zing. Some examples of tool subfolders for different “error types”:

“error type” tool subfolders

StaticError Pc

DynamicError Pc + Prt + Zing

Correct Pc + Zing

Note that in case of a “Correct” test, the runtime program is non-terminating – hence no Prt folder for such tests.

1. In each tool subfolder, create a file called “testconfig\*.txt”, which will define how to run your test case (where “\*” stands for “Pc”, “Prt” or “Zing”).

In each tool folder, standardized testconfig\*.txt files are used. There are about 9 different templates for testconfig\*.txt. The template applicable to a specific tool folder is completely determined by the location of a specific test T, in particular, by the error type (Correct, DynamicError and StaticError) and the tool that is being run in a specfic subdirectory (Pc, Zc or Prt). Use existing testconfig\*.txt files from the tests withing the same category as the new test.

1. Run “testP.bat” with the text file containing the full path to the directory X with “/reset” flag. Make sure that the result is what you expected.
2. The tools will run in the corresponding tool subfolders, starting with pc.exe, then tester.exe, then zinger.exe. 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 subsection.

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, including the subsection number from TestingFramework.docx of the

specific feature tested, for example: “dsc: 1.7. Transition to undefined state: error”

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”

* the regression tool runs pc.exe (under Pc folder) only once for each test, and zinger and runtime re-use the files generated by pc.exe. That means that it is dangerous to run Zc or Prt without first running Pc. So, when running regression on a particular test, a good practice is to use the test folder (where .p file is located) as a parameter of testP.bat (and not Zc or Prt subfolders).

Running regression on s specific test (faster way)

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\Tool (where Tool is Pc, Zc or Prt), do:

Tests\...\X\Tool> ...\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\Tool> ...\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.