

Structure and Function of the PS-MOPS Testing System

Daniel Chang

July 21, 2006

1 Overview

The PS-MOPS Testing System provides a general framework for testing the operation of the PS-MOPS system. The purpose of the testing system is to provide continual verification of the correct operation of PS-MOPS. In the event of a PS-MOPS failure, the archived history of the testing system can be used to readily determine the cause. The testing system will also be useful when PS-MOPS components are upgraded. The tests will ensure that the system functions correctly with the new components.

The testing system includes both automated and manual operation. For automated operations, a daily build, install and test sequence is performed on the code residing in CVS. Testing results are archived in a database that includes a record of the tests run, their running time, as well as the state of the testing system used to perform the testing. An HTTP interface is provided along with result reporting by email.

Most features of the testing system are configurable through configuration files in CSV format, through definitions in `PS::MOPS::Test::Config`, or through command-line options. The system currently implements unit testing of Perl modules.

1.1 Testing State

As the tests themselves are in a constant state of development, it is essential that the testing module versions, or the testing state, be recorded along with the modules being tested. The testing system resides in CVS thereby maintaining a revision history. In this way, tests can be developed freely without concern for affecting the function of the testing system as previous revisions can be readily retrieved through CVS.

1.2 Testing Environment

The testing system provides a testing environment that includes a PS-MOPS Instance which is in turn served by a database. It does this by providing a unique PS-MOPS Test Instance. Thus all the database operations of PS-MOPS can be run within an isolated testing environment. The database and all the directories associated with a PS-MOPS Instance are created on demand and are automatically removed upon completion of testing. This is done so that tests do not interfere with previous testing situations.

2 Testing Installer

The PS-MOPS Testing System has its own installer, `psmops-testing-installer.pl`. The installer can install the testing system into a user-defined location. For example, running command

```
./testing-bin/psmops-testing-installer.pl \  
--base_lib_path=/usr/local/MOPS_DEV \  
--noninteractive
```

will install the system such that the modules will reside in `/usr/local/MOPS_DEV/lib/perl5`. Currently, the testing installer is designed to be run when the current working directory is the `perl5` directory from CVS.

3 Unit testing

The testing of units verifies the correct operation of the functional units of computer programs. For the PS-MOPS Perl code, these units consist of Perl modules.

3.1 UnitTester

The UnitTester is a Perl module designed to execute a series of tests listed in a predefined configuration file. It implements `Test::Harness::TAP` or 'Test Anything Protocol' and can be used with 'make test' or 'prove'.

3.1.1 Module test mapping

The testing module names need not be identical to their PS-MOPS counterparts. Correspondences between unit testing modules and PS-MOPS modules are defined in `module-test-mapping.csv`. This provides additional flexibility when implementing unit tests.

3.2 Classification of tests

The majority of tests correspond to a single function or method in PS-MOPS. The field **PSMOPS function** in the configuration file indicates whether this is the case for a particular test. The methods that implement these tests are then named

`test_function__modifier1__modifier2()`

where `function` is the name of the function being tested and `modifier1` and `modifier2` are optional. An example of a test that does not correspond to a PS-MOPS function or method would be `make_test_fields__simple()`. These tests are named according to

`function__modifier1__modifier2()`

where `function` is the name of the test and `modifier1` and `modifier2` are optional. Separate fields exist for `function`, `modifier1`, and `modifier2` in the CSV configuration files. Here is an example of a unit test configuration file:

```
PSMOPS function,Module,Test,1st Modifier,2nd Modifier,Description
0,PS::MOPS::DC::Field,make_test_fields,simple,,
1,PS::MOPS::DC::Field,new,,,
1,PS::MOPS::DC::Field,insert,,,
1,PS::MOPS::DC::Field,modcf_retrieve,,,
0,PS::MOPS::DC::Detection,make_test_detections,simple,,
1,PS::MOPS::DC::Detection,addToField,,,
1,PS::MOPS::DC::Detection,modcd_retrieve,,,
```

3.3 Configuration files

Configuration files are in CSV format. They require a header line. Comments can be included. All text on a line following a '#' character will be ignored. Configuration files for unit testing currently reside in

`/usr/local/MOPS_DEV/unit-test-config.`

Configuration files for the testing installer are in

`/usr/local/MOPS_DEV/testing-config.`

The testing system determines which PS-MOPS Perl modules to install from

`/usr/local/MOPS_DEV/unit-test-config/perl-modules-to-install.csv.`

There is a type field in the file that indicates whether a module corresponds to Module::Build or ExtUtils::MakeMaker format.

3.4 Automation

CVS testing is performed in an automated manner. PS-MOPS code is extracted from CVS, along with the testing system. A temporary directory is made in

`/usr/local/MOPS_DEV/psmops-build-tmp.`

Each PS-MOPS component is then built and installed. The testing system is installed by its installer. The UnitTester is then called to perform unit testing. Testing results are written to the database transactionally. Full output is redirected to a log file in `/usr/local/MOPS_DEV/testing-logs.`

3.5 Manual operation

In addition to automated unit testing, the unit tests defined in the PS-MOPS Testing System can be used to test individual modules. This is accomplished through a configuration file containing tests relevant to a particular module. When this file is passed as an argument to `testUnits()` of the UnitTester, tests specific to a single module can be run within a PS-MOPS Testing Environment. The function call would be of the form:

```
testUnits(config_file=>"full path to configuration file");
```

Some example individual module configurations are available in

`/usr/local/MOPS_DEV/unit-test-config.`

A script with a call to test units can be placed in the `t` directory of a module and test results will be reported to the Perl test harness via the UnitTester.

3.5.1 CVS Testing

While CVS testing is done primarily during automated operation of the testing system, it can also be performed manually by running `cvs-tester.pl` with options `--noninteractive`, `--database` or `--email`.

4 Testing Results

Testing results are archived in a database to maintain a history of testing. They can be accessed through the HTTP interface. The `UnitTester` has additional options for writing to `STDOUT` or a file.

4.1 Future development

It would be beneficial to view the code referenced in the testing system directly through the HTTP interface. A dynamic CVS extraction routine along with a source code parser could be implemented to accomplish this.

5 Testing System Design

The testing system is written primarily in Perl employing both object-oriented and procedural methodologies. The source code contains copious commenting. Attempts have been made to integrate more formal documentation via the POD and Doxygen documentation structure. The existing code contains a mixture of these two formats.

The modules of the testing system are located in directories with dash-separated names corresponding to their installation paths. For example, `PS::MOPS::Test::UnitTester` is in directory `PS-MOPS-Test-UnitTester` in CVS. The module source code is located directly under this directory in contrast to the Perl convention of storing source code under multiple levels of subdirectories. Any differences from the typical Perl implementation are accounted for through `Build-master.PL`, a custom `Module::Build` build script that is used for every module in the testing system.

All database operations are transactional using MySQL as the database server. The database structure is defined in `testing-mysql` in CVS. It uses a 1NF schema and contains tables for test runs, unit test results and testing state. All database operations are handled through `PS::MOPS::Test::DataHandler`.

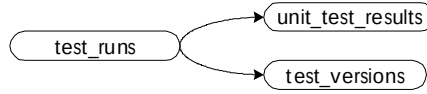


Figure 1: Database structure. Tables `unit_test_results` and `test_versions` are related to `test_runs` by a test run ID in `test_runs`.

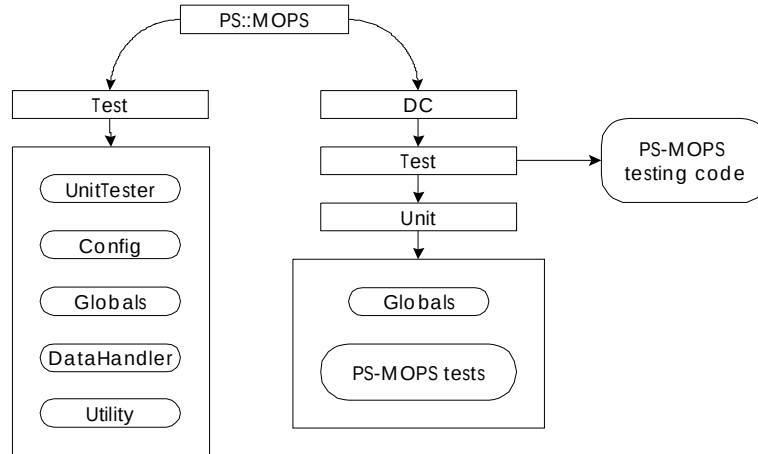


Figure 2: General layout of the PS-MOPS Testing System. Perl modules are indicated by the rounded shapes with paths indicated by the sharp-cornered boxes.

Automation is achieved by running `auto-cvs-install-and-test.sh` as a cron job. Email reports are predefined and are sent at the end of testing if the email option is defined when calling the CVS tester.

5.1 Test development

For use with the `UnitTester`, tests should return a value of 1 for success or a value of 0 for failure. It is intended that the modules implementing these tests will reside under the `Test::Unit` namespace appropriate to the subject of testing. For example, unit tests for `PS::MOPS::DC::Detection` reside in `PS::MOPS::DC::Test::Unit::Detections`. A one-to-one correspondence is not required. Unit tests for many PS-MOPS modules can be placed in

a single test module and be mapped accordingly.

As a convention, the testing code under the Unit path should contain relatively brief code. The primary code for testing exists one level higher under Test. For example, the primary code for testing PS::MOPS::DC::Detection resides in PS::MOPS::DC::Test::Detections. However, testing code is not forced to reside in any particular location.

Configuration constants are in PS::MOPS::Test::Config. Global variables for the testing system as a whole are in PS::MOPS::Test::Globals and those specifically for unit testing are in PS::MOPS::Test::Unit::Globals.

6 Summary

The PS-MOPS Testing System is designed to provide continual verification of the PS-MOPS system. It provides for both automated and manual operation. Testing results are recorded in a MySQL database. Every automated action is fully logged. Any errors occurring in the testing system can be readily tracked. The PS-MOPS Testing System provides an essential service to PS-MOPS. It's potential can be realized through the development of further unit tests. Additionally, it can be extended to support testing needs beyond unit testing such as simulation-level testing.