U.S. DEPARTMENT OF
ENERGY

PNNL-23798

Prepared for the U.S. Department of Energy
under Contract DE-AC05-76RL01830

Drupal Automated Testing: Using Behat and Gherkin

Thomas Williams – Thom.Williams@pnnl.gov
Carolyn Wolkenhauer – Carolyn.Wolkenhauer@pnnl.gov
Michael Madison – Michael.Madison@pnnl.gov

October 2014

Pacific Northwest
NATIONAL LABORATORY

Proudly Operated by Battelle *Since 1965*

DISCLAIMER

PACIFIC NORTHWEST NATIONAL LABORATORY
operated by
BATTELLE
for the
UNITED STATES DEPARTMENT OF ENERGY
under Contract DE-AC05-76RL01830

Printed in the United States of America

Available to DOE and DOE contractors from the
Office of Scientific and Technical Information,
P.O. Box 62, Oak Ridge, TN 37831-0062;
ph: (865) 576-8401
fax: (865) 576-5728
email: reports@adonis.osti.gov

Available to the public from the National Technical Information Service,
U.S. Department of Commerce, 5285 Port Royal Rd., Springfield, VA 22161
ph: (800) 553-6847
fax: (703) 605-6900
email: orders@ntis.fedworld.gov
online ordering: http://www.ntis.gov/ordering.htm

This document was printed on recycled paper.
(9/2003)

Table of Contents

Overview

Drupal is an open source, web based content management system. It is managed and maintained by an international community made up of thousands of users. The Pacific Northwest National Laboratory (PNNL) has a group of experts that develop web applications using the Drupal 7 framework and actively participate in the large Drupal community.

This document describes techniques used by PNNL to perform automated testing in Drupal. Most of the guidance here should be considered to be generic Drupal guidance, and not specific to any individual website(s). Additional resources that may be consulted include:

Drupal FAQ: https://www.drupal.org/drupal-faq

Drupal Term Glossary: https://www.drupal.org/glossary

List of US Government Sites Using Drupal: https://groups.drupal.org/government-sites#USA

PNNL Drupal Team: http://drupal.pnnl.gov

Disclaimer:
Drupal has many different configuration options, settings, and options for administrative users. This guide is specifically focused on automated testing within the Drupal framework. It is strongly recommended that you work closely with a developer from the PNNL team to make any changes to automated tests to ensure they are a part of the development cycle (development, testing, deployment). It is also highly likely that changes to tests made directly to the production environment may be lost on future deployments.

Automated Testing on Drupal sites using Behat and Gherkin

Throughout the development cycle of Drupal sites, more features, content types, and elements are continuously added to the site. As this happens, it becomes readily apparent that the testing efforts need to include a quick yet comprehensive way to run a regression pass of all test cases against all features that come together to make the final production website.

To address this we needed an automated testing tool, with a Drupal compatible framework to store the test cases in and a simple language that didn't require extra effort and time delays to learn. Gherkin in conjunction with Behat presented a useful solution.

1.1 What are Gherkin and Behat - The Language and the Framework:

Gherkin is a business readable (not English only) language designed for behavior testing tools like Behat. It serves two purposes; it provides documentation for your project and helps create automated testing in a human readable language.

Behat is a behavior-driven development framework for PHP. Behavior-driven development starts with human readable sentences that describe your objective, and then implement that objective in software.

More information on Behat and Gherkin can be found here:

- Gherkin Documentation
- Behat Documentation

1.2 How does it work

Behat testing is executed via the command line. This can be done in either a Windows, or a Mac/Linux/Unix environment. Because these tests use a command line browser behind the scenes, JavaScript is not available; JavaScript testing, however, is handled with manual test execution.

When running a Behat test, the test can be selected either by its name or by tags that can represent one or more tests (illustrated below).

```
# Tests run by feature name
behat --profile project --name accesstest
# Tests run by tag name
behat --profile project --tags procedure
# Tests run by both feature name and tag name
behat --profile project --name accesstest --tags procedure
```

Test names are designed to correspond with a testing objective or content type. Behat tests written in Gherkin are named both with a "Feature" and a "Scenario". The Feature is the name of the suite of tests saved in a single file, and a Scenario is a single test. Following is an example:

```
Feature: accesstest

Scenario: Get a result
  When some test code
  Then I get a result

Scenario: Get a different result
  When some more test code
  Then I get a different result
```

In this example, the tests can be run from the command line with the name "accesstest".

Behat tests are saved in files with a ".feature" extension in the "tests/features" directory.

These tests are written in the Gherkin language which describes how a test can be run. For example, a file named access.feature, would contain tests that check who can and cannot access the site based on their role or login level.

When Behat is running a test by name it looks for the text naming the Features and Scenarios. To prevent multiple tests running under the same name, Features for a site, should have unique names. When looking for names or tags, Behat is case sensitive. As a convention we use all lower case in our names and tags.

Tags can also be used to group one or more tests within a feature.

We have developed a system of tags and feature name that allow any set of tests to be run simply: Feature names follow the file name in which they are stored. If the file is Access.feature, the test will be named accesstest. If the file is named Hello_World.feature the test would be named helloworldtest. We removed any separation between words, made everything lowercase, and appended test to the end of the name in an effort to make all Feature names unique.

Tags are marked by the @ symbol. `@tag_name` is a tag called tag_name. We use five primary tags in our tests.

- @1 (numbered tags)
- @setup
- @procedure
- @reset
- @cleanup

Numbered tags consecutively number each scenario in the feature. These tags are primarily used during development to run a single test and make sure that a particular website feature is performing properly.

Setup tags are designed to create content. Each test of a Drupal content type needs to create content used to test that content type or be used as related content in other tests.

Procedure tags are used to run the body of the tests. This tag runs the majority of the tests in a Feature and performs all of the functional testing. It also creates any relationships needed between content types.

Reset tags are used to undo any changes the procedure tag created. If a piece of content is published in the procedure, it will be unpublished in the reset. If it is related to other content in the procedure, reset will clear out that relationship.

Cleanup tags are used to delete content. Run these when done testing to eliminate the content created for the various tests. It will return the test environment to original state, so this is also useful if a test needs to be repeated to confirm results.

Installation

1.3 Copy the files

The version of Behat used at PNNL with any modifications needed to run the tests on your site is included in the zip file provided by PNNL.

In the zip file you will find the "Behat" and "tests" directories. The “Behat” directory contains the Behat source code while the “tests” directory contains all of the tests for the site.

Copy both the “Behat” and “tests” directories up to the server as siblings of the root Drupal folder (master) provided.

1.4 Setup behat.yml

Make a copy of example.behat.yml and name it behat.yml

Bash shell:

```
# Navigate to 'tests' sub-directory of project working copy
cd /path/to/project-repo/tests

# Create local behat.yml if it doesn't exist
cp example.behat.yml behat.yml
```

Windows shell:

```
# Navigate to 'tests' sub-directory of project working copy
cd C:\path\to\project-repo\tests

# Create local behat.yml if it doesn't exist
copy example.behat.yml behat.yml
```

Edit behat.yml

```
project-name:
  extensions:
    Behat\MinkExtension\Extension:
      base_url: 'http://localhost/project-name'
  context:
    parameters:
      http_auth: false
```

Change `project-name` to match the name of your project Change url in `base_url` to match the URL used to reach the project site (be sure to include http:// or https://) If using http:// to access the site make sure `http_auth` is set to `false` If using https:// set `http_auth` to `true` Save behat.yml

Usage

If new terminal session, "activate" Behat (add Behat sub-directory to session PATH) bash shell:

Bash shell:

```
# Navigate to Behat working copy
cd /path/to/behat-repo

# Add Behat sub-directory to session PATH
source activate
```

Windows shell:

```
# Navigate to Behat working copy
cd C:\path\to\behat-repo

# Add Behat sub-directory to session PATH
activate
```

Note: If you are comfortable doing so, you can add the directory behat-repo/behat to your system PATH; then, you will no longer need to "activate" Behat each for each new terminal session.

Navigate to project ‘tests’ directory

Bash shell:

```
# Navigate to 'tests' sub-directory of project working copy
cd /path/to/project-repo/tests
```

Windows shell:

```
# Navigate to 'tests' sub-directory of project working copy
cd C:\path\to\project-repo\tests
```

Run project tests:

```
# Run setup for all the tests
behat --profile project-name --tags setup

# Run procedure for all the tests
behat --profile project-name --tags procedure

# Run cleanup for all the tests
behat --profile project-name --tags cleanup

# Run only tests matching (feature) name ex. 'accesstest'
behat --profile project-name --tags setup
behat --profile project-name --name accesstest --tags procedure
behat --profile project-name --tags cleanup
```

Troubleshooting

Problem: Configuration for profile "project-name" cannot be found.

Solution 1: Make sure you are in the "tests" directory. In order for behat to find your profile you need to be running behat from the same directory as your behat.yml file.

Solution 2: Make sure your behat.yml file is named correctly. If your behat.yml file gets accidentally renamed as anything else (including behat .yml) it will not read the file correctly.

Solution 3: Make sure your behat.yml settings are set correctly:

- Check your url, make sure it contains the correct prefix (http:// vs. https://).
- Make sure your http_auth setting is set correctly.

Problem: Setup tag fails.

Solution 1: This can happen when content already exists. Drupal will add a numbered suffix on the url for content if the content has the same title as another piece of content. If the setup tag is run more than once without deleting the content that is created, the test will fail because it will not find the correct url on file creation.

To fix this you need to remove all of the autotest content and then re-run the setup tag. To delete the autotest content, log into the site as an admin user and go to the "Content Management" page. Select all of the content prefixed with "Autotest" and delete it.

Solution 2: If the server is really busy, connection timeouts can also cause a test to fail. Content is usually created but should be deleted and recreated with a successful setup run. Run the cleanup tag or remove the content using the method in Solution 1 and then run the setup tag again to create the content. If you still have problems with timeouts, try waiting a couple of hours, or try running the tests at a different time of day.

Problem: Cleanup tag fails.

Solution 1: The cleanup scenarios can fail if content is not unrelated before it is deleted. If this happens, before continuing testing, navigate to the site and log in as an admin user. Go to the "Content Management" page, select all of the content prefixed with "Autotest" and delete it.

Solution 2: If the cleanup tag is run multiple times in a row, it will fail because the content it is supposed to remove has already been deleted. Simply verify that all test related content has been removed and you will be good to go.