the Drupal Extension to Behat and Mink Documentation

Release 1.1

Melissa Anderson

Jan 08, 2018

Contents

Contents:

CHAPTER 1

Testing your site with the Drupal Extension to Behat and Mink

The Drupal Extension to Behat and Mink provides Drupal-specific functionality for the Behavior-Driven Development testing frameworks of Behat and Mink.

1.1 What do Behat and Mink Do?

Behat and Mink allow you to describe the behavior of a web site in plain, but stylized language, and then turn that description into an automated test that will visit the site and perform each step you describe. Such functional tests can help site builders ensure that the added value they've created when building a Drupal site continues to behave as expected after any sort of site change – security updates, new module versions, changes to custom code, etc.

1.2 What does the Drupal Extension add?

The Drupal Extension to Behat and Mink assists in the performance of these common Drupal testing tasks:

- Set up test data with Drush or the Drupal API
- Define theme regions and test data appears within them
- Clear the cache, log out, and other useful steps
- Detect and discover steps provided by contributed modules and themes

CHAPTER 2

System Requirements

2.1 Meet the system requirements

1. Check your PHP version:

   ```
   php --version
   ```

   It must be higher than 5.3.5! Note: This means you cannot use the same version of PHP for testing that you might use to run a Drupal 5 site.

   PHP will also need to have the following libraries installed:

   - curl
   - mbstring
   - xml

   Check your current modules by running:

   ```
   php -m
   ```

2. Check for Java:

   ```
   java -version
   ```

   It doesn't necessarily matter what version, but it will be required for Selenium.

3. Directions are written to use command-line cURL. You can make sure it's installed with:

   ```
   curl --version
   ```

4. Selenium

   Download the latest version of Selenium Server It's under the heading Selenium Server (formerly the Selenium RC Server). This is a single file which can be placed any where you like on your system and run with the following command:

```
java -jar selenium-server-standalone-2.44.0.jar &
// replace with the name of the version you downloaded
```

CHAPTER 3

Stand-alone installation

A stand-alone installation is recommended when you want your tests and testing environment to be portable, from local development to CI server, to client infrastructure. It also makes documentation consistent and reliable.

1. Create a folder for your BDD tests:

```
mkdir projectfolder
cd projectfolder
```

   All the commands that follow are written to install from the root of your project folder.

2. Install Composer, a php package manager:

```
curl -s https://getcomposer.org/installer | php
```

3. Create a composer.json file to tell Composer what to install. To do that, paste the following code into your editor and save as composer.json. The Drupal Extension requires Behat, Mink, and the Mink Extension. They will all be set up because they're dependencies of the Drupal Extension, so you don't have to specify them directly in the composer.json file:

```
{
  "require": {
    "drupal/drupal-extension": "^3.2"
  },
  "config": {
    "bin-dir": "bin/"
  }
}
```

4. Run the following command to install the Drupal Extension and all those dependencies. This takes a while before you start to see output:

```
php composer.phar install
```

5. Configure your testing environment by creating a file called behat.yml with the following. Be sure that you point the base_url at the web site YOU intend to test. Do not include a trailing slash:

```
default:
  suites:
    default:
      contexts:
        - FeatureContext
        - Drupal\DrupalExtension\Context\DrupalContext
        - Drupal\DrupalExtension\Context\MinkContext
        - Drupal\DrupalExtension\Context\MessageContext
        - Drupal\DrupalExtension\Context\DrushContext
  extensions:
    Behat\MinkExtension:
      goutte: ~
      selenium2: ~
      base_url: http://seven.l
    Drupal\DrupalExtension:
      blackbox: ~
```

6. Initialize behat. This creates the features folder with some basic things to get you started, including your own FeatureContext.php file:

```
bin/behat --init
```

7. This will generate a FeatureContext.php file that looks like:

```
<?php

use Behat\Behat\Tester\Exception\PendingException;
use Drupal\DrupalExtension\Context\RawDrupalContext;
use Behat\Behat\Context\SnippetAcceptingContext;
use Behat\Gherkin\Node\PyStringNode;
use Behat\Gherkin\Node\TableNode;

/**
 * Defines application features from the specific context.
 */
class FeatureContext extends RawDrupalContext implements␣
↪SnippetAcceptingContext {

  /**
   * Initializes context.
   *
   * Every scenario gets its own context instance.
   * You can also pass arbitrary arguments to the
   * context constructor through behat.yml.
   */
  public function __construct() {
  }
}
```

This FeatureContext.php will be aware of both the Drupal Extension and the Mink Extension, so you'll be able to take advantage of their drivers add your own custom step definitions as well.

8. To ensure everything is set up appropriately, type:

```
bin/behat -dl
```

You'll see a list of steps like the following, but longer, if you've installed everything successfully:

```
default | Given I am an anonymous user
default | Given I am not logged in
default | Given I am logged in as a user with the :role role(s)
default | Given I am logged in as :name
```

CHAPTER 4

System-wide installation

A system-wide installation allows you to maintain a single copy of the testing tool set and use it for multiple test environments. Configuration is slightly more complex than the stand-alone installation but many people prefer the flexibility and ease-of-maintenance this setup provides.

4.1 Overview

To install the Drupal Extension globally:

1. Install Composer
2. Install the Drupal Extension in */opt/drupalextension*
3. Create an alias to the behat binary in */usr/local/bin*
4. Create your test folder

4.2 Install Composer

Composer is a PHP dependency manager that will make sure all the pieces you need get installed. Full directions for global installation and more information can be found on the Composer website.:

```
curl -sS https://getcomposer.org/installer |
php mv composer.phar /usr/local/bin/composer
```

4.3 Install the Drupal Extension

1. Make a directory in /opt (or wherever you choose) for the Drupal Extension:

```
cd /opt/
sudo mkdir drupalextension
cd drupalextension/
```

2. Create a file called *composer.json* and include the following:

```
{
  "require": {
    "drupal/drupal-extension": "^3.2"
  },
  "config": {
    "bin-dir": "bin/"
  }
}
```

3. Run the install command:

```
sudo composer install
```

It will be a bit before you start seeing any output. It will also suggest that you install additional tools, but they're not normally needed so you can safely ignore that message.

4. Test that your install worked by typing the following:

```
bin/behat --help
```

If you were successful, you'll see the help output.

5. Make the binary available system-wide:

```
ln -s /opt/drupalextension/bin/behat /usr/local/bin/behat
```

4.4 Set up tests

1. Create the directory that will hold your tests. There is no technical reason this needs to be inside the Drupal directory at all. It is best to keep them in the same version control repository so that the tests match the version of the site they are written for.

   One clear pattern is to keep them in the sites folder as follows:

   Single site: *sites/default/behat-tests*

   Multi-site or named single site: */sites/my.domain.com/behat-tests*

2. Wherever you make your test folder, inside it create the behat.yml file:

```
default:
  suites:
    default:
      contexts:
        - FeatureContext
        - Drupal\DrupalExtension\Context\DrupalContext
        - Drupal\DrupalExtension\Context\MinkContext
        - Drupal\DrupalExtension\Context\MessageContext
        - Drupal\DrupalExtension\Context\DrushContext
  extensions:
    Behat\MinkExtension:
```

```
      goutte: ~
      selenium2: ~
      base_url: http://seven.l
    Drupal\DrupalExtension:
      blackbox: ~
```

3. Initialize behat. This creates the features folder with some basic things to get you started:

```
bin/behat --init
```

4. This will generate a FeatureContext.php file that looks like:

```
<?php

use Behat\Behat\Tester\Exception\PendingException;
use Drupal\DrupalExtension\Context\RawDrupalContext;
use Behat\Behat\Context\SnippetAcceptingContext;
use Behat\Gherkin\Node\PyStringNode;
use Behat\Gherkin\Node\TableNode;

/**
 * Defines application features from the specific context.
 */
class FeatureContext extends RawDrupalContext implements SnippetAcceptingContext {

  /**
   * Initializes context.
   *
   * Every scenario gets its own context instance.
   * You can also pass arbitrary arguments to the
   * context constructor through behat.yml.
   */
  public function __construct() {
  }
}
```

This will make your FeatureContext.php aware of both the Drupal Extension and the Mink Extension, so you'll be able to take advantage of their drivers and step definitions and add your own custom step definitions here. The FeatureContext.php file must be in the same directory as your behat.yml file otherwise in step 5 you will get the following error:

> [BehatBehatContextExceptionContextNotFoundException] *FeatureContext* context class not found and can not be used.

5. To ensure everything is set up appropriately, type:

```
behat -dl
```

You'll see a list of steps like the following, but longer, if you've installed everything successfully:

```
default | Given I am an anonymous user
default | Given I am not logged in
default | Given I am logged in as a user with the :role role(s)
default | Given I am logged in as :name
```

CHAPTER 5

Environment specific settings

Some of the settings in `behat.yml` are environment specific. For example the base URL may be `http://mysite.localhost` on your local development environment, while on a test server it might be `http://127.0.0.1:8080`. Some other environment specific settings are the Drupal root path and the paths to search for subcontexts.

If you intend to run your tests on different environments these settings should not be committed to `behat.yml`. Instead they should be exported in an environment variable. Before running tests Behat will check the `BEHAT_PARAMS` environment variable and add these settings to the ones that are present in `behat.yml`. This variable should contain a JSON object with your settings.

Example JSON object:

```
{
    "extensions": {
        "Behat\\MinkExtension": {
            "base_url": "http://myproject.localhost"
        },
        "Drupal\\DrupalExtension": {
            "drupal": {
                "drupal_root": "/var/www/myproject"
            }
        }
    }
}
```

To export this into the `BEHAT_PARAMS` environment variable, squash the JSON object into a single line and surround with single quotes:

```
$ export BEHAT_PARAMS='{"extensions":{"Behat\\MinkExtension":{"base_url":"http://
↪myproject.localhost"},"Drupal\\DrupalExtension":{"drupal":{"drupal_root":"/var/www/
↪myproject"}}}}'
```

There is also a Drush extension that can help you generate these environment variables.

CHAPTER 6

Drupal Extension Drivers

The Drupal Extension provides drivers for interacting with your site which are compatible with Drupal 6, 7, and 8. Each driver has its own limitations.

Feature	Blackbox	Drush	Drupal API
Map Regions	Yes	Yes	Yes
Create users	No	Yes	Yes
Create nodes	No	Yes [*]	Yes
Create vocabularies	No	Yes [*]	Yes
Create taxonomy terms	No	Yes [*]	Yes
Run tests and site on different servers	Yes	Yes	No

[*] Requires that the Behat Drush Endpoint be installed on the Drupal site under test.

CHAPTER 7

Blackbox Driver

The blackbox driver assumes no privileged access to the site. You can run the tests on a local or remote server, and all the actions will take place through the site's user interface. This driver was enabled as part of the installation instructions by lines 13 and 14, highlighted below.

```
1  default:
2    suites:
3      default:
4        contexts:
5          - FeatureContext
6          - Drupal\DrupalExtension\Context\DrupalContext
7          - Drupal\DrupalExtension\Context\MinkContext
8    extensions:
9      Behat\MinkExtension:
10       goutte: ~
11       selenium2: ~
12       base_url: http://seven.l
13     Drupal\DrupalExtension:
14       blackbox: ~
```

7.1 Region steps

It may be really important that a block is in the correct region, or you may have a link or button that doesn't have a unique label. The blackbox driver allows you to create a map between a CSS selector and a user-readable region name so you can use steps like the following without having to write any custom PHP:

```
I press "Search" in the "header" region
I fill in "a value" for "a field" in the "content" region
I fill in "a field" with "Stuff" in the "header" region
I click "About us" in the "footer" region
```

7.1.1 Example:

A stock Drupal 7 installation has a footer area identified by the CSS Id "footer". By editing the behat.yml file and adding lines 15 and 16 below:

```
default:
  suites:
    default:
      contexts:
        - FeatureContext
        - Drupal\DrupalExtension\Context\DrupalContext
        - Drupal\DrupalExtension\Context\MinkContext
  extensions:
    Behat\MinkExtension:
      goutte: ~
      selenium2: ~
      base_url: http://seven.l
    Drupal\DrupalExtension:
      blackbox: ~
      region_map:
        footer: "#footer"
```

You can use a step like the following without writing any custom PHP:

```
When I click "About us" in the "footer" region.
```

Using the blackbox driver configured with the regions of your site, you can access the following region-related steps:

Note:

These examples won't work unless you define the appropriate regions in your behat.yml file.

```
Feature: Test DrupalContext
  In order to prove the Drupal context using the blackbox driver is working properly
  As a developer
  I need to use the step definitions of this context

  Scenario: Test the ability to find a heading in a region
    Given I am on the homepage
    When I click "Download & Extend"
    Then I should see the heading "Core" in the "content" region

  Scenario: Clicking content in a region
    Given I am at "download"
    When I click "About Distributions" in the "content" region
    Then I should see "Page status" in the "right sidebar"
    And I should see the link "Drupal News" in the "footer" region

  Scenario: Viewing content in a region
    Given I am on the homepage
    Then I should see "Come for the software, stay for the community" in the "left␣
↪header"

  Scenario: Test ability to find text that should not appear in a region
    Given I am on the homepage
    Then I should not see the text "Proprietary software is cutting edge" in the
↪"left header"
```

```
24
25    Scenario: Submit a form in a region
26      Given I am on the homepage
27      When I fill in "Search Drupal.org" with "Views" in the "right header" region
28      And I press "Search" in the "right header" region
29      Then I should see the text "Search again" in the "right sidebar" region
30
31    Scenario: Check a link should not exist in a region
32      Given I am on the homepage
33      Then I should not see the link "This link should never exist in a default Drupal␣
    ↪install" in the "right header"
34
35    Scenario: Find a button
36      Given I am on the homepage
37      Then I should see the "Search" button
38
39    Scenario: Find a button in a region
40      Given I am on the homepage
41      Then I should see the "Search" button in the "right header"
42
43    Scenario: Find an element in a region
44      Given I am on the homepage
45      Then I should see the "h1" element in the "left header"
46
47    Scenario: Element not in region
48      Given I am on the homepage
49      Then I should not see the "h1" element in the "footer"
50
51    Scenario: Text not in element in region
52      Given I am on the homepage
53      Then I should not see "DotNetNuke" in the "h1" element in the "left header"
54
55    Scenario: Find an element with an attribute in a region
56      Given I am on the homepage
57      Then I should see the "h1" element with the "id" attribute set to "site-name" in␣
    ↪the "left header" region
58
59    Scenario: Find text in an element with an attribute in a region
60      Given I am on the homepage
61      Then I should see "Drupal" in the "h1" element with the "id" attribute set to
    ↪"site-name" in the "left header" region
```

7.2 Message selectors

The Drupal Extension makes use of three selectors for message. If your CSS values are different than the defaults (shown below), you'll need to update your behat.yml file:

```
1  Drupal\DrupalExtension:
2    selectors:
3      message_selector: '.messages'
4      error_message_selector: '.messages.messages-error'
5      success_message_selector: '.messages.messages-status'
```

Message-related steps include:

```
  Scenario: Error messages
   Given I am on "/user"
   When I press "Log in"
   Then I should see the error message "Password field is required"
   And I should not see the error message "Sorry, unrecognized username or password"
   And I should see the following error messages:
   | error messages             |
   | Username field is required |
   | Password field is required |
   And I should not see the following error messages:
   | error messages                                                                 |
   | Sorry, unrecognized username or password                                       |
   | Unable to send e-mail. Contact the site administrator if the problem persists |

 Scenario: Messages
   Given I am on "/user/register"
   When I press "Create new account"
   Then I should see the message "Username field is required"
   But I should not see the message "Registration successful. You are now logged in"
```

7.3 Override text strings

The Drupal Extension relies on default text for certain steps. If you have customized the label visible to users, you can change that text as follows:

```
Drupal\DrupalExtension:
  text:
    log_out: "Sign out"
    log_in: "Sign in"
    password_field: "Enter your password"
    username_field: "Nickname"
```

CHAPTER 8

Drush Driver

Many tests require that a user logs into the site. With the blackbox driver, all user creation and login would have to take place via the user interface, which quickly becomes tedious and time consuming. You can use the Drush driver to add users, reset passwords, and log in by following the steps below, again, without having to write custom PHP. You can also do this with the Drupal API driver. The main advantage of the Drush driver is that it can work when your tests run on a different server than the site being tested.

8.1 Install Drush

See the Drush project page for installation directions.

8.2 Install the Behat Drush Endpoint

The Behat Drush Endpoint is a Drush-based service that the Drush Driver uses in order to create content on the Drupal site being tested. See the Behat Drush Endpoint project page for instructions on how to install it with your Drupal site.

8.3 Point Drush at your Drupal site

8.3.1 Drupal Alias (For local or remote sites)

You'll need ssh-key access to a remote server to use Drush. If Drush and Drush aliases are new to you, see the Drush site for detailed examples

The alias for our example looks like:

```
<?php
$aliases['local'] = array(
  'root' => '/var/www/seven/drupal',
  'uri'  =>  'seven.l'
```

```
);
$aliases['git7site'] = array(
  'uri'  =>  'git7site.devdrupal.org',
  'host' => 'git7site.devdrupal.org'
);
```

8.3.2 Path to Drupal (local sites only)

If you'll only be running drush commands to access a site on the same machine, you can specify the path to your Drupal root:

```
 Drupal\DrupalExtension:
   blackbox: ~
 drush:
   root: /my/path/to/drupal
```

8.4 Enable the Drush driver

In the behat.yml file:

```
default:
  suites:
    default:
      contexts:
        - FeatureContext
        - Drupal\DrupalExtension\Context\DrupalContext
        - Drupal\DrupalExtension\Context\MinkContext
  extensions:
    Behat\MinkExtension:
      goutte: ~
      selenium2: ~
      base_url: http://seven.l
    Drupal\DrupalExtension:
      blackbox: ~
      api_driver: 'drush'
      drush:
        alias: 'local'
      region_map:
        footer: "#footer"
```

Note: Line 15 isn't strictly necessary for the Drush driver, which is the default for the API.

8.5 Calling the Drush driver

Untagged tests use the blackbox driver. To invoke the Drush driver, tag the scenario with @api

```
Feature: Drush alias
  In order to demonstrate the Drush driver
  As a trainer
```

```
  I need to show how to tag scenarios

  Scenario: Untagged scenario uses blackbox driver and fails
    Given I am logged in as a user with the "authenticated user" role
    When I click "My account"
    Then I should see the heading "History"

  @api
  Scenario: Tagged scenario uses Drush driver and succeeds
    Given I am logged in as a user with the "authenticated user" role
    When I click "My account"
    Then I should see the heading "History"
```

If you try to run a test without that tag, it will fail.

8.5.1 Example:

```
Feature: Drush alias
  In order to demonstrate the Drush driver
  As a trainer
  I need to show how to tag scenarios

  Scenario: Untagged scenario uses blackbox driver and fails
    # features/drush.feature:6
    Given I am logged in as a user with the "authenticated user" role
    # FeatureContext::iAmLoggedInWithRole()
      No ability to create users in Drupal\Driver\BlackboxDriver.
      Put `@api` into your feature and add an api driver
      (ex: `api_driver: drupal`) in behat.yml.
    When I click "My account"
    # FeatureContext::iClick()
    Then I should see the heading "History"
    # FeatureContext::assertHeading()

  @api
  Scenario: Tagged scenario uses Drush driver and succeeds
            # features/drush.feature:12
    Given I am logged in as a user with the "authenticated user" role
    # FeatureContext::iAmLoggedInWithRole()
    When I click "My account"
    # FeatureContext::iClick()
```

The Drush driver gives you access to all the blackbox steps, plus those used in each of the following examples:

```
@api
Feature: Drush driver
  In order to show functionality added by the Drush driver
  As a trainer
  I need to use the step definitions it supports

  Scenario: Drush alias
    Given I am logged in as a user with the "authenticated user" role
    When I click "My account"
    Then I should see the heading "History"

  Scenario: Target links within table rows
```

```
    Given I am logged in as a user with the "administrator" role
    When I am at "admin/structure/types"
    And I click "manage fields" in the "Article" row
    Then I should be on "admin/structure/types/manage/article/fields"
    And I should see text matching "Add new field"

  Scenario: Clear cache
    Given the cache has been cleared
    When I am on the homepage
    Then I should get a "200" HTTP response
```

If the Behat Drush Endpoint is installed on the Drupal site being tested, then you will also have access to all of the examples shown for the Drupal API driver.

CHAPTER 9

Drupal API Driver

The Drupal API Driver is the fastest and the most powerful of the three drivers. Its biggest limitation is that the tests must run on the same server as the Drupal site.

9.1 Enable the Drupal API Driver

To enable the Drupal API driver, edit the behat.yml file, change the api_driver to drupal and add the path to the local Drupal installation as shown below:

```
default:
  suites:
    default:
      contexts:
        - FeatureContext
        - Drupal\DrupalExtension\Context\DrupalContext
        - Drupal\DrupalExtension\Context\MinkContext
  extensions:
    Behat\MinkExtension:
      goutte: ~
      selenium2: ~
      base_url: http://seven.l
    Drupal\DrupalExtension:
      blackbox: ~
      api_driver: 'drupal'
      drush:
        alias: 'local'
      drupal:
        drupal_root: '/var/www/seven/drupal'
      region_map:
        footer: "#footer"
```

Note: It's fine to leave the information for the drush driver in the file. It's the api_driver value that declares which

setting will be used for scenarios tagged @api.

Using this driver, you gain the ability to use all the steps in the examples below (and more).

```
@api
  Scenario: Create a node
    Given I am logged in as a user with the "administrator" role
    When I am viewing an "article" content with the title "My article"
    Then I should see the heading "My article"

  Scenario: Run cron
    Given I am logged in as a user with the "administrator" role
    When I run cron
    And am on "admin/reports/dblog"
    Then I should see the link "Cron run completed"

  Scenario: Create many nodes
    Given "page" content:
    | title    |
    | Page one |
    | Page two |
    And "article" content:
    | title          |
    | First article  |
    | Second article |
    And I am logged in as a user with the "administrator" role
    When I go to "admin/content"
    Then I should see "Page one"
    And I should see "Page two"
    And I should see "First article"
    And I should see "Second article"

  Scenario: Create nodes with fields
    Given "article" content:
    | title                     | promote | body             |
    | First article with fields |       1 | PLACEHOLDER BODY |
    When I am on the homepage
    And follow "First article with fields"
    Then I should see the text "PLACEHOLDER BODY"

  Scenario: Create and view a node with fields
    Given I am viewing an "Article" content:
    | title | My article with fields! |
    | body  | A placeholder           |
    Then I should see the heading "My article with fields!"
    And I should see the text "A placeholder"

  Scenario: Create users
    Given users:
    | name     | mail            | status |
    | Joe User | joe@example.com | 1      |
    And I am logged in as a user with the "administrator" role
    When I visit "admin/people"
    Then I should see the link "Joe User"

  Scenario: Login as a user created during this scenario
    Given users:
    | name      | status |
```

```
    | Test user |      1 |
    When I am logged in as "Test user"
    Then I should see the link "Log out"

  Scenario: Create a term
    Given I am logged in as a user with the "administrator" role
    When I am viewing a "tags" term with the name "My tag"
    Then I should see the heading "My tag"

  Scenario: Create many terms
    Given "tags" terms:
    | name    |
    | Tag one |
    | Tag two |
    And I am logged in as a user with the "administrator" role
    When I go to "admin/structure/taxonomy/tags"
    Then I should see "Tag one"
    And I should see "Tag two"

  Scenario: Create nodes with specific authorship
    Given users:
    | name     | mail            | status |
    | Joe User | joe@example.com | 1      |
    And "article" content:
    | title          | author   | body             | promote |
    | Article by Joe | Joe User | PLACEHOLDER BODY | 1       |
    When I am logged in as a user with the "administrator" role
    And I am on the homepage
    And I follow "Article by Joe"
    Then I should see the link "Joe User"

  Scenario: Create an article with multiple term references
    Given "tags" terms:
    | name      |
    | Tag one   |
    | Tag two   |
    | Tag three |
    | Tag four  |
    And "article" content:
    | title             | field_tags                    |
    | My first article  | Tag one                       |
    | My second article | Tag two, Tag three            |
    | My third article  | Tag two, Tag three, Tag four |
```

CHAPTER 10

Contexts

Before Behat 3, each test suite was limited to a single context class. As of Behat 3, it is possible to flexibly structure your code by using multiple contexts in a single test suite.

10.1 Available Contexts

In accordance with this new capability, The Drupal Extension includes the following contexts:

RawDrupalContext A context that provides no step definitions, but all of the necessary functionality for interacting with Drupal, and with the browser via Mink sessions.

DrupalContext Provides step-definitions for creating users, terms, and nodes.

MinkContext Builds on top of the Mink Extension and adds steps specific to regions and forms.

MarkupContext Contains step definitions that deal with low-level markup (such as tags, classes, and attributes).

MessageContext Step-definitions that are specific to Drupal messages that get displayed (notice, warning, and error).

DrushContext Allows steps to directly call drush commands.

10.2 Custom Contexts

You can structure your own code with additional contexts. See Behat's testing features documentation for a detailed discussion of how contexts work.

Important: Every context you want to use in a suite must be declared in the behat.yml file.

10.2.1 Example

In this example, you would have access to:

- pre-written step definitions for users, terms, and nodes (from the `DrupalContext`)
- steps you've implemented in the main `features/bootstrap/FeatureContext.php` file
- steps you've implemented in the `CustomContext` class

You would not have access to the steps from the `MarkupContext`, `MessageContext`, or `DrushContext`, however.

```
default:
  suites:
    default:
      contexts:
        - Drupal\DrupalExtension\Context\DrupalContext
        - FeatureContext
        - CustomContext
```

10.3 Context communication

Since Behat 3 can have many concurrent contexts active, communication between those contexts can be important.

The following will gather any specified contexts before a given scenario is run:

```
<?php

// Snippet to demonstrate context communications.

  /**
   * Gather any contexts needed.
   *
   * @BeforeScenario
   */
  public function gatherContexts(BeforeScenarioScope $scope) {
    $environment = $scope->getEnvironment();

    $this->drupalContext = $environment->getContext(
↪'Drupal\DrupalExtension\Context\DrupalContext');
    $this->minkContext = $environment->getContext(
↪'Drupal\DrupalExtension\Context\MinkContext');
  }
```

10.4 Drupal Extension Hooks

In addition to the hooks provided by Behat, the Drupal Extension provides three additional ways to tag the methods in your `CustomContext` class in order to have them fire before certain events.

1. `@beforeNodeCreate`
2. `@beforeTermCreate`
3. `@beforeUserCreate`

10.4.1 Example

```
use Drupal\DrupalExtension\Hook\Scope\EntityScope;
 ...
 /**
  * Call this function before nodes are created.
  *
  * @beforeNodeCreate
  */
  public function alterNodeObject(EntityScope $scope) {
    $node = $scope->getEntity();
    // Alter node object as needed.
  }
```

CHAPTER 11

Contributed Module Subcontexts

Although not yet a wide-spread practice, the Drupal Extension to Behat and Mink makes it easy for maintainers to include custom step definitions in their contributed projects.

11.1 Discovering SubContexts

In order to use contributed step definitions, define the search path in the behat.yml

// sites/default/behat-tests/behat.yml

```
default:
  suites:
    default:
      contexts:
        - FeatureContext
        - Drupal\DrupalExtension\Context\DrupalContext
        - Drupal\DrupalExtension\Context\MinkContext
      paths:
        - "./path_to_module/features"
  extensions:
    Behat\MinkExtension:
      goutte: ~
      selenium2: ~
      base_url: http://seven.l
    Drupal\DrupalExtension:
      blackbox: ~
      api_driver: 'drupal'
      drush:
        alias: 'local'
      drupal:
        drupal_root: '/var/www/seven/drupal'
      region_map:
        footer: "#footer"
      subcontexts:
```

```
        paths:
          - "/var/www/seven/drupal/sites/all"
```

The Drupal Extension will search recursively within the directory or directories specified to discover and load any file ending in *.behat.inc*. This system, although created with Drupal contrib projects in mind, searches where it's pointed, so you can also use it for your own subcontexts, a strategy you might employ to re-use step definitions particular to your shop or company's development patterns. The *paths* key allows running tests located in features within the *features* directory of a contributed/custom module.

11.2 Disable autoloading

Autoloading can be disabled in the behat.yml file temporarily with the following:

```
default:
  suites:
    default:
      contexts:
        - FeatureContext
        - Drupal\DrupalExtension\Context\DrupalContext
        - Drupal\DrupalExtension\Context\MinkContext
  extensions:
    Behat\MinkExtension:
      goutte: ~
      selenium2: ~
      base_url: http://seven.l
    Drupal\DrupalExtension:
      blackbox: ~
      api_driver: 'drupal'
      drush:
        alias: 'local'
      drupal:
        drupal_root: '/var/www/seven/drupal'
      region_map:
        footer: "#footer"
      subcontexts:
        paths:
          - "/var/www/seven/drupal/sites/all"
        autoload: 0
```

11.3 For Contributors

Behat subcontexts are no longer supported in version 3. The Drupal Extension, however, continues to support saving module-specific contexts in a file ending with *.behat.inc*

Just like functions, preface the filename with the project's machine name to prevent namespace collisions.

```
<?php

/**
 * Contains \FooFoo.
 */

use Behat\Behat\Hook\Scope\BeforeScenarioScope;
```

```
8  use Behat\Behat\Tester\Exception\PendingException;
9  use Drupal\DrupalExtension\Context\DrupalSubContextBase;
10 use Drupal\DrupalExtension\Context\DrupalSubContextInterface;
11
12 /**
13  * Example subcontext.
14  */
15 class FooFoo extends DrupalSubContextBase implements␣
   ↪DrupalSubContextInterface {
16
17   /**
18    * @var \Drupal\DrupalExtension\Context\DrupalContext
19    */
20   protected $drupalContext;
21
22   /**
23    * @var \Drupal\DrupalExtension\Context\MinkContext
24    */
25   protected $minkContext;
26
27   /**
28    * @BeforeScenario
29    */
30   public function gatherContexts(BeforeScenarioScope $scope) {
31     $environment = $scope->getEnvironment();
32
33     $this->drupalContext = $environment->getContext(
   ↪'Drupal\DrupalExtension\Context\DrupalContext');
34     $this->minkContext = $environment->getContext(
   ↪'Drupal\DrupalExtension\Context\MinkContext');
35   }
36
37   /**
38    * @Given I create a(an) :arg1 content type
39    */
40   public function CreateAContentType($arg1) {
41     $this->minkContext->assertAtPath("admin/structure/types/add");
42     $node = [
43       'title' => 'Test content!',
44     ];
45     $this->drupalContext->nodeCreate($node);
46   }
47
48   /**
49    * @Then /^I should have a subcontext definition$/
50    */
51   public function assertSubContextDefinition() {
52     throw new PendingException();
53   }
54
55 }
```