LEARNING drupal

#drupal

Table of Contents

About

Chapter 1: Getting started with drupal

Remarks

Drupal is an open-source content management system built in PHP. Drupal is designed to be flexible and powerful allowing developers to build a wide variety of sites, from blogs and brochure-style sites to complex e-commerce platforms. Through it's community driven modular architecture Drupal is able to provide tools to extend the core functions to help speed development of large and complex projects.

Currently there are two supported versions of Drupal: 7 and 8. Drupal 8 is built on components from the Symfony framework and many other third party libraries to provide modern development structures.

Examples

Installing Drupal with Drush

```
drush dl drupal --drupal-project-rename=example
cd example
drush site-install standard --db-url='mysql://[db_user]:[db_pass]@localhost/[db_name]' --site-
name=Example
```

Installation of Drupal 8 with Drupal Console

Drupal Console

The new CLI for Drupal. A tool to generate boilerplate code, interact with and debug Drupal.

First, we need to install Drupal Console.

Drupal Console is needed not only for this time, but for future installations.

```
# Run this in your terminal to get the latest project version:
curl https://drupalconsole.com/installer -L -o drupal.phar

# Or if you don't have curl:
php -r "readfile('https://drupalconsole.com/installer');" > drupal.phar

# Accessing from anywhere on your system:
mv drupal.phar /usr/local/bin/drupal

# Apply executable permissions on the downloaded file:
chmod +x /usr/local/bin/drupal

# Copy configuration files to user home directory:
drupal init --override

# Check and validate system requirements
```

```
drupal check
```

You may call `drupal list` to see all available commands.

On the next step we'll download Drupal source code

```
drupal site:new
```

Console will prompt you to choose a folder to download Drupal. And on the next step you'll be asked to choose version of Drupal to download. I recommend to select the last one.

So, when Drupal is downloaded you need to install it.

```
drupal site:install
```

After few simple steps your Drupal site will be ready.

With this methodology, a Drupal fresh install take us between 5 to 7 minutes all from the command-line.

Drupal Concepts

Versions

```
Release Date
```

Version	Release Date
8.2.4	December 07, 2016
7.53	December 07, 2016
6.38 (unsupported)	February 24, 2016
5.23 (unsupported)	August 11, 2010

Entity types

In earlier versions of Drupal, the field system was only used on content types. Now, thanks to the Entity API, we can add fields to other things, like comments. Fieldable entities make Drupal eminently flexible. An entity type is a useful abstraction to group together fields. Below are the Entity types in Drupal core:

- Nodes (content)
- Comments
- Files
- Taxonomy terms
- Taxonomy vocabularies

- Users

You can also build new kinds of entity types where the options above don't suit your needs.

Bundles

Bundles are an implementation of an entity type to which fields can be attached. You can consider bundles as subtypes of an entity type. With content nodes (an entity type), for example, you can generate bundles (subtypes) like articles, blog posts, or products. Not all entity types have bundles, however. For example, users do not have separate bundles (subtypes). For the entity types that do allow bundles, you can create as many bundles (subtypes) as you want. Then, using the Field system, you can add different fields to each bundle. Examples include a file download field on Basic Pages and a subtitle field on Articles.

Fields

A field is a reusable piece of content. In technical terms, each field is a primitive data type, with custom validators and widgets for editing and formatters for display. You can read further for a developer's guide to using the Drupal 7 Fields API.

What's important to know as it relates to Entities is that Fields can be added to any of the bundles (or entity types) to help organize their data.

Say, for example, you create a content type with an unstructured text field and use HTML to structure parts of it, like a summary section, or prices. That would make it more difficult, then, to control how these were displayed, or to make connections between different types of related content.

This is where using fields is essential. You could create a summary field of type Long Text as well as price fields of type Decimal.

Entity

An entity would be one instance of a particular entity type such as a comment, taxonomy term or user profile or of a bundle such as a blog post, article or product.

You can use entity_load to load any entity. Note, however, that the core does not provide a save or delete function, but thanks to Entity API module the missing pieces are added (entity_create(), entity_save(), entity_delete(), entity_view() and entity_access()).

Putting this in Object-Oriented Design/Programming terms...

If you come from an OOD/P background and are trying to better understand what these key concepts are, the following suggested mapping might help (albeit not strictly true from a purist's perspective) :-

- An ***entity type*** is a ***base class***
- A ***bundle*** is an ***extended class***
- A ***field*** is a ***class member***, ***property***, ***variable*** or ***field instance*** (depending on your naming preference)

- An ***entity*** is an ***object*** or ***instance*** of a ***base*** or ***extended class***

All these four OOD/P concepts are special in that they are serialisable (stored - e.g. to a database or file). Serialisation takes place via the Entity API.

Read Getting started with drupal online: https://riptutorial.com/drupal/topic/1135/getting-started-with-drupal

Chapter 2: Drupal 8 Entity API

Introduction

The Entity System in Drupal 8 allows developers to easily create custom content types and model data relationships around these types.The extensive API provides support for form generation,data validation,configuration,routing and a lot of other features.

Examples

Create a content entity using Drupal Console

Drupal console brings scaffolding to the Drupal ecosystem and makes it easy to generate a content entity.

In most instances you will find it easier to work with a custom entity within a custom module.

Step 1 : Generate a module

```
vendor/bin/drupal generate:module
```

Follow the prompts and create your custom module.

Step 2: Generate a content entity

```
vendor/bin/drupal generate:entity:content
```

Follow the prompts on your commandline making sure to select the custom module created in the previous step.

Read Drupal 8 Entity API online: https://riptutorial.com/drupal/topic/10681/drupal-8-entity-api

Chapter 3: Drupal cache and performace

Introduction

Cache has been used for the site or system to improve the content delivery fast for the end-users. This topic is created to explore about Drupal inbuilt caching mechanism and provide info how to use it. We need to explore Drupal's inbuilt caching feature with the external contributed modules like Varnish, Memcache, Authcache, File cache etc. those are available to improve the site performance. You can found the best suited example and caching options to use with your site in this topic.

Examples

Enable Drupal site and block cache

Drupal itself provide good caching options to increase the page speed and serve pages fast to end users. Caches are used to improve the performance of your Drupal site. But it also has a drawback that sometimes it could lead the "stale" data. This means, sometimes, the system may start to serve the old pages from the cache.

How to enable Drupal site and block cache on various Drupal versions? See below for answers:
Drupal 6:

1. Go to Administer -> Site Configuration -> Performance.
2. Turn on cache options
3. Enable the JS/CSS files aggregation and save it.

Drupal 7:

1. Go to Administer -> Config -> Development -> Performance.
2. Turn on cache options
3. Enable the JS/CSS files aggregation and save it.

Read Drupal cache and performace online: https://riptutorial.com/drupal/topic/10082/drupal-cache-and-performace

Chapter 4: Drush

Remarks

What is Drush?

Drush is a command-line scripting interface for Drupal sites. It allows for command-line management of Drupal sites.

Examples

Drush commands

Drush status

```
drush status
```

This will give you an overview of your Drupal site. Version, URI, database location, file paths, default theme etc. If you use this command and you do not see this information, it means you are in a wrong folder and Drush does not know which Drupal site you are referring to.

Resetting password for any user

```
drush upwd admin --password="newpassword"
```

Where "admin" is an existing username and "newpassword" is the desired password.

Generate a one time use admin login URL

```
drush uli
```

Generates a URL which can be used to login into the admin section. The URL has a one time use token. The result should look like this:

```
http://example.com/user/reset/1/1469178712/MEn1QOXo3YGKAUHCknFQF0rEPJ_itkS-a6I8LJwaNYs/login
```

Sometimes the hostname or IP is not resolvable, and the result is a warning like this:

```
default does not appear to be a resolvable hostname or IP, not starting browser.   [warning]
```

```
You may need to use the --uri option in your command or site alias to indicate
the correct URL of this site.
http://default/user/reset/1/1469178629/-zFS_0u8is2N2uCKuLUdGBpJ3cZzV9am5_irsbtVAOs/login
```

The solution is using the "--url" parameter like the following example:

```
drush uli --uri="http://example.com/"
```

Clearing the caches

```
drush cache-rebuild
```

Rebuilds the cache for Drupal 8. For drupal 7, you can use

```
drush cache-clear
```

These commands are also available shorter with

```
drush cr
```

or

```
drush cc // optionally pass all to clear all the caches
```

Enable modules

```
drush pm-enable mymodule
```

Enable 'mymodule' like activating a module in the admin interface

These commands are also available shorter with

```
drush en mymodule // optionally pass the -y option to avoid the interactive question
```

Mainteneace

To enable the maintenance mode using Drush you can use this command:

```
drush vset maintenance_mode 1 // pass 0 to disable the maintenance
```

Remember to clear the caches after you have enabled / disabled the maintenance mode.

Export Configuration

In Drupal 7 and lower, your configuration is probably stored using the Features module. To update a feature with changes from the databases use this command:

```
drush features-update [feature-name] // e.g. drush features-update content_type_news
```

You can also use this shorthand:

```
drush fu [feature-name]
```

Drupal 8 using "Configuration Management". To export your configuration with drush use this command

```
drush config-export // optionally add -y to not have to verify it
```

You can also use the shorthand command

```
drush cex -y
```

Install Drush

Manual Global Installation

For OS X and Linux:

1. Bring up Terminal, Bash, or your normal shell.
2. Type the following into Terminal/Bash:

```
# Download latest stable release using the code below or browse to github.com/drush-
ops/drush/releases.
php -r "readfile('http://files.drush.org/drush.phar');" > drush
# Or use our upcoming release: php -r "readfile('http://files.drush.org/drush-
unstable.phar');" > drush

# Test your install.
php drush core-status

# Make `drush` executable as a command from anywhere. Destination can be anywhere on
$PATH.
chmod +x drush
sudo mv drush /usr/local/bin

# Optional. Enrich the bash startup file with completion and aliases.
drush init
```

For Windows:

1. Download Drush from GitHub.
2. Extract the compressed file in your desired drive, for eg.

   ```
   C:\
   ```

3. Install Drush, define the extracted folder path in the environment path variable which should also include the `Apache, PHP and MySQL`.

   ```
   C:\xampp\apache\bin;C:\xampp\mysql\bin;C:\xampp\php;C:\drush;
   ```

4. Verify that Drush works:

   ```
   drush status
   ```

Composer Global Installation

1. Install Composer globally.
2. Add Composer's `bin` directory to the system path by placing `export PATH="$HOME/.composer/vendor/bin:$PATH"` into your ~/.bash_profile (OS X) or ~/.bashrc (Linux).
3. Install Drush:

   ```
   composer global require drush/drush
   ```

4. Verify that Drush works:

   ```
   drush status
   ```

More details from Drush Docs

Install Drush

Manual installation

Type below commands in Terminal.

```
php -r "readfile('http://files.drush.org/drush.phar');" > drush
chmod +x drush
sudo mv drush /usr/local/bin
drush init # Add alias in bash startup file.
```

Composer

Assuming composer is installed.

```
composer global require drush/drush:dev-master
```

Read Drush online: https://riptutorial.com/drupal/topic/2062/drush

Chapter 5: Example for Drupal 8 Queue API and the Batch API

Examples

An example module to help understanding the Queue API and the Batch API in Drupal 8

xml_import_example.info.yml

```
type: module
name: XML import example
package: Examples
description: "This module helps understanding the Batch API and Queue API with an XML import
example"
core: 8.x
```

xml_import_example.permissions.yml

```
import content from xml:
  title: 'Import content from xml'
  description: 'With this permission user can import contents from a XML source'
  restrict access: TRUE
```

xml_import_example.routing.yml

```
# Get contents from the xml source
xml_import_example.get_contents_from_xml:
  path: '/get-contents-from-xml'
  defaults: { _controller:
'\Drupal\xml_import_example\Controller\ImportContentFromXML::getContentsFromXMLPage' }
  requirements:
    _permission: 'import content from xml'
# Process all queue items with batch
xml_import_example.process_all_queue_items_with_batch:
  path: '/process-all-queue-items'
  defaults: { _controller:
'\Drupal\xml_import_example\Controller\ImportContentFromXML::processAllQueueItemsWithBatch' }
  requirements:
    _permission: 'import content from xml'
```

src/Controller/ImportContentFromXML.php

```
<?php
/**
 * @file
 * Contains \Drupal\xml_import_example\Controller\ImportContentFromXML.
 */

namespace Drupal\xml_import_example\Controller;
```

```
use Symfony\Component\DependencyInjection\ContainerInterface;
use Drupal\Core\Controller\ControllerBase;
use Drupal\Core\Queue\QueueWorkerManager;
use Drupal\Core\Queue\QueueFactory;

/**
 * You can use this constant to set how many queued items
 * you want to be processed in one batch operation
 */
define("IMPORT_XML_BATCH_SIZE", 1);

class ImportContentFromXML extends ControllerBase {

  /**
   * We add QueueFactory and QueueWorkerManager services with the Dependency Injection
solution
   */

  /**
   * @var QueueFactory
   */
  protected $queueFactory;

  /**
   * @var QueueWorkerManager
   */
  protected $queueManager;

  /**
   * {@inheritdoc}
   */
  public function __construct(QueueFactory $queue_factory, QueueWorkerManager $queue_manager)
{
    $this->queue_factory = $queue_factory;
    $this->queue_manager = $queue_manager;
  }

  /**
   * {@inheritdoc}
   */
  public static function create(ContainerInterface $container) {
    $queue_factory = $container->get('queue');
    $queue_manager = $container->get('plugin.manager.queue_worker');

    return new static($queue_factory, $queue_manager);
  }

  /**
   * Get XML from the API and convert it to
   */
  protected function getContentsFromXML() {
    // Here you should get the XML content and convert it to an array of content arrays for
example
    // I use now an example array of contents:
    $contents = array();

    for ($i = 1; $i <= 20; $i++) {
      $contents[] = array(
        'title' => 'Test title ' . $i,
        'body' => 'Test body ' . $i,
      );
    }

    // Return with the contents
    return $contents;
  }

  /**
   * Page where the xml source is preprocessed
   */
  public function getContentsFromXMLPage() {
    // Get contents array
    $contents = $this->getContentsFromXML();

    foreach ($contents as $content) {
      // Get the queue implementation for import_content_from_xml queue
      $queue = $this->queue_factory->get('import_content_from_xml');

      // Create new queue item
      $item = new \stdClass();
      $item->data = $content;
      $queue->createItem($item);
    }

    return array(
      '#type' => 'markup',
      '#markup' => $this->t('@count queue items are created.', array('@count' =>
count($contents))),
    );
  }

  /**
   * Process all queue items with batch
   */
  public function processAllQueueItemsWithBatch() {

    // Create batch which collects all the specified queue items and process them one after
another
    $batch = array(
      'title' => $this->t("Process all XML Import queues with batch"),
      'operations' => array(),
      'finished' =>
'Drupal\xml_import_example\Controller\ImportContentFromXML::batchFinished',
    );

    // Get the queue implementation for import_content_from_xml queue
    $queue_factory = \Drupal::service('queue');
    $queue = $queue_factory->get('import_content_from_xml');

    // Count number of the items in this queue, and create enough batch operations
    for($i = 0; $i < ceil($queue->numberOfItems() / IMPORT_XML_BATCH_SIZE); $i++) {
      // Create batch operations
      $batch['operations'][] =
array('Drupal\xml_import_example\Controller\ImportContentFromXML::batchProcess', array());
    }

    // Adds the batch sets
    batch_set($batch);
    // Process the batch and after redirect to the frontpage
    return batch_process('<front>');
  }
  /**
   * Common batch processing callback for all operations.
   */
  public static function batchProcess(&$context) {

    // We can't use here the Dependency Injection solution
    // so we load the necessary services in the other way
    $queue_factory = \Drupal::service('queue');
    $queue_manager = \Drupal::service('plugin.manager.queue_worker');

    // Get the queue implementation for import_content_from_xml queue
    $queue = $queue_factory->get('import_content_from_xml');
    // Get the queue worker
    $queue_worker = $queue_manager->createInstance('import_content_from_xml');

    // Get the number of items
    $number_of_queue = ($queue->numberOfItems() < IMPORT_XML_BATCH_SIZE) ? $queue-
>numberOfItems() : IMPORT_XML_BATCH_SIZE;

    // Repeat $number_of_queue times
    for ($i = 0; $i < $number_of_queue; $i++) {
      // Get a queued item
      if ($item = $queue->claimItem()) {
        try {
          // Process it
          $queue_worker->processItem($item->data);
          // If everything was correct, delete the processed item from the queue
          $queue->deleteItem($item);
        }
        catch (SuspendQueueException $e) {
          // If there was an Exception trown because of an error
          // Releases the item that the worker could not process.
          // Another worker can come and process it
          $queue->releaseItem($item);
          break;
        }
      }
    }
  }

  /**
   * Batch finished callback.
   */
  public static function batchFinished($success, $results, $operations) {
    if ($success) {
     drupal_set_message(t("The contents are successfully imported from the XML source."));
    }
    else {
      $error_operation = reset($operations);
      drupal_set_message(t('An error occurred while processing @operation with arguments :
@args', array('@operation' => $error_operation[0], '@args' => print_r($error_operation[0],
TRUE))));
    }
  }
}
```

src/Plugin/QueueWorker/ImportContentFromXMLQueueBase.php

```
<?php
```

```
/**
 * @file
 * Contains Drupal\xml_import_example\Plugin\QueueWorker\ImportContentFromXMLQueueBase
 */

namespace Drupal\xml_import_example\Plugin\QueueWorker;

use Drupal\Core\Plugin\ContainerFactoryPluginInterface;
use Drupal\Core\Queue\QueueWorkerBase;
use Drupal\Core\Queue\SuspendQueueException;
use Symfony\Component\DependencyInjection\ContainerInterface;
use Drupal\node\Entity\Node;

/**
 * Provides base functionality for the Import Content From XML Queue Workers.
 */
abstract class ImportContentFromXMLQueueBase extends QueueWorkerBase implements
ContainerFactoryPluginInterface {

  // Here we don't use the Dependency Injection,
  // but the create method and __construct method are necessary to implement

  /**
   * {@inheritdoc}
   */
  public function __construct() {}

  /**
   * {@inheritdoc}
   */
  public static function create(ContainerInterface $container, array $configuration,
$plugin_id, $plugin_definition) {
    return new static();
  }

  /**
   * {@inheritdoc}
   */
  public function processItem($item) {
    // Get the content array
    $content = $item->data;
    // Create node from the array
    $this->createContent($content);
  }

  /**
   * Create content
   *
   * @return int
   */
  protected function createContent($content) {
    // Create node object from the $content array
    $node = Node::create(array(
      'type'  => 'page',
      'title' => $content['title'],
      'body'  => array(
        'value'  => $content['body'],
        'format' => 'basic_html',
      ),
    ));
    $node->save();
  }
}
```

src/Plugin/QueueWorker/ImportContentFromXMLQueue.php

```
<?php

namespace Drupal\xml_import_example\Plugin\QueueWorker;

/**
 * Create node object from the imported XML content
 *
 * @QueueWorker(
 *   id = "import_content_from_xml",
 *   title = @Translation("Import Content From XML"),
 *   cron = {"time" = 60}
 * )
 */
class ImportContentFromXMLQueue extends ImportContentFromXMLQueueBase {}
```

So this is the working module, you can test it in you site.

If you visit the **/get-contents-from-xml** URL 20 queue items are made from a contents array.

The **src/Plugin/QueueWorker/ImportContentFromXMLQueue.php** contains this annotation: **cron = {"time" = 60}**

So if you run cron, the queue items are processed for maximum 60 seconds. You can increase or decrease this time, with that annotation.

If you remove the **cron = {"time" = 60}** line, cron do nothing with your queue items.

If you would like to process all the queue items in your browser, you have to visit the following url: **/process-all-queue-items**

It will collect all of your queue items, creates batch operations from them, and after that it process one after another.

Read Example for Drupal 8 Queue API and the Batch API online:
https://riptutorial.com/drupal/topic/3786/example-for-drupal-8-queue-api-and-the-batch-api

Chapter 6: Field Formatter

Introduction

A field formatter specifies the way a field is rendered in Drupal templates. The formatter used by each field can be configured in the *Manage display* tab associated with the entity type that you are configuring.

Fields can have different formatters depending on the view mode that is being displayed which allows you to control how a field is rendered in different parts of your website.

Remarks

Some things are important to consider when implementing a Field Formatter.

The implementation of your formatter must be inside your module in the folder `src/Plugin/Field/FieldFormatter`. The annotations are also critical as they identify your module and which field types it's applicable to.

In this example, this formatter is applicable only to fields of the type `email`. You can apply your formatter to a number of fields if necessary. If your formatter would, for whatever reason, be applicable to email and date fields:

```
field_type = {
  "email",
  "date",
}
```

One pitfall I've faced when first implementing field formatters with settings is that the settings weren't saved when changed. There is no explicit save method and the solution is to implement the `defaultSettings()` method and specify the field names that make up your configuration form. Also don't forget to set the `#default_value` in the `settingsForm` method.

If you want to have a specific TWIG template for your formatter it's as simple as configuring a `#theme` key while building the render array in the `viewElements` method then in your `.module` file implement `hook_theme`

```
function obfuscator_field_formatter_theme() {
  return [
    'obfuscator_field_formatter' => [
      'variables' => array('title' => NULL, 'url' => NULL),
      'template' => 'obfuscator-field-formatter'
    ],
  ];
}
```

Then create the `templates` folder in the root of your module and have a file named `obfuscator-field-formatter.twig.html` where you output the markup you need. In this example the variables the from the render `#title` and `#url` will be available.

Examples

Obfuscated Email Formatter

In our example we will be creating a customized formatter for email addresses that will allow us to display obfuscated email addresses to fool those nasty spammers.

The formatter will have a some configuration options that will allow us to control how the email address is obfuscated:

- Remove @ and . and replace them with a space,
- Replace @ by at and . by dot,

Please note that this is just an academic example to show how field formatters are displayed and configured and is obviously not very useful for actual anti-spamming.

This example assumes that you have a module named `obfuscator_field_formatter` properly configured and activated.

```
namespace Drupal\obfuscator_field_formatter\Plugin\Field\FieldFormatter;

use Drupal\Core\Field\FieldDefinitionInterface;
use Drupal\Core\Field\FieldItemInterface;
use Drupal\Core\Field\FieldItemListInterface;
use Drupal\Core\Field\FormatterBase;
use Drupal\Core\Form\FormStateInterface;

/**
 * Plugin implementation of the 'example_field_formatter' formatter.
 *
 * @FieldFormatter(
 *   id = "email_obfuscator_field_formatter",
 *   label = @Translation("Obfuscated Email"),
 *   field_types = {
 *     "email"
 *   }
 * )
 */
class ObfuscatorFieldFormatter extends FormatterBase {
  private $options = [];

  public function __construct($plugin_id, $plugin_definition, FieldDefinitionInterface
$field_definition, array $settings, $label, $view_mode, array $third_party_settings) {
    parent::__construct($plugin_id, $plugin_definition, $field_definition, $settings, $label,
$view_mode, $third_party_settings);

    $this->options = [
      'remove_chars' => $this->t('Remove @ and . and replace them with a space'),
      'replace_chars' => $this->t('Replace @ by at and . by dot'),
    ];
  }

  public static function defaultSettings() {
    return [
      'obfuscator_formatter_type' => '',
    ] + parent::defaultSettings();
  }

  public function settingsForm(array $form, FormStateInterface $form_state) {
    return [
      'obfuscator_formatter_type' => [
        '#type' => 'select',
        '#title' => $this->t('Obfuscation Type'),
        '#options' => $this->options,
        '#default_value' => $this->getSetting('obfuscator_formatter_type'),
      ]
    ] + parent::settingsForm($form, $form_state);
  }

  public function settingsSummary() {
    $summary = parent::settingsSummary();
    $type = $this->getSetting('obfuscator_formatter_type');

    if(!is_null($type) && !empty($type)) {
      $summary[] = $this->t('Obfuscation: @value', ['@value' => $this->options[$type]]);
    }

    return $summary;
  }

  public function viewElements(FieldItemListInterface $items, $langcode) {
    $elements = [];

    foreach ($items as $delta => $item) {
      $elements[$delta] = [
        '#type' => 'inline_template',
        '#template' => '{{ value|nl2br }}',
        '#context' => ['value' => $this->viewValue($item)],
      ];
    }

    return $elements;
  }

  protected function viewValue(FieldItemInterface $item) {
    $obfuscated = $item->value;
    $type = $this->getSetting('obfuscator_formatter_type');

    switch($type) {
      case 'remove_chars': {
        $obfuscated = str_ireplace(['@', '.'], ' ', $item->value);
        break;
      }

      case 'replace_chars': {
        $obfuscated = str_ireplace(['@', '.'], [' AT ', ' DOT '], $item->value);
        break;
      }
    }

    return $obfuscated;
  }
}
```

Read Field Formatter online: https://riptutorial.com/drupal/topic/8792/field-formatter

Chapter 7: Module development - Drupal 7

Remarks

Examples for developers module should be used as a reference for module development ideally. It has explanation of all the major APIs, well documented usage. It is all in for begineers to understand module development.

Examples

Basic module providing a simple page

really_neat.info

```
name = Really Neat Module
description = Provides a really neat page for your site
core = 7.x
```

really_neat.module

```
<?php

/**
 * @file
 * Hook implementation and shared functions for the Really Neat Module.
 */

/**
 * Implements hook_menu().
 */
function really_neat_menu() {
  $items = array();

  $items ['really/neat'] = array(
      'title' => 'A Really Neat Page',
      'page_callback' => 'really_neat_page',
      'access_callback' => TRUE, //Anyone can access.
      // Or replace with array([name-of-permission]),
    ),

  return $items;
}

/**
 * Page callback: Displays something really neat
 */
function really_neat_page() {
  return "Really Neat!"
}
```

Basic module providing a custom block

custom_module.info

```
name = Custom Module
description = Creates a block containing a custom output.
core = 7.x
```

custom_module.module

```
/**
 * Initiates hook_block_info.
 *
 * Registers the block with Drupal.
 */
function custom_module_block_info() {
  $blocks = array();
    //Registers the machine name of the block.
  $blocks['custom_block'] = array(
      //Sets the human readable, administration name.
    'info' => t('My Custom Block'),
      //Tells Drupal not to cache this block.
      //Used if there is dynamic content.
    'cache' => DRUPAL_NO_CACHE,
  );
  return $blocks;
}

/**
 * Initiates hook_block_view().
 *
 * Sets the block title and content callback.
 */
function custom_module_block_view($delta = '') {
  $block = array();

  switch ($delta) {
      //Must be the machine name defined in the hook_block_info.
    case 'custom_block':
        //The blocks title.
      $block['subject'] = 'My custom block';
        //The string or function that will provide the content of the block.
      $block['content'] = custom_module_block_content();
      break;
  }

  return $block;
}

/**
 * Returns the content of the custom block.
 */
function custom_module_block_content() {
  $content = "This function only returns a string, but could do anything."

  return $content;
}
```

Basic custom form for inclusion in either page or block examples.

Simple form, validation and submission functions to create a "mailing list" feature. This can then be applied to either the basic page or basic block examples.

Assumes you have created a table in the drupal database called 'mailing_list' with the fields first name, last name and email address.

Additional information on the Form API and additional field options:
https://api.drupal.org/api/drupal/developer!topics!forms_api_reference.html/7.x/

```
function custom_module_form($form, &$form_state) {
  $form['first_name'] = array (
    '#type' => 'textfield',
    '#title' => 'First Name',
    '#required' => TRUE,
  );
  $form['last_name'] = array (
    '#type' => 'textfield',
    '#title' => 'Last Name',
    '#required' => TRUE,
  );
  $form['email'] = array (
    '#type' => 'textfield',
    '#title' => 'First Name',
    '#required' => TRUE,
  );

  return $form;
}

function custom_module_form_validate($form, &$form_state) {
  if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
    form_set_error('email', t('Please provide a valid email address.'));
  }
}

function custom_module_form_submit($form, &$form_state) {
  //Useful function for just getting the submitted form values
  form_state_values_clean($form_state);

  //Save time later by assigning the form values to variables.
  $first_name = $form_state['values']['first_name'];
  $last_name = $form_state['values']['last_name'];
  $email = $form_state['values']['email'];

  //Insert the submitted data to the mailing_list database table.
  db_insert('mailing_list')
    ->fields(array(
      'first name' => $first_name,
      'last name' => $last_name,
      'email' => $email,
    ))
    ->execute();
  //Set a thank you message.
  drupal_set_message('Thank you for subscribing to our mailing list!');

  //drupal_goto() could be used here to redirect to another page or omitted to reload the same
page.
  //If used, drupal_goto() must come AFTER drupal_set_message() for the message to be
displayed on the new page.
}
```

Basic module providing a custom block

custom_module.info

```
name = Custom Module
description = Creates a block containing a custom output.
core = 7.x
```

custom_module.module

```
/**
 * Initiates hook_block_info.
 *
 * Registers the block with Drupal.
 */
function custom_module_block_info() {
  $blocks = array();
    //Registers the machine name of the block.
  $blocks['custom_block'] = array(
      //Sets the human readable, administration name.
    'info' => t('Titania Price Widget'),
      //Tells Drupal not to cache this block.
      //Used if there is dynamic content.
    'cache' => DRUPAL_NO_CACHE,
  );
  return $blocks;
}

/**
 * Initiates hook_block_view().
 *
 * Sets the block title and content callback.
 */
function custom_module_block_view($delta = '') {
  $block = array();

  switch ($delta) {
      //Must be the machine name defined in the hook_block_info.
    case 'custom_block':
        //The blocks title.
      $block['subject'] = 'My custom block';
        //The string or function that will provide the content of the block.
      $block['content'] = custom_module_block_content();
      break;
  }

  return $block;
}

/**
 * Returns the content of the custom block.
 */
function custom_module_block_content() {
  $content = "This function only returns a string, but could do anything."

  return $content;
}
```

Basic custom form for inclusion in either page or block examples.

Simple form, validation and submission functions to create a "mailing list" feature. This can then be applied to either the basic page or basic block examples.

Assumes you have created a table in the drupal database called 'mailing_list' with the fields first name, last name and email address.

Additional information on the Form API and additional field options:
https://api.drupal.org/api/drupal/developer!topics!forms_api_reference.html/7.x/

```
function custom_module_form($form, &$form_state) {
  $form['first_name'] = array (
    '#type' => 'textfield',
    '#title' => 'First Name',
    '#required' => TRUE,
  );
  $form['last_name'] = array (
    '#type' => 'textfield',
    '#title' => 'Last Name',
    '#required' => TRUE,
  );
  $form['email'] = array (
    '#type' => 'textfield',
    '#title' => 'First Name',
    '#required' => TRUE,
  );

  return $form;
}

function custom_module_form_validate($form, &$form_state) {
  if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
    form_set_error('email', t('Please provide a valid email address.'));
  }
}

function custom_module_form_submit($form, &$form_state) {
  //Useful function for just getting the submitted form values
  form_state_values_clean($form_state);

  //Save time later by assigning the form values to variables.
  $first_name = $form_state['values']['first_name'];
  $last_name = $form_state['values']['last_name'];
  $email = $form_state['values']['email'];

  //Insert the submitted data to the mailing_list database table.
  db_insert('mailing_list')
    ->fields(array(
      'first name' => $first_name,
      'last name' => $last_name,
      'email' => $email,
    ))
    ->execute();
  //Set a thank you message.
  drupal_set_message('Thank you for subscribing to our mailing list!');

  //drupal_goto() could be used here to redirect to another page or omitted to reload the same
page.
  //If used, drupal_goto() must come AFTER drupal_set_message() for the message to be
displayed on the new page.
}
```

Example custom_module.install file for creating a database table

*Can be used in conjunction with the **custom form example** to create a table in the drupal database for a Mailing List feature.*

This example was made by creating the table directly in my development database, then created the data for hook_schema() using the Schema module.

This allows for automatic table creation during module install on staging and production sites.

custom_module.install

```
/**
 * Installs the database schema.
 */
function custom_module_install() {
  drupal_install_schema('mailing_list');
}

/**
 * Uninstalls the database schema.
 */
function custom_module_uninstall() {
  drupal_uninstall_schema('mailing_list');
}

/**
* Creates the tables using the schema API.
*/
function custom_module_schema() {
  $schema['mailing_list'] = array(
    'description' => 'TODO: please describe this table!',
    'fields' => array(
      'first name' => array(
        'description' => 'TODO: please describe this field!',
        'type' => 'int',
        'not null' => TRUE,
      ),
      'last name' => array(
        'description' => 'TODO: please describe this field!',
        'type' => 'int',
        'not null' => TRUE,
      ),
      'email' => array(
        'description' => 'TODO: please describe this field!',
        'type' => 'int',
        'not null' => TRUE,
      ),
    ),
  );
}
```

Read Module development - Drupal 7 online: https://riptutorial.com/drupal/topic/2456/module-development---drupal-7

Chapter 8: The Rules module

Introduction

The Rules module is an engine which allows site administrators to automate actions to be conditionally executed, either programmatically or in response to predetermined events.

Rules can react to **Rules Events** occurring on a Drupal site, such as a user logging in. And it can perform customized follow-up **Rules Actions**, such as redirecting to a certain page, which are to be conditionally executed if some **Rules Conditions** are satisfied.

Remarks

Resources

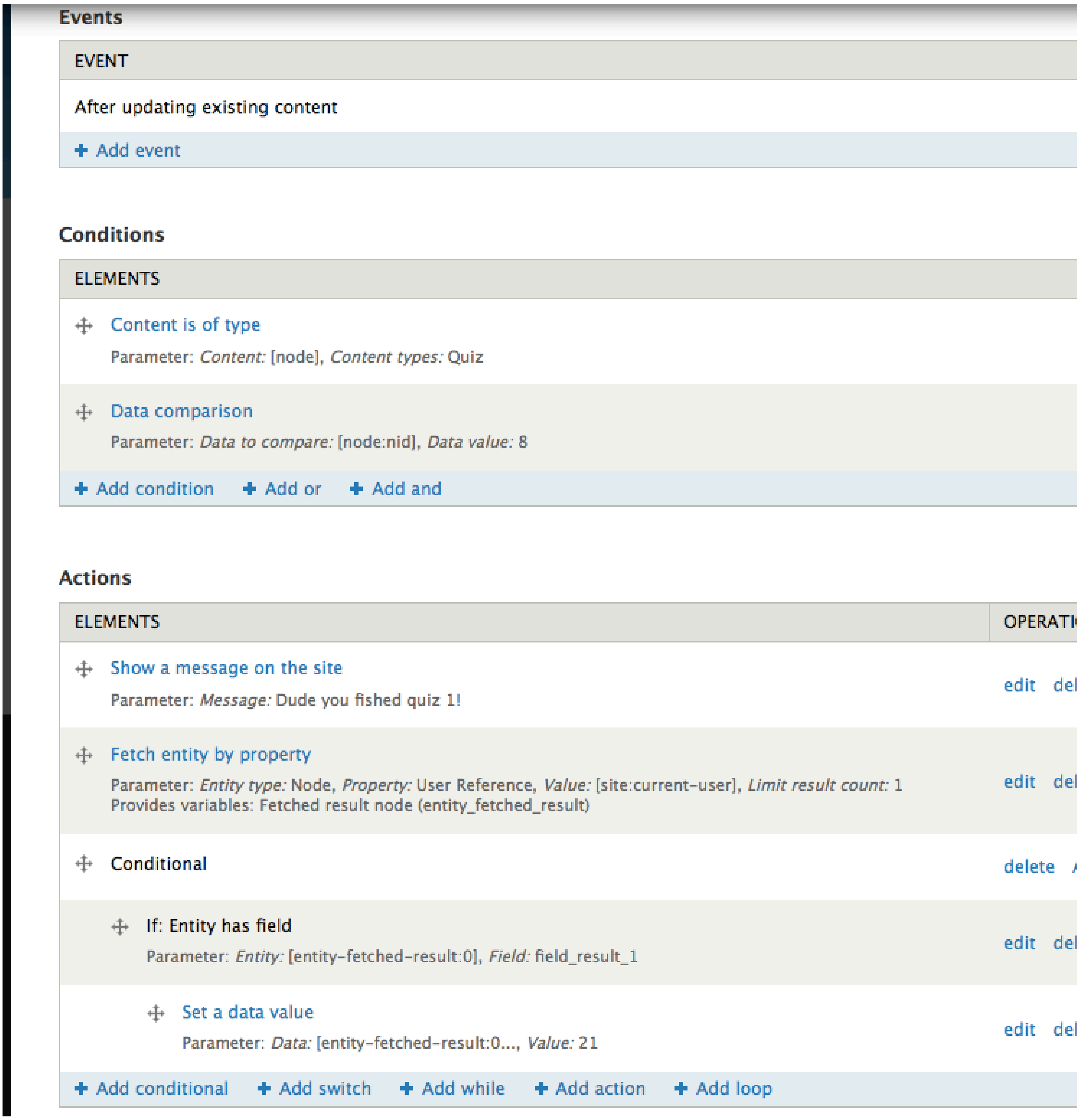
- **Video tutorials**: Johan Falk did an amazing job in the early Drupal 7 days by creating an impressive set of tutorials to "*Learn the Rules Framework*" with a set of over 30 videos and related blogposts hosted at `nodeone.se` (license = Attribution-Noncommercial-Share Alike 3.0).

  However the nodeone.se domain is no longer hosting them. Learn Rules is an attempt to recover these valuable blogposts (with related links to the corresponding videos).

- **The Tiny Book of Rules** is a (15 pages) jumpstart about the Rules module.

Examples

A custom rule shown using the Rules UI



Events

EVENT
After updating existing content
✚ Add event

Conditions

ELEMENTS
Content is of type Parameter: *Content:* [node], *Content types:* Quiz
Data comparison Parameter: *Data to compare:* [node:nid], *Data value:* 8
✚ Add condition ✚ Add or ✚ Add and

Actions

ELEMENTS	OPERATI
Show a message on the site Parameter: *Message:* Dude you fished quiz 1!	edit del
Fetch entity by property Parameter: *Entity type:* Node, *Property:* User Reference, *Value:* [site:current-user], *Limit result count:* 1 Provides variables: Fetched result node (entity_fetched_result)	edit del
Conditional	delete
If: Entity has field Parameter: *Entity:* [entity-fetched-result:0], *Field:* field_result_1	edit del
Set a data value Parameter: *Data:* [entity-fetched-result:0..., *Value:* 21	edit del
✚ Add conditional ✚ Add switch ✚ Add while ✚ Add action ✚ Add loop	

A custom rule shown in Rules Export format

Here is a Rules example in ***Rules export format***:

```
{ "rules_display_userpoints_after_updating_content" : {
    "LABEL" : "Display userpoints after updating content",
    "PLUGIN" : "reaction rule",
    "OWNER" : "rules",
    "REQUIRES" : [ "userpoints_rules", "rules", "rules_conditional" ],
    "ON" : { "node_update" : [] },
    "DO" : [
      { "userpoints_rules_get_current_points" : {
          "USING" : { "user" : [ "site:current-user" ], "tid" : "all" },
          "PROVIDE" : { "loaded_points" : { "total_points" : "Number of points in all
categories together" } }
        }
      },
      { "drupal_message" : { "message" : "You now have [total-points:value] points" } },
      { "CONDITIONAL" : [
          {
            "IF" : { "NOT data_is" : { "data" : [ "total-points" ], "op" : "\u003C", "value" :
"20" } },
            "DO" : [
              { "drupal_message" : { "message" : "You have sufficient points (you still have
[total-points:value] ...)." } }
            ]
          },
          { "ELSE" : [
              { "drupal_message" : { "message" : "You DO NOT have sufficient points (you only
have [total-points:value] ...)." } }
            ]
          }
        ]
      }
    ]
  }
}
```

It does retrieve, as the very first Rules Action (not Rules Condition!) the current amount of user points of a user. If the amount is at least 20, it will display a message starting with "You have sufficient points ...", otherwise the message starts with "You DO NOT have sufficient points ...".

Processing field collection items with Rules

Processing Field collection items with Rules is fun, really! Have a look at this Rule (in Rules export format):

```
{ "rules_calculate_sum_of_prices_in_all_field_collection_items" : {
    "LABEL" : "Calculate sum of prices in all field collection items",
    "PLUGIN" : "reaction rule",
    "OWNER" : "rules",
    "REQUIRES" : [ "rules" ],
    "ON" : { "node_view--article" : { "bundle" : "article" } },
    "IF" : [
      { "entity_has_field" : { "entity" : [ "node" ], "field" : "field_article_details" } }
    ],
    "DO" : [
      { "drupal_message" : { "message" : "\u003Cstrong\u003EDrupal
calculator\u003C\/strong\u003E started ..." } },
      { "variable_add" : {
          "USING" : { "type" : "decimal", "value" : "0" },
          "PROVIDE" : { "variable_added" : { "total_price" : "Price total" } }
        }
      },
      { "LOOP" : {
          "USING" : { "list" : [ "node:field-article-details" ] },
          "ITEM" : { "article_details_item" : "Article details item" },
          "DO" : [
            { "data_calc" : {
                "USING" : {
                  "input_1" : [ "total-price" ],
                  "op" : "+",
                  "input_2" : [ "article-details-item:field-price" ]
                },
                "PROVIDE" : { "result" : { "calculation_result" : "Calculation result" } }
              }
            },
            { "data_set" : { "data" : [ "total-price" ], "value" : [ "calculation-result" ] }
},
            { "drupal_message" : { "message" : "After adding a price of
\u003Cstrong\u003E[article-details-item:field-price]\u003C\/strong\u003E for field collection
item with id \u003Cstrong\u003E[article-details-item:item-id]\u003C\/strong\u003E, subtotal is
\u003Cstrong\u003E[calculation-result:value]\u003C\/strong\u003E." } }
          ]
        }
      },
      { "drupal_message" : { "message" : "The \u003Cstrong\u003ETotal
price\u003C\/strong\u003E for all prices included as field collection items is
\u003Cstrong\u003E[total-price:value]\u003C\/strong\u003E." } },
      { "drupal_message" : { "message" : "\u003Cstrong\u003EDrupal
calculator\u003C\/strong\u003E ended ..." } }
    ]
  }
}
```

Some more details about this rule are below ...

Rules Event:

Content is viewed (of type Article), adapt the machine name of the content type `article` to whatever fits (or use any other Rules Event that fits).

Rules Condition:

Entity has field, whereas the entity is "node", and the machine name of my field collection field is `field_article_details` (adapt this machine name to whatever fits, but make sure you use the field collection field itself).

Rules Actions:

Wake up, here is where the magic (fun?) is going to happen ... These are the Rules Actions involved:

1. `Show a message on the site`, with a message like so:

   Drupal calculator started ...

2. `Add a variable`, whereas it is a variable named `total_price`, decimal (2 digits), initial value 0.

3. `Add a loop`, to iterate over each item of my field collection field (with machine name `field_article_details`), and perform these Rules Actions for each iteration:

- `Calculate a value`, which calculates the sum of `total_price` (defined in Rules Action 2 above) and `article-details-item:field-price` (this is the machine name of the field in the field collection that contains the prices, decimal with 2 digits), and stores the result (sum) in variable `calculation_result`.

- `Set a data value`, which simply copies the value stored in variable `calculation_result` in my `total_price` (defined in Rules Action 2 above). Remark: not sure (not tested), but maybe this `calculation_result` variable can be replaced straight by `total_price` (in the previous action), so that you would not need this action.

- `Show a message on the site`, with a message like so:

  After adding a price of 3.40 for field collection item with id 3, subtotal is 15.00.

4. `Show a message on the site`, with a message like so:

   The Total price for all prices included as field collection items is 26.23.

5. `Show a message on the site`, with a message like so:

   Drupal calculator ended ...

Obviously, this rule is rather a prototype. After you're convinced it works as it should, just remove all Rules Actions with `Show a message on the site`. So that only item 2 and 3 (without its last sub-bullet) is left as Rules Actions.

Showtime ...

Here is a sample of my test results, i.e. the Drupal messages that are shown:

```
Drupal calculator started ...
After adding a price of 2.45 for field collection item with id 1, subtotal is 2.45.
After adding a price of 9.15 for field collection item with id 2, subtotal is 11.60.
After adding a price of 3.40 for field collection item with id 3, subtotal is 15.00.
After adding a price of 1.23 for field collection item with id 4, subtotal is 16.23.
The Total price for all prices included as field collection items is 26.23.
Drupal calculator ended ...
```

More info

If you're not familiar with field collections, try to first digest the answer to "this question".

Read The Rules module online: https://riptutorial.com/drupal/topic/8921/the-rules-module

Chapter 9: The Views module

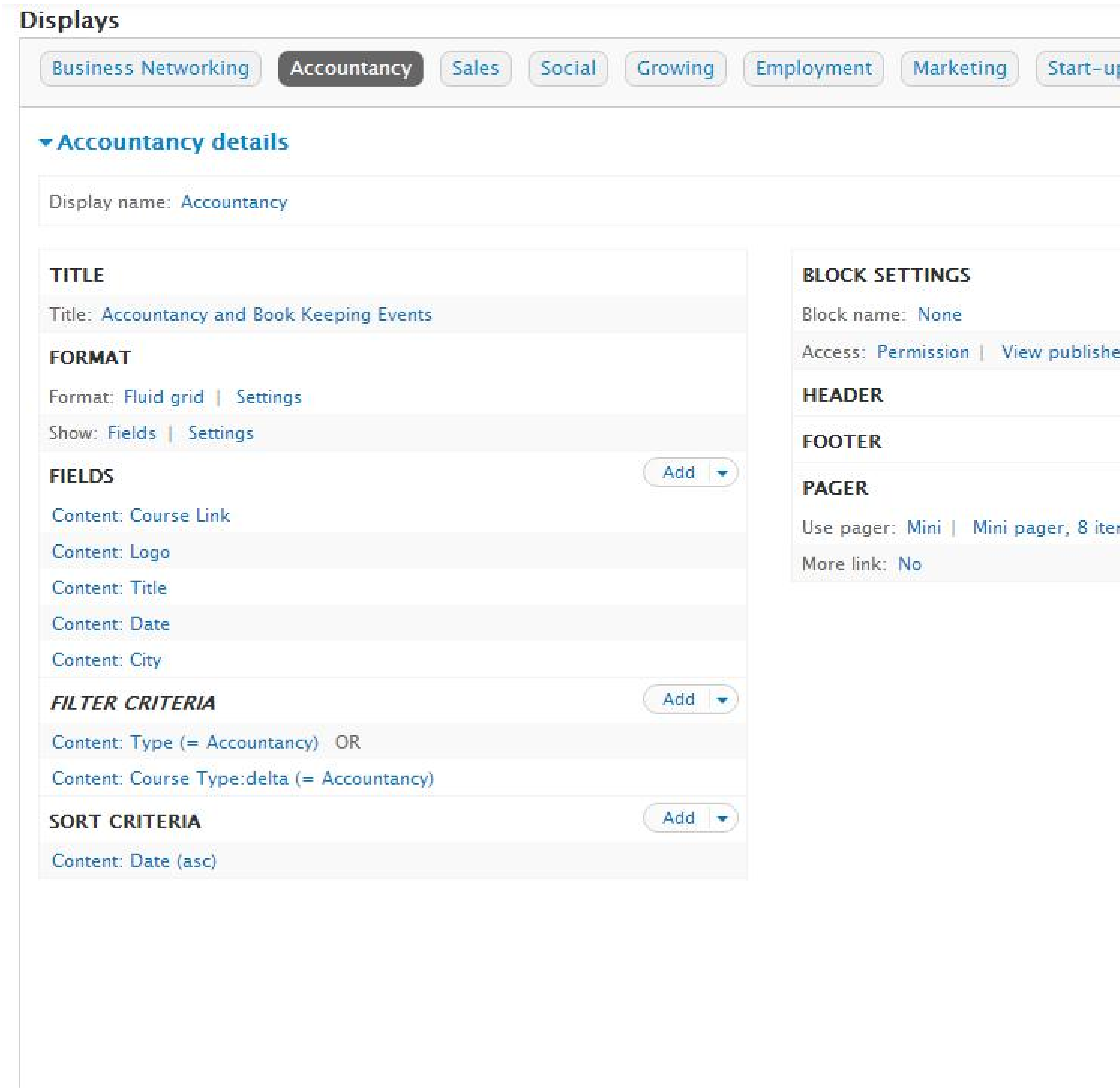
Introduction

The Views module is a reporting engine which allows site builders to create all sorts of lists. These lists can be formatted as either a block or a page.

To create (design) a view, it is required to enter the desired specifications, such as Fields, Relationships, Filters, Sort criteria, Displays, etc.

Examples

A view shown using the Views UI


Displays

Business Networking | Accountancy | Sales | Social | Growing | Employment | Marketing | Start-up

Accountancy details

Display name: Accountancy

TITLE

Title: Accountancy and Book Keeping Events

FORMAT

Format: Fluid grid | Settings

Show: Fields | Settings

FIELDS

Content: Course Link

Content: Logo

Content: Title

Content: Date

Content: City

FILTER CRITERIA

Content: Type (= Accountancy) OR

Content: Course Type:delta (= Accountancy)

SORT CRITERIA

Content: Date (asc)

BLOCK SETTINGS

Block name: None

Access: Permission | View publishe

HEADER

FOOTER

PAGER

Use pager: Mini | Mini pager, 8 iter

More link: No

Read The Views module online: https://riptutorial.com/drupal/topic/9553/the-views-module

Chapter 10: Theme development - Drupal 7

Examples

Writing theme .info files

The .info file is a static text file for defining and configuring a theme. Each line in the .info file is a key-value pair with the key on the left and the value on the right, with an "equals sign" between them (e.g. *name = my_theme*).

Semicolons are used to comment out a line. Some keys use a special syntax with square brackets for building a list of associated values, referred to as an "array". If you are unfamiliar with arrays, have a look at the default .info files that come with Drupal and read the explanations of the examples that follow. Even though the .info file extension is not natively opened by an Application, you can use TextEdit on a Mac or Notepad on a Windows computer in order to view, edit, and save your changes.

Theme name requirements

The name should start with an alphabetic character, can contain numbers and underscores, but not hyphens, spaces or punctuation. The name will be used by Drupal in forming various functions in PHP and therefore it has the same limitations.

Do not choose names that are already used by installed modules, as all installed components must have unique names.

One of the best practices is to use prefixes when naming a site's custom theme, to guarantee unique names for themes. A site named example.com might use theme names such as ex_themename.

Because the .info file is cached, you must clear the cache before any changes are displayed in your site.

The .info file can also specify which theme settings should be accessed from the Drupal administration interface, as you will soon see.

Encoding

The file must be saved as UTF-8 without a Byte Order Mark (BOM).

Contents

Drupal understands the keys listed below. Drupal will use default values for the optional keys not present in the .info file. See the examples set for core themes.

- name ***required***
- description ***recommended***

- screenshot
- version ***discouraged***
- core ***required***
- engine
- base theme
- regions
- features
- theme settings
- stylesheets
- scripts
- php

Theme .info File

```
name = MyCompany Theme
description = A Bootstrap Sub-theme.
core = 7.x
base theme = bootstrap

;;;;;;;;;;;;;;;;;;;;;
;; Regions
;;;;;;;;;;;;;;;;;;;;;

regions[navigation]     = 'Navigation'
regions[header]         = 'Top Bar'
regions[highlighted]    = 'Highlighted'
regions[help]           = 'Help'
regions[content]        = 'Content'
regions[sidebar_first]  = 'Primary'
regions[sidebar_second] = 'Secondary'
regions[footer]         = 'Footer'
regions[page_top]       = 'Page top'
regions[page_bottom]    = 'Page bottom'

;;;;;;;;;;;;;;;;;;;;;
;; MyCompany Custom Regions
;;;;;;;;;;;;;;;;;;;;;
regions[footer_menu_left] = 'Footer menu left'
regions[footer_menu_right] = 'Footer menu right'

;;;;;;;;;;;;;;;;;;;;;
;; CSS
;; these css files will be included on every page
;;;;;;;;;;;;;;;;;;;;;

stylesheets[all][] = css/bootstrap.min.css
stylesheets[all][] = css/MyCompany.css

;;;;;;;;;;;;;;;;;;;;;
;; JS
;; this JS file will be included on every page
;;;;;;;;;;;;;;;;;;;;;

scripts[] = js/MyCompany.min.js
```

Read Theme development - Drupal 7 online: https://riptutorial.com/drupal/topic/2715/theme-development---drupal-7

Chapter 11: Twig

Introduction

Twig is the template engine that is part of Drupal 8. In Drupal 8, Twig files have the extension `.html.twig` and are used in every aspect of Drupal theming. Entities, fields, views can all be rendered using `.html.twig` files.

In this topic the goal is to have a cookbook on how work with Twig in the context of Drupal. If you want to learn more about the syntax or functions available consult the documentation.

Examples

Twig Filter

Contrary to Drupal 7 you cannot call regular PHP functions in your templates. In Drupal 8 the way to go is by creating filters and functions.

You should use a **filter** when: you want to transform the data you want to display. Imagine you have a title that you want to always be uppercase. For example, twig has the `capitalize` filter by default that allows you to transform any text into its uppercase equivalent.

For this example we will create a filter that will allow us to shuffle a string. The way to create filters and functions is exactly the same as regular Twig.

The main difference between regular Twig and Drupal 8 Twig is that in Drupal 8 you must create a service definition of the class your are creating and the class must also belong to a namespace otherwise it will not be registered as a Twig filter within the Drupal environment.

This example assumes that you have a module called `twig_shuffle_extension`.

This will be the basic service definition inn `twig_shuffle_extension.services.yml`

```
services:
  twig_shuffle_extension.twig_extension:
    class: Drupal\twig_shuffle_extension\TwigExtension\TwigShuffleExtension
    tags:
      - { name: twig.extension }
```

The `tags` key is also absolutely required and is what tells Drupal what this class is supposed to do (i.e. register it as a Twig extension).

And now the source code that must be placed in the path defined in the `class` key of the service definition.

```
// Don't forget the namespace!
namespace Drupal\twig_shuffle_extension\TwigExtension;
```

```
use Twig_Extension;
use Twig_SimpleFilter;

class TwigShuffleExtension extends Twig_Extension  {
  /**
   * This is the same name we used on the services.yml file
   */
  public function getName() {
    return 'twig_shuffle_extension.twig_extension';
  }

  // Basic definition of the filter. You can have multiple filters of course.
  // Just make sure to create a more generic class name ;)
  public function getFilters() {
    return [
      new Twig_SimpleFilter('shuffle', [$this, 'shuffleFilter']),
    ];
  }

  // The actual implementation of the filter.
  public function shuffleFilter($context) {
    if(is_string($context)) {
      $context = str_shuffle($context);
    }
    return $context;
  }
}
```

Clear your caches and now, if everything goes according to plan, you can use the filter in your templates.

```
{{ "shuffle me!" | shuffle }}
```

Dependency Injection Into Twig Extensions

This example will show you how to use Dependency Inject to use other services registered in the Drupal environment.

Imagine you have an SVG image file that changes colors depending on some random CSS/Javascript thing in your project. To be able to target the SVG with CSS you have to actually have the SVG file in the DOM. So you create a base SVG file without any colors and place it in your theme folder.

Of course you could just paste the contents of the file in the Twig template but that wouldn't be nice. You can create a Twig extension but you also don't want to hardcode your theme path in the extension source code.

This means we have to get the path dynamically. You have two options:

1. Use the equivalent to a global variable by calling `\Drupal::theme()->getActiveTheme()->getPath();`
2. Inject the `ThemeManager` (given by `\Drupal::theme()`) in your extension class

In this example we will take the second example because it can be widely applicable to any service (you import the Request or the Database Connection if you want).

This assumes that you have a module called `twig_svg_extension` and a `twig_svg_extension.services.yml` file:

```
services:
  twig_svg_extension.twig_extension:
    class: Drupal\twig_svg_extension\TwigExtension\TwigSvgExtension
    arguments: ['@theme.manager']
    tags:
      - { name: twig.extension }
```

Please not the `arguments` key that tells Drupal the service to inject.

```
namespace Drupal\twig_svg_Extension\TwigExtension;

use Drupal\Core\Theme\ThemeManager;
use Twig_Extension;
use Twig_SimpleFilter;

class TwigSvgExtension extends Twig_Extension  {
  private $theme;

  // Dependency injection at work!
  public function __construct(ThemeManager $theme) {
    $this->theme = $theme;
  }

  public function getFilters() {
    return [
      'svg' =>new Twig_SimpleFilter('svg', [$this, 'svgFilter']),
    ];
  }

  public function getName() {
    return 'twig_svg_extension.twig_extension';
  }

  public function svgFilter(string $filepath) {
    $realpath = realpath($this->theme->getActiveTheme()-
>getPath().DIRECTORY_SEPARATOR.$filepath);
    $pathinfo = pathinfo($realpath);

    if($realpath !== false && strtolower($pathinfo['extension']) === 'svg') {
      return file_get_contents($realpath);
    }

    return '"'.$filepath.'" does not exist or is not an SVG';
  }
}
```

Please note the constructor which contains the dependency we injected in the service configuration as well as the `svgFilter` that gets the current active theme path.

`$filepath` should be a relative path to your themes folder. The extension will convert the file path into the contents of the file it points to.

Read Twig online: https://riptutorial.com/drupal/topic/8804/twig

Credits

S. No	Chapters	Contributors
1	Getting started with drupal	acrosman, Adrian Cid Almaguer, chx, Community, dobeerman, Jimmy Ko, Karl Buys, kiamlaluno, Mika A., Morsok, pal4life, Pierre Buyle, Sidney Gijzen
2	Drupal 8 Entity API	Bernard Nandwa
3	Drupal cache and performace	Anurag
4	Drush	Adrian Cid Almaguer, darc1n, Jimmy Ko, lastYorsh, L-four, Mark Conroy, Morsok, pbonnefoi, Rishi Kulshreshtha, Sjoerd van der Vis, Wade Burelbach
5	Example for Drupal 8 Queue API and the Batch API	David Czinege
6	Field Formatter	Ricardo Velhote
7	Module development - Drupal 7	Ajit S, doublejosh, Hermann Döppes, Jimmy Ko, Jimmyb_1991, Morsok, Pierre Buyle
8	The Rules module	Pierre.Vriens
9	The Views module	Pierre.Vriens
10	Theme development - Drupal 7	Adrian Cid Almaguer, Sam Thompson
11	Twig	Ricardo Velhote