CHAPTER 7

Optimizing Your Site Architecture

A poorly designed architecture will haunt you every day of your existence until you either perform major surgery on your site or start over from scratch. Over complicating your Drupal site's foundation will likely result in frustrated content creators, poor performance, and maintenance nightmares. There is an easier way and that is to focus on the right architecture from the beginning.

This chapter focuses on choosing the right approach for the foundation of your site, minimizing the risk of complexity, security, and performance over the life of your site. The four primary areas of focus are as follows:

- The content types that will be used to author and store content on your site.
- The taxonomy that will be used to categorize the content on your site.
- The location of content in an enterprise setting.
- Off-the-shelf versus custom development.

Content Types

One of the first rules of thumb that I learned early in my Drupal career was the fewer content types I could have on a site the easier it was to do virtually everything on the site from content editing to page building. I've walked into several situations where a site had dozens if not hundreds of content types and the poor content editors were at their wit's end trying to figure out which content type to use for what purpose as each content type was nearly identical with the only elements on dozens of content types being title and body.

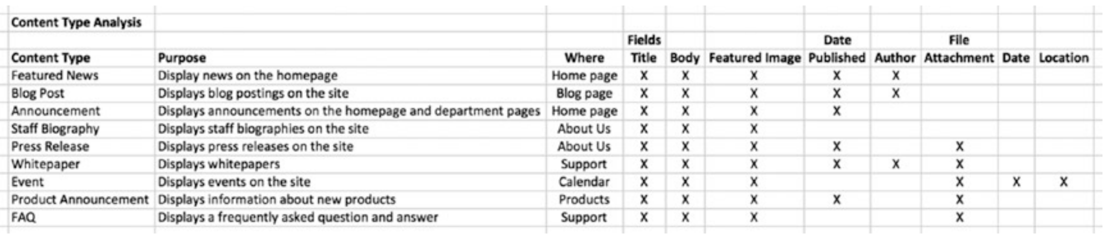
The site builder in those cases believed that they needed to have a different content type for every purpose, for example a separate content type for news, company news, student news, department X news, announcements, staff announcements, product announcements, department X announcements, articles, whitepapers, blog postings, staff biographies, customer success stories, and so on. I've pored through dozens of content types that were exactly the same with the only exception being the name of the content type and where that content resides on the organization's site. The mistaken belief in this case is that in order to segregate content into different categories you need to have a separate content type. Another belief is that if one use case calls for a field that isn't applicable to other use cases, for example, an article content type with just a body and title and a posting content type with a body, title, and image. The reality is that combining the two use cases into a single content type with an image doesn't complicate the work of the editorial staff that maintains the content and, in the scenarios where they don't need an image, they don't add an image. And with different displays, you can render that content type with or without an image.

T. Tomlinson, *Enterprise Drupal 8 Development*, DOI 10.1007/978-1-4842-0253-1_7

Simplification is the key when it comes to defining and constructing your content types. A common technique that will help you distill the list of required content types is to create an inventory of what you believe the content types should be on your site during the information architecture phase of your project. The inventory should list:

- The fields required to capture all of the information to be represented by that content type, e.g., title, body, featured image
- What the content type will be used for, e.g., to display news articles
- Where that content will be rendered on the site, e.g., about us page

A spreadsheet is a great tool for capturing, sorting, and distilling the information. The examples shown in Figure 7-1 point to the likely scenario of only needing one or at the most two content types to address all of the requirements for a site.



Content Type Analysis										
			Fields			Date		File		
Content Type	Purpose	Where	Title	Body	Featured Image	Published	Author	Attachment	Date	Location
Featured News	Display news on the homepage	Home page	X	X	X	X	X			
Blog Post	Displays blog postings on the site	Blog page	X	X	X	X	X			
Announcement	Displays announcements on the homepage and department pages	Home page	X	X	X	X				
Staff Biography	Displays staff biographies on the site	About Us	X	X	X					
Press Release	Displays press releases on the site	About Us	X	X	X	X		X		
Whitepaper	Displays whitepapers	Support	X	X	X	X	X	X		
Event	Displays events on the site	Calendar	X	X	X			X	X	X
Product Announcement	Displays information about new products	Products	X	X	X	X		X		
FAQ	Displays a frequently asked question and answer	Support	X	X	X			X		

Figure 7-1. *Content type analysis spreadsheet*

Examining the spreadsheet, you'll see that all of the content types have common fields with the lone exception of events, which may warrant the use of a separate content type, as it is the only one that has date and location fields. While not all content types may equally use each field, having a flexible general-purpose content type outweighs the complexity of having dozens or hundreds of specific content types.

Using the examples in Figure 7-1, my recommendation is to have two content types, a general-purpose article content type with the following fields:

- Title
- Body
- Featured Image
- Date published
- Author
- File attachment

I would also create an event content type with the same fields plus an event date and a location field.

On the article content type, I suggest adding two taxonomy-driven fields—Article Type and Where Used. The Article Type taxonomy would include the values found in the first column of the spreadsheet, e.g., featured news, blog post, announcement, staff biography, etc. Those values would be used to segregate content based on the type of content. The Where Used taxonomy would be used to simplify the process placing the content in the correct section of the site. Using the Views module, you could create generic views that render article content based on site section from the URL and the type of content, simplifying and minimizing the effort required to build and expand your site.

Simplifying the Editorial Interface

While Chapter 9 focuses on the details of improving the editorial interface in Drupal 8, it warrants a brief discussion on the benefits of leveraging two helpful modules that we recommend installing—the Field Group module (`drupal.org/project/field_group`) and the Simplify module (`drupal.org/project/simplify`). The Field Group module provides the ability to logically group fields on the Node Edit form and to arrange those groups of fields onto vertical tabs, horizontal tabs, or accordions. The Simplify module provides the ability to hide certain fields from the Node Edit form based on user role.

After installing the Field Group module, navigate to the content types administration page (Structure ➤ Content Types) and select the Manage Form Display option in the Operations column. Note that the Add a New Group button has been added to the top of the list of fields (see Figure 7-2).

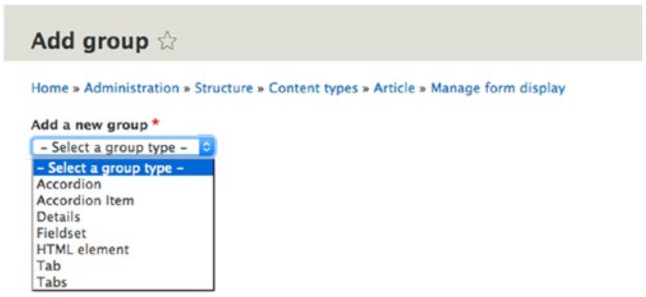
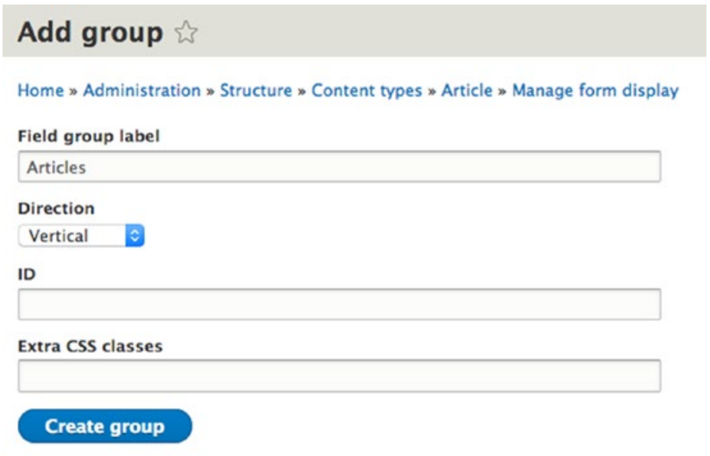


Figure 7-2. *The list of field group options*

For demonstration purposes, we create a new Tabs group by selecting the Tabs option. When you select the Tabs option, a Label field appears. We enter `Articles` in the label and click Save and Continue button. The next form that is displayed (see Figure 7-3) provides the ability to select in which direction the tabs in this group will be displayed (Vertical or Horizontal) as well as the ability to add a CSS ID and extra CSS classes. Let's leave the options at their default values and click Create Group to continue the process.


Add group

Home » Administration » Structure » Content types » Article » Manage form display

Field group label: Articles

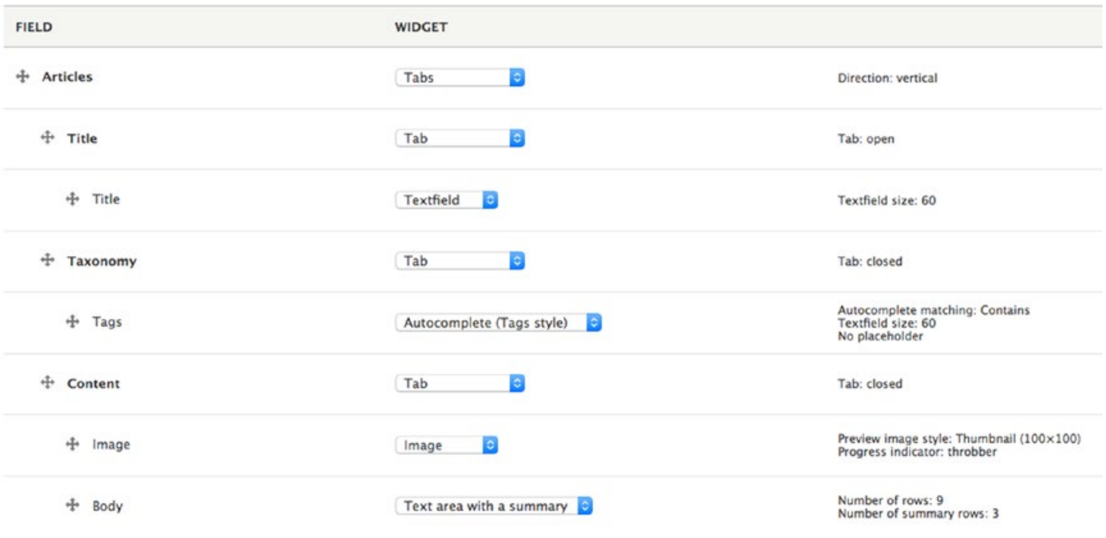
Direction: Vertical

ID:

Extra CSS classes:

Figure 7-3. *Configuring a field group*

After creating the Tabs container, I'll add three individual tabs following the same process with the exception of selecting Tab instead of Tabs and the options from the drop-down list shown in Figure 7-2. The three tabs will be labeled Title, Taxonomy, and Content. After creating the three tabs, I will rearrange the items on the Article Form display (Structure ➤ Content Types ➤ Article ➤ Manage Form Display), as shown in Figure 7-4. The Articles container is the parent of all items that appear on the Article Node Edit form. The Title, Taxonomy, and Content tabs are the next level (indented), and the fields have been moved under their appropriate tabs. To rearrange the items on the page, simply click the + icon and drag the items to their appropriate position and drop them on the page. After rearranging the items, click the Save button to preserve your changes.



FIELD	WIDGET	
Articles	Tabs	Direction: vertical
Title	Tab	Tab: open
Title	Textfield	Textfield size: 60
Taxonomy	Tab	Tab: closed
Tags	Autocomplete (Tags style)	Autocomplete matching: Contains Textfield size: 60 No placeholder
Content	Tab	Tab: closed
Image	Image	Preview image style: Thumbnail (100×100) Progress indicator: throbber
Body	Text area with a summary	Number of rows: 9 Number of summary rows: 3

Figure 7-4. *Rearranging the form fields*

After modifying the Article Node form, navigate to the Content page and click the Add Content button. Select Article on the next page and note the arrangement of the elements on the Create Article page (see Figure 7-5).

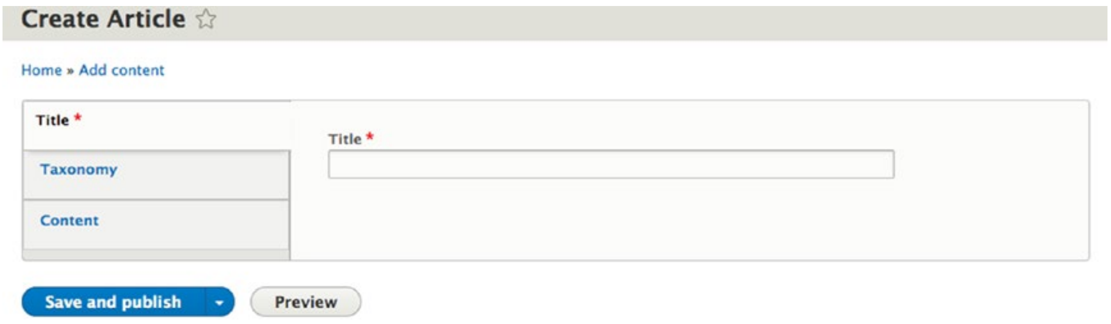
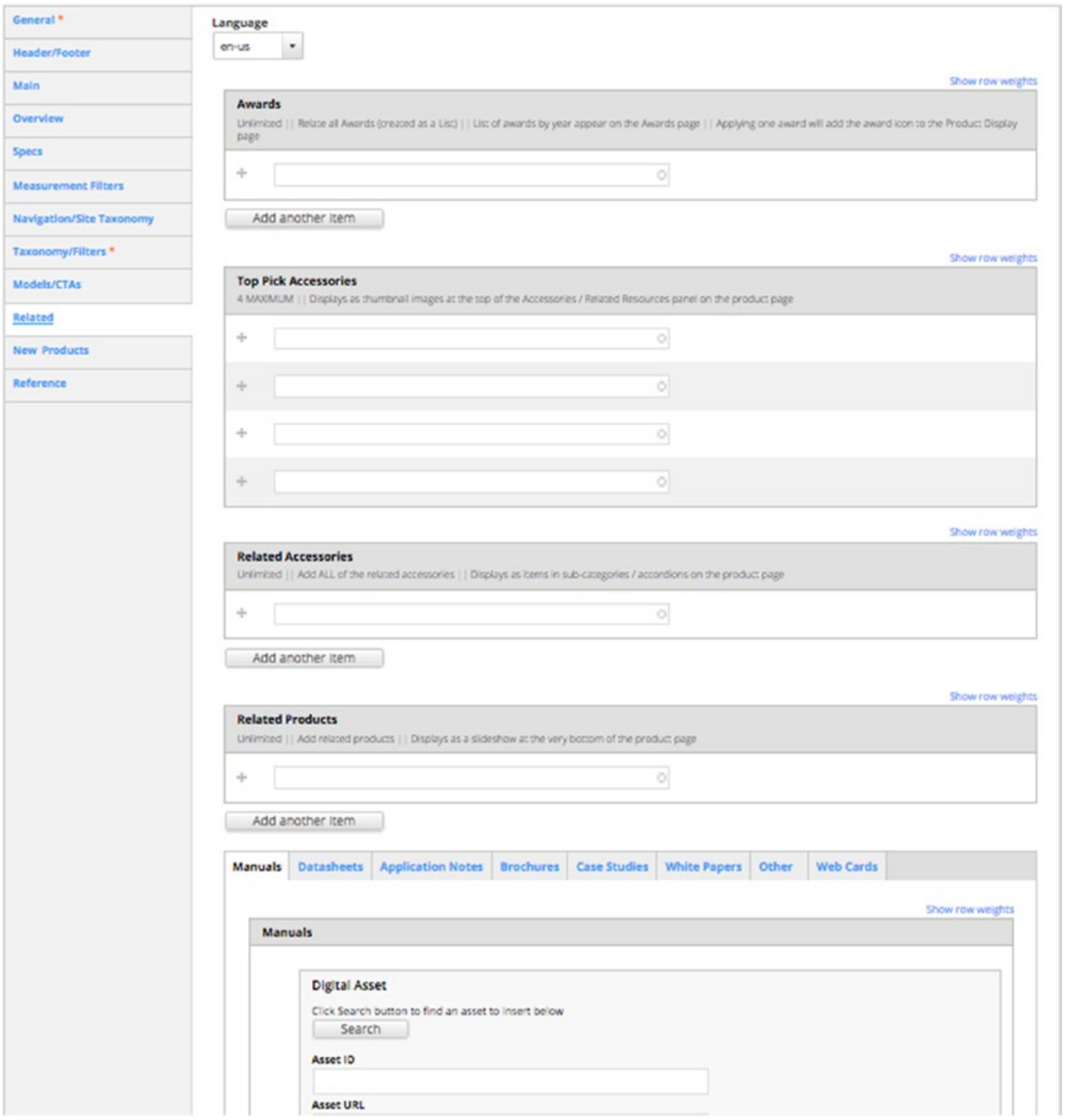


Figure 7-5. *The revised Create Article page*

The Create Article page now has three vertical tabs titled Title, Taxonomy, and Content. Clicking through the three tabs, you can see the fields that were placed on each tab. While the Article Content Type is relatively simple, your editorial team will love you for arranging fields in logical order using tabs instead of a long “river” of fields down the page.

You can also embed horizontal tabs in a vertical tab, making it even more powerful for complex content types. In the example shown in Figure 7-6, this content type has hundreds of fields organized into 12 tabs and on the Related tab there are several embedded horizontal tabs (shown at the bottom of the figure).



Language: en-us

Awards
Unlimited || Relate all Awards (created as a List) || List of awards by year appear on the Awards page || Applying one award will add the award icon to the Product Display page

Top Pick Accessories
4 MAXIMUM || Displays as thumbnail images at the top of the Accessories / Related Resources panel on the product page

Related Accessories
Unlimited || Add ALL of the related accessories || Displays as items in sub-categories / accordions on the product page

Related Products
Unlimited || Add related products || Displays as a slideshow at the very bottom of the product page

Manuals

Digital Asset
Click Search button to find an asset to insert below

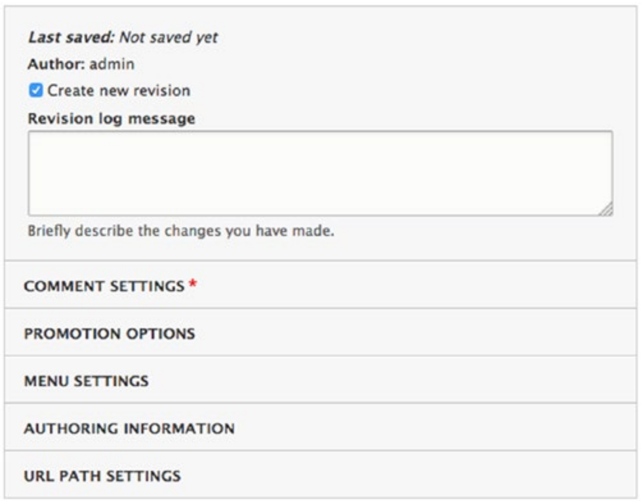
Asset ID

Asset URL

Figure 7-6. *A complicated Node Edit form*

Removing Options from the Node Edit Form

The Node Edit form has several options that you may want to hide (see Figure 7-7).



Last saved: *Not saved yet*

Author: admin

[✓ Create new revision]

Revision log message

Briefly describe the changes you have made.

COMMENT SETTINGS *

PROMOTION OPTIONS

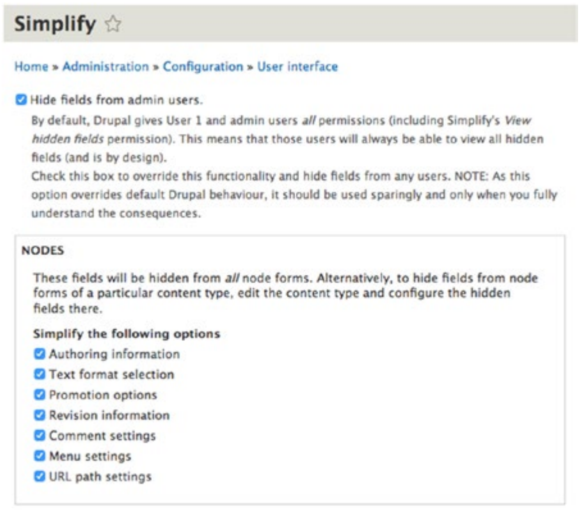
MENU SETTINGS

AUTHORING INFORMATION

URL PATH SETTINGS

Figure 7-7. *The node edit options*

The Simplify module provides the ability to hide the fields shown in Figure 7-7 by simply checking the options listed in Figure 7-8. To access the Simplify form, navigate to Configuration ➤ Simplify.



Simplify ☆

Home » Administration » Configuration » User interface

[✓ Hide fields from admin users.]

By default, Drupal gives User 1 and admin users *all* permissions (including Simplify's *View hidden fields* permission). This means that those users will always be able to view all hidden fields (and is by design).
Check this box to override this functionality and hide fields from any users. NOTE: As this option overrides default Drupal behaviour, it should be used sparingly and only when you fully understand the consequences.

NODES

These fields will be hidden from *all* node forms. Alternatively, to hide fields from node forms of a particular content type, edit the content type and configure the hidden fields there.

Simplify the following options

[✓ Authoring information]
[✓ Text format selection]
[✓ Promotion options]
[✓ Revision information]
[✓ Comment settings]
[✓ Menu settings]
[✓ URL path settings]

Figure 7-8. *The Simplify administration form*

After checking the appropriate options and saving, visit the Node Edit form to see the simplified interface.

Content Types versus Entity Types

Drupal 7 introduced the concept of custom entities, where a custom entity represents any structured content that you want to define outside of Drupal's node, comment, file, user, and taxonomy entities. Examining each of these entity types, you can see that there are fundamental differences between each of them; for example, a node has an author and date published whereas a user does not have either of those two fields. While a node entity type will likely handle 99% of the use cases where you need to create a custom template for capturing, storing, and displaying information, there may be instances where you need something special and you don't want to carry the weight associated with using the node entity type (e.g., permissions). In those rare cases, a custom entity type is likely the best solution.

Creating a custom entity type in Drupal 8 requires a seemingly daunting amount of code. For example, assume that you need to create a new custom entity called a Contact. The Contact entity type has basic information about a person, such as their name, address, phone number, and e-mail address. To create the custom Contact entity, you would need at minimum a:

- `contact_entity.info.yml` file to describe the entity to Drupal 8.
- `contact_entity.routing.yml` file to define the routes associated with the Contact entity type.
- `contact_entity.links.menu.yml` file to define the menu items for the Contact entity type.
- `contact_entity.links.action.yml` file to define the action links for the Contact entity type.
- `contact_entity.links.task.yml` file to add the view, edit, and delete tabs on the entity view page and the settings tab on the entity settings page.
- `src/ContactInterface.php` file to define the public access to the Contact entity.
- `src/Entity/Contact.php` file to define the Contact entity class.
- `src/Form/ContactForm.php` file, which defines the form for adding and editing Contact entity content.
- `src/Form/ContactDeleteForm.php` file, which defines the confirmation form that is called when deleting Contact.
- `src/Entity/Controller/ContactListBuilder.php` file, which defines the header and row content for the Contact listing page.
- `src/Form/ContactSettingsForm.php` file, which creates a settings form for Contact.
- `src/ContactAccessControlHandler.php` file, which defines the access control mechanisms for Contact.

When you examine the amount of code required to create a custom entity it is significant, meaning you really need to have a valid case for why a standard custom content type that uses the node entity wouldn't work for your use case. But there may be a case and you fortunately have an alternative to hand-coding hundreds of lines of code to create an entity and that option is the *Drupal Console.*

The Drupal Console is a tool that generates boilerplate code for Drupal, as well as provides tools for interacting with and debugging Drupal. It will save you countless hours of coding and the high probability of frustration, so I suggest installing it and getting to know its capabilities early in your Drupal 8 journey.

You can download and install Drupal Console from the web site, `drupalconsole.com`. There are three methods for downloading and installing:

- Downloading as a new dependency:

  ```
  # Change directory to Drupal site
  cd path-to-your-drupal8-root-directory

  # Download DrupalConsole
  composer require drupal/console:~1.0 \
  --prefer-dist \
  --optimize-autoloader \
  --sort-packages
  ```

- Downloading using DrupalComposer:

  ```
  composer create-project \
  drupal-composer/drupal-project:8.x-dev \
  path-to-your-drupal8-root-directory \
  --prefer-dist \
  --no-progress \
  --no-interaction
  ```

- Install Drupal Console Launcher:

  ```
  curl https://drupalconsole.com/installer -L -o drupal.phar mv drupal.phar
  /usr/local/bin/drupal chmod +x /usr/local/bin/drupal
  ```

For additional details on installing Drupal Console, visit the `drupalconsole.com` web site.

After installing Drupal Console, the next step is to create your new Contact entity using the Drupal Console command for constructing all of the code required to define the skeleton of the new entity. The first step is to create a new module that will be used as the foundation for the new Contact entity. To create the module, enter the following command from the root directory of your Drupal 8 site:

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

Drupal Console will then walk you through a series of questions about your new Drupal 8 module:

```
// Welcome to the Drupal module generator
  Enter the new module name:
  > mymodule
  Enter the module machine name [mymodule]:
  Enter the module Path [/modules/custom]:
  Enter module description [My Awesome Module]:
  > Creates a Customer entity
  Enter package name [Custom]:
  > Other
  Enter Drupal Core version [8.x]:
Do you want to generate a .module file (yes/no) [yes]:
Define module as feature (yes/no) [no]:
Do you want to add a composer.json file to your module (yes/no) [yes]:
Would you like to add module dependencies (yes/no) [no]:
```

```
Do you want to generate a unit test class (yes/no) [yes]:
Do you confirm generation? (yes/no) [yes]:

Generated or updated files

  1 - /Applications/MAMP/htdocs/d8/modules/custom/mymodule/mymodule.info.yml
  2 - /Applications/MAMP/htdocs/d8/modules/custom/mymodule/mymodule.module
  3 - /Applications/MAMP/htdocs/d8/modules/custom/mymodule/composer.json
  4 - /Applications/MAMP/htdocs/d8/modules/custom/mymodule/src/Tests/LoadTest.php
```

After creating the module, the next step is to create the code required to create the entity. At the command prompt, enter the following:

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

Drupal console will then prompt you with the following questions:

```
  Enter the module name [mymodule]:
  >

  Enter the class of your new content entity [DefaultEntity]:
  > Customer

  Enter the machine name of your new content entity [customer]:
  >

  Enter the label of your new content entity [Customer]:
  >

  Enter the base-path for the content entity routes [/admin/structure]:
  >

  Do you want this (content) entity to have bundles (yes/no) [no]:
  >

  Is your entity translatable (yes/no) [yes]:
  >

  Is your entity revisionable (yes/no) [yes]:
  >

Generated or updated files
  1 - modules/custom/mymodule/mymodule.permissions.yml
  2 - modules/custom/mymodule/mymodule.links.menu.yml
  3 - modules/custom/mymodule/mymodule.links.task.yml
  4 - modules/custom/mymodule/mymodule.links.action.yml
  5 - modules/custom/mymodule/src/CustomerAccessControlHandler.php
  6 - modules/custom/mymodule/src/CustomerTranslationHandler.php
  7 - modules/custom/mymodule/src/Entity/CustomerInterface.php
  8 - modules/custom/mymodule/src/Entity/Customer.php
  9 - modules/custom/mymodule/src/CustomerHtmlRouteProvider.php
```

```
 10 - modules/custom/mymodule/src/Entity/CustomerViewsData.php
 11 - modules/custom/mymodule/src/CustomerListBuilder.php
 12 - modules/custom/mymodule/src/Form/CustomerSettingsForm.php
 13 - modules/custom/mymodule/src/Form/CustomerForm.php
 14 - modules/custom/mymodule/src/Form/CustomerDeleteForm.php
 15 - modules/custom/mymodule/customer.page.inc
 16 - modules/custom/mymodule/templates/customer.html.twig
 17 - modules/custom/mymodule/src/Form/CustomerRevisionDeleteForm.php
 18 - modules/custom/mymodule/src/Form/CustomerRevisionRevertTranslationForm.php
 19 - modules/custom/mymodule/src/Form/CustomerRevisionRevertForm.php
 20 - modules/custom/mymodule/src/CustomerStorage.php
 21 - modules/custom/mymodule/src/CustomerStorageInterface.php
 22 - modules/custom/mymodule/src/Controller/CustomerController.php
```

After constructing the module and the entity, visit your site's modules/custom directory and you will find all of the files generated by Drupal Console:

```
├── composer.json
├── customer.page.inc
├── mymodule.info.yml
├── mymodule.links.action.yml
├── mymodule.links.menu.yml
├── mymodule.links.task.yml
├── mymodule.module
├── mymodule.permissions.yml
├── src
│   ├── Controller
│   │   └── CustomerController.php
│   ├── CustomerAccessControlHandler.php
│   ├── CustomerHtmlRouteProvider.php
│   ├── CustomerListBuilder.php
│   ├── CustomerStorage.php
│   ├── CustomerStorageInterface.php
│   ├── CustomerTranslationHandler.php
│   ├── Entity
│   │   ├── Customer.php
│   │   ├── CustomerInterface.php
│   │   └── CustomerViewsData.php
│   ├── Form
│   │   ├── CustomerDeleteForm.php
│   │   ├── CustomerForm.php
│   │   ├── CustomerRevisionDeleteForm.php
│   │   ├── CustomerRevisionRevertForm.php
│   │   ├── CustomerRevisionRevertTranslationForm.php
│   │   └── CustomerSettingsForm.php
│   └── Tests
│       └── LoadTest.php
└── templates
    └── customer.html.twig

6 directories, 26 files
```

You could at this juncture enable your new module and examine the new custom Customer entity type; however, I want to add a few fields to the entity first so that it represents the requirements that I have for a Customer, namely the name, address, city, state, ZIP code, phone number, and e-mail address fields. For simplicity's sake, I'm going to add each of the fields to the entity type as simple text fields.

Let's edit the `Customer.php` file in the `src/Entity` directory of my module's directory and add the following fields after the `name` field in the `baseFieldDefinitions` function. (Note: The simple way to add all of these fields is to copy the `name` field and change the appropriate values to represent the new field, for example the index in the `$fields` array, the `setLabel` and `setDescription` values):

```
$fields['address'] = BaseFieldDefinition::create('string')
  ->setLabel(t('Address'))
  ->setDescription(t('The address of the the Contact entity.'))
  ->setRevisionable(TRUE)
  ->setSettings(array(
    'max_length' => 50,
    'text_processing' => 0,
  ))
  ->setDefaultValue('')
  ->setDisplayOptions('view', array(
    'label' => 'above',
    'type' => 'string',
    'weight' => -4,
  ))
  ->setDisplayOptions('form', array(
    'type' => 'string_textfield',
    'weight' => -4,
  ))
  ->setDisplayConfigurable('form', TRUE)
  ->setDisplayConfigurable('view', TRUE);

$fields['city'] = BaseFieldDefinition::create('string')
  ->setLabel(t('City'))
  ->setDescription(t('The city of the Contact entity.'))
  ->setRevisionable(TRUE)
  ->setSettings(array(
    'max_length' => 50,
    'text_processing' => 0,
  ))
  ->setDefaultValue('')
  ->setDisplayOptions('view', array(
    'label' => 'above',
    'type' => 'string',
    'weight' => -4,
  ))
  ->setDisplayOptions('form', array(
    'type' => 'string_textfield',
    'weight' => -4,
  ))
  ->setDisplayConfigurable('form', TRUE)
  ->setDisplayConfigurable('view', TRUE);
```

```
    $fields['state'] = BaseFieldDefinition::create('string')
      ->setLabel(t('State'))
      ->setDescription(t('The state of the Contact entity.'))
      ->setRevisionable(TRUE)
      ->setSettings(array(
        'max_length' => 50,
        'text_processing' => 0,
      ))
      ->setDefaultValue('')
      ->setDisplayOptions('view', array(
        'label' => 'above',
        'type' => 'string',
        'weight' => -4,
      ))
      ->setDisplayOptions('form', array(
        'type' => 'string_textfield',
        'weight' => -4,
      ))
      ->setDisplayConfigurable('form', TRUE)
      ->setDisplayConfigurable('view', TRUE);

    $fields['zipcode'] = BaseFieldDefinition::create('string')
      ->setLabel(t('Zipcode'))
      ->setDescription(t('The zipcode of the Contact entity.'))
      ->setRevisionable(TRUE)
      ->setSettings(array(
        'max_length' => 50,
        'text_processing' => 0,
      ))
      ->setDefaultValue('')
      ->setDisplayOptions('view', array(
        'label' => 'above',
        'type' => 'string',
        'weight' => -4,
      ))
      ->setDisplayOptions('form', array(
        'type' => 'string_textfield',
        'weight' => -4,
      ))
      ->setDisplayConfigurable('form', TRUE)
      ->setDisplayConfigurable('view', TRUE);

    $fields['phone'] = BaseFieldDefinition::create('string')
      ->setLabel(t('Phone'))
      ->setDescription(t('The phone number of the Contact entity.'))
      ->setRevisionable(TRUE)
      ->setSettings(array(
        'max_length' => 50,
        'text_processing' => 0,
      ))
      ->setDefaultValue('')
      ->setDisplayOptions('view', array(
        'label' => 'above',
        'type' => 'string',
        'weight' => -4,
      ))
      ->setDisplayOptions('form', array(
        'type' => 'string_textfield',
        'weight' => -4,
      ))
      ->setDisplayConfigurable('form', TRUE)
      ->setDisplayConfigurable('view', TRUE);

    $fields['email'] = BaseFieldDefinition::create('string')
      ->setLabel(t('Email'))
      ->setDescription(t('The email of the Contact entity.'))
      ->setRevisionable(TRUE)
      ->setSettings(array(
        'max_length' => 50,
        'text_processing' => 0,
      ))
      ->setDefaultValue('')
      ->setDisplayOptions('view', array(
        'label' => 'above',
        'type' => 'string',
        'weight' => -4,
      ))
      ->setDisplayOptions('form', array(
        'type' => 'string_textfield',
        'weight' => -4,
      ))
      ->setDisplayConfigurable('form', TRUE)
      ->setDisplayConfigurable('view', TRUE);
```

After adding the fields, save the `Contact.php` file and edit the `CustomerListBuilder.php` file to add the new fields to the `buildHeader` and `buildRow` functions, as shown here. Add each of the fields.

```
class CustomerListBuilder extends EntityListBuilder {

  use LinkGeneratorTrait;

  /**
   * {@inheritdoc}
   */
  public function buildHeader() {
    $header['id'] = $this->t('Customer ID');
    $header['name'] = $this->t('Name');
    $header['address'] = $this->t('Address');
    $header['city'] = $this->t('City');
    $header['state'] = $this->t('State');
    $header['zipcode'] = $this->t('Zip');
    $header['phone'] = $this->t('Phone');
    $header['email'] = $this->t('Email');
    return $header + parent::buildHeader();
  }

  /**
   * {@inheritdoc}
   */
  public function buildRow(EntityInterface $entity) {
    /* @var $entity \Drupal\mymodule\Entity\Customer */
    $row['id'] = $entity->id();
    $row['name'] = $this->l(
      $entity->label(),
      new Url(
        'entity.customer.edit_form', array(
          'customer' => $entity->id(),
        )
      )
    );
    $row['address'] = $entity->address->value;
    $row['city'] = $entity->city->value;
    $row['state'] = $entity->state->value;
    $row['zipcode'] = $entity->zip->value;
    $row['phone'] = $entity->phone->value;
    $row['email'] = $entity->email->value;
    return $row + parent::buildRow($entity);
  }

}
```

After updating the `CustomerListBuilder.php` file and saving it, enable your new module on the Extend page (see Figure 7-9).

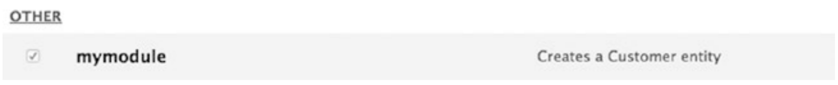
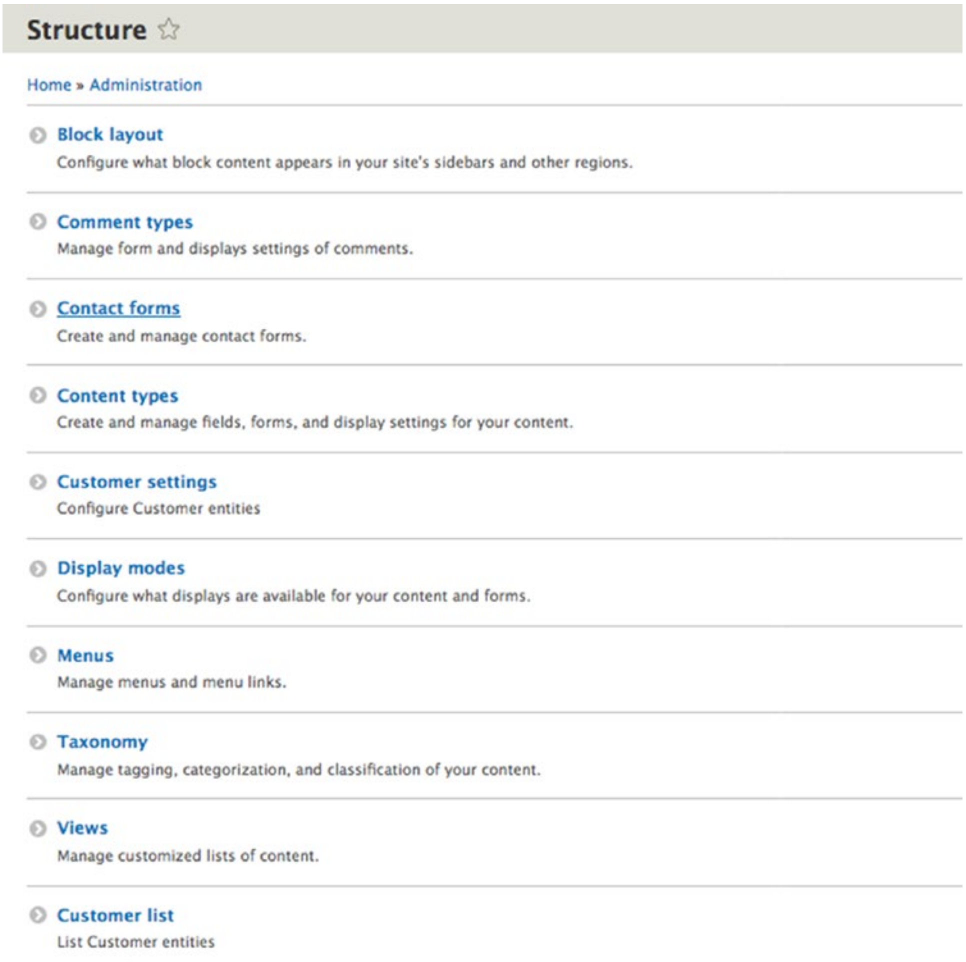


Figure 7-9. *The new customer entity module*

After the module is enabled, you can create, view, and update the customer content by navigating to the Structure page (see Figure 7-10). You will see two new links on the page—Customer Settings and Customer List.



Structure ☆

Home » Administration

Block layout
Configure what block content appears in your site's sidebars and other regions.

Comment types
Manage form and displays settings of comments.

Contact forms
Create and manage contact forms.

Content types
Create and manage fields, forms, and display settings for your content.

Customer settings
Configure Customer entities

Display modes
Configure what displays are available for your content and forms.

Menus
Manage menus and menu links.

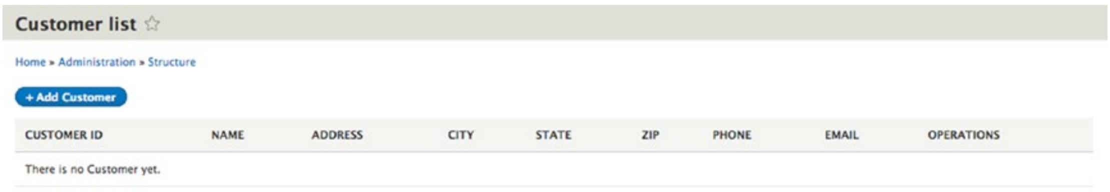
Taxonomy
Manage tagging, categorization, and classification of your content.

Views
Manage customized lists of content.

Customer list
List Customer entities

Figure 7-10. *The Structure page*

Click on the Customer List link to see a list of existing customer records. You'll see each of the fields that were added to the `CustomerListBuilder.php` file at the top of the list (see Figure 7-11).



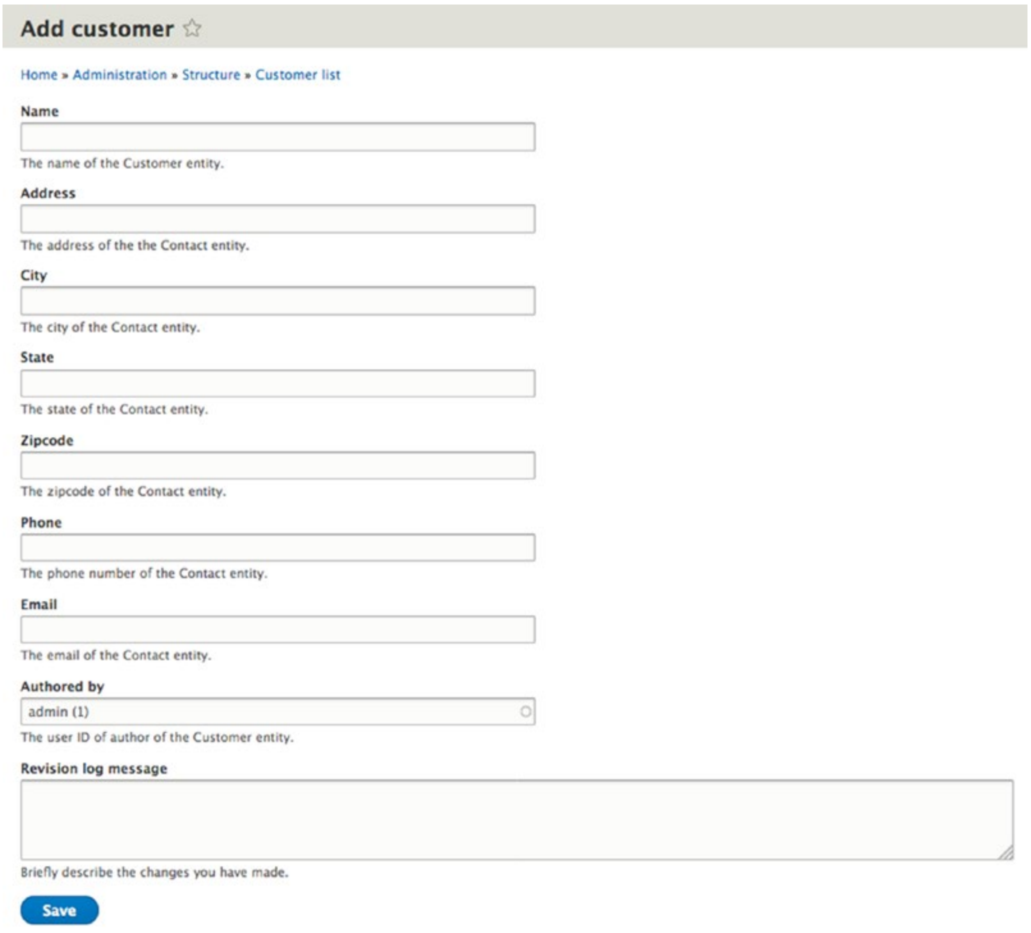
Customer list ☆

Home » Administration » Structure

CUSTOMER ID	NAME	ADDRESS	CITY	STATE	ZIP	PHONE	EMAIL	OPERATIONS
There is no Customer yet.								

Figure 7-11. *The Customer list page*

To add a new Customer, click on the Add Customer button and fill in the fields that were added to the Customer entity. Click the Save button after entering the values (see Figure 7-12).



Add customer ☆

Home » Administration » Structure » Customer list

Name

The name of the Customer entity.

Address

The address of the the Contact entity.

City

The city of the Contact entity.

State

The state of the Contact entity.

Zipcode

The zipcode of the Contact entity.

Phone

The phone number of the Contact entity.

Email

The email of the Contact entity.

Authored by

admin (1)

The user ID of author of the Customer entity.

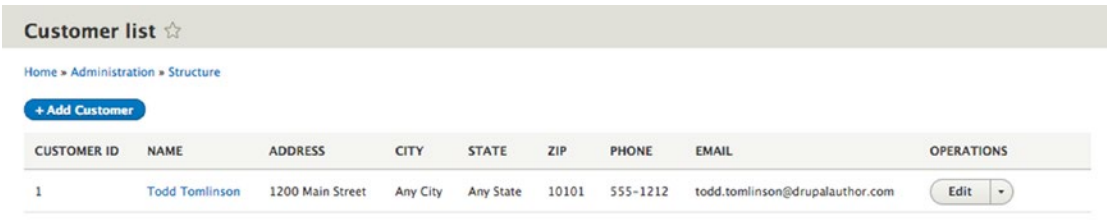
Revision log message

Briefly describe the changes you have made.

Save

Figure 7-12. *Adding a new customer*

After adding a new customer, return to the Structure page and click on the Customer List link. You'll see the new customer in the list (see Figure 7-13).



Customer list ☆

Home » Administration » Structure

+ Add Customer

CUSTOMER ID	NAME	ADDRESS	CITY	STATE	ZIP	PHONE	EMAIL	OPERATIONS
1	Todd Tomlinson	1200 Main Street	Any City	Any State	10101	555-1212	todd.tomlinson@drupalauthor.com	Edit

Figure 7-13. *The new customer appears in the list*

For each item in the custom list, you can perform edits or deletes through the options presented in the Operations column.

You can also perform operations on the Customer entity itself by navigating to Structure ➤ Customer Settings. On this page you'll find tabs to Manage Fields, Manage Form Display, and Manage Display. Each of the operations is identical to a standard entity such as a node. You can add custom fields to your custom entity through the Manage Fields tab (Note: Fields defined in code do not appear on this page; only custom fields that are added through this page appear here.). Rearrange the fields on the Customer Edit form and change the rendering of a customer through the Manage Display tab.

It's also key to understand that customers will not appear on the Content page, similar to how users, comments, and taxonomy terms don't appear on that list. To view the customers, you'll need to visit the Structure page.

Leveraging Taxonomy

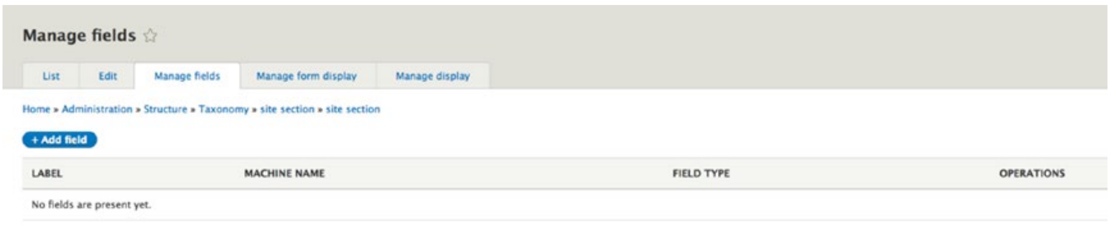
When asked what taxonomy is used for, many people shrug their shoulders and relay the common "freeform tagging" use case as the only area where taxonomy is used on their site, making taxonomy one of the most underutilized capabilities of Drupal's core capabilities. While freeform tagging is a valid use case, there are many more powerful and useful approaches for leveraging taxonomy that will help optimize and streamline your site's architecture.

If you are unfamiliar with taxonomy in Drupal 8, I suggest picking up a copy of *Beginning Drupal 8* from Apress and reviewing the chapter on taxonomy. This section focuses on more advanced use cases of taxonomy.

Taxonomy as an Entity

Before diving into the details of other use cases for taxonomy, it's important to understand that taxonomy is another entity type, like nodes, comments, and users. Because taxonomy is an entity type, Drupal provides the ability to add fields to a taxonomy vocabulary and the terms that are contained with that vocabulary. Having the ability to add custom fields has several benefits, for example, displaying a page banner at the top of a list of content filtered by a taxonomy term. Typically we would have solved this use case by creating a custom block and using block visibility to control when that block would be rendered. The problem with this approach is when you have hundreds of taxonomy terms. In this case, you would have to have hundreds of custom blocks. Managing hundreds of blocks and ensuring that taxonomy terms are synchronized with custom blocks is an administrator's nightmare. Putting the banner image on the term itself solves the problem, and through the use of a generic view that displays the banner image based on an argument in the URL (for example), provides a simple solution to a complex problem.

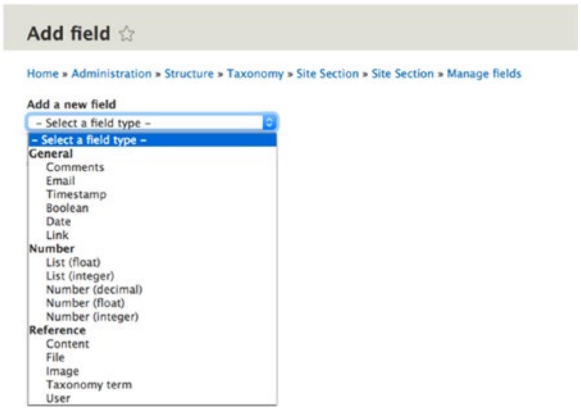
You can extend the scenarios well beyond just storing a banner image on a taxonomy term; you can add virtually any field type to a taxonomy term. To demonstrate this capability, visit Structure ➤ Taxonomy and click on the Add Vocabulary button. We'll create a new vocabulary called Site Section which we'll use to categorize content by where it is supposed to reside on the site. After creating the vocabulary, we click on the Manage Fields tab (see Figure 7-14). The process for creating and managing fields is identical to creating and managing fields on a content type.



LABEL	MACHINE NAME	FIELD TYPE	OPERATIONS
No fields are present yet.			

Figure 7-14. *Creating taxonomy fields*

For demonstration purposes, we add the banner image field. After clicking on the Add Field button, we're presented with a list of types of fields that can be added (see Figure 7-15).



Add field ☆

Home » Administration » Structure » Taxonomy » Site Section » Site Section » Manage fields

Add a new field

- Select a field type -

- Select a field type -

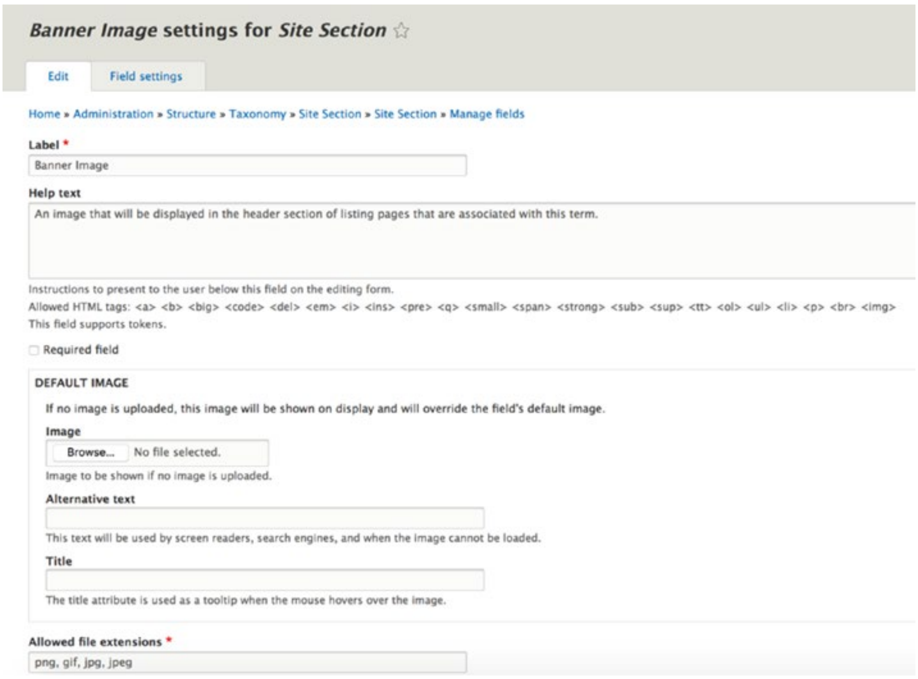
General
- Comments
- Email
- Timestamp
- Boolean
- Date
- Link

Number
- List (float)
- List (integer)
- Number (decimal)
- Number (float)
- Number (integer)

Reference
- Content
- File
- Image
- Taxonomy term
- User

Figure 7-15. *Types of fields*

We select the Image Field type and click the Save and Continue button (which is hidden under the drop-down list in Figure 7-15) to enter the details of the new image field (see Figure 7-16).



Banner Image settings for *Site Section* ☆

Edit | Field settings

Home » Administration » Structure » Taxonomy » Site Section » Site Section » Manage fields

Label *

Banner Image

Help text

An image that will be displayed in the header section of listing pages that are associated with this term.

Instructions to present to the user below this field on the editing form.
Allowed HTML tags: <a> <b> <big> <code> <del> <em> <i> <ins> <pre> <q> <small> <span> <strong> <sub> <sup> <tt> <ol> <ul> <li> <p>
 <img>
This field supports tokens.

☐ Required field

DEFAULT IMAGE

If no image is uploaded, this image will be shown on display and will override the field's default image.

Image

Browse... No file selected.

Image to be shown if no image is uploaded.

Alternative text

This text will be used by screen readers, search engines, and when the image cannot be loaded.

Title

The title attribute is used as a tooltip when the mouse hovers over the image.

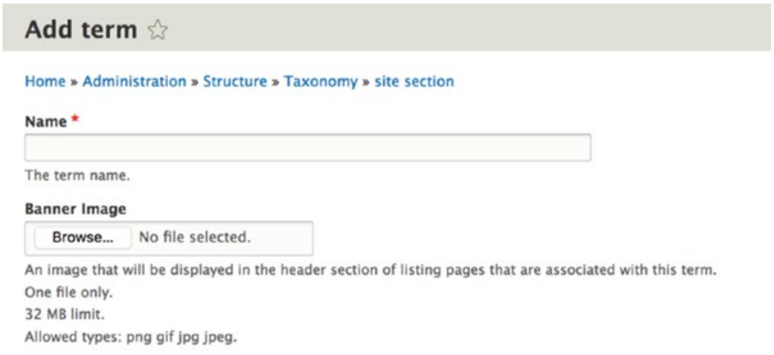
Allowed file extensions *

png, gif, jpg, jpeg

Figure 7-16. *Field details*

After creating the Banner Image field, we can manage where it appears on the Add Term form and manage how it is displayed when a taxonomy term is rendered. The position of the field on the Add Term form can be updated by clicking on the Manage Form display tab (see Figure 7-14) and dragging the new field to the appropriate position in the list of fields. The position of the field when a term is rendered can be updated by clicking on the Manage Display tab. You may reposition the field by dragging and dropping it in the list of fields, you may hide the field label, and you can change the format of the field on this page. All of these actions are identical to how fields are managed on a content type.

After updating the form and display, we add a few new Site Section taxonomy terms. Click on the List tab and then the Add Term button. The Add Term form now has the ability to add a banner image to the term being created (see Figure 7-17).


Add term ☆

Home » Administration » Structure » Taxonomy » site section

Name *

The term name.

Banner Image

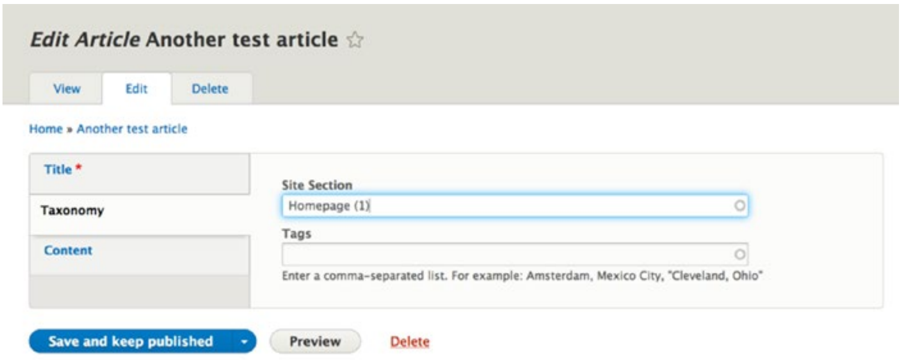
Browse... No file selected.

An image that will be displayed in the header section of listing pages that are associated with this term.
One file only.
32 MB limit.
Allowed types: png gif jpg jpeg.

Figure 7-17. *Add term with the Banner image field*

After creating several Site Section taxonomy terms and attaching banner images to each term, we then update the Article Content Type to include the Site Section taxonomy term as a reference field (see Figure 7-18). To do so, follow these steps:

1. Navigate to Structure ➤ Content Types.
2. Click the Manage Fields button in the Operations column.
3. Click the Add Field button and selected Taxonomy term from the list of possible field types.
4. On the Configuration page for the new field, select Site Section as the source of terms that will be presented to the content author.
5. Save the field.
6. Reposition the field to the position where you want it to appear.
7. Save the Article Content Type.
8. Then create several articles and select one of the Site Section taxonomy terms (homepage) as the location where you want those articles to appear. See Figure 7-18.



Site Section

Homepage (1)

Tags

Enter a comma-separated list. For example: Amsterdam, Mexico City, "Cleveland, Ohio"

Figure 7-18. *Adding a Site Section taxonomy term to an article*

After creating several articles tagged with the Homepage taxonomy term, navigate to Structure ➤ Views and click on the Add View button to create a new view called Page Banner. On the Add View page, set the values as follows:

- View name: Page Banner
- View Settings:
    - Show: Taxonomy terms
    - Of type: Site Section
    - Sorted by: Unsorted
- Block Settings
    - Create a Block: checked
    - Block Display Settings
        - Display format: Unformatted list of Fields
    - Items per block: 1

After clicking the Save and Edit button, update the view's configuration as follows:

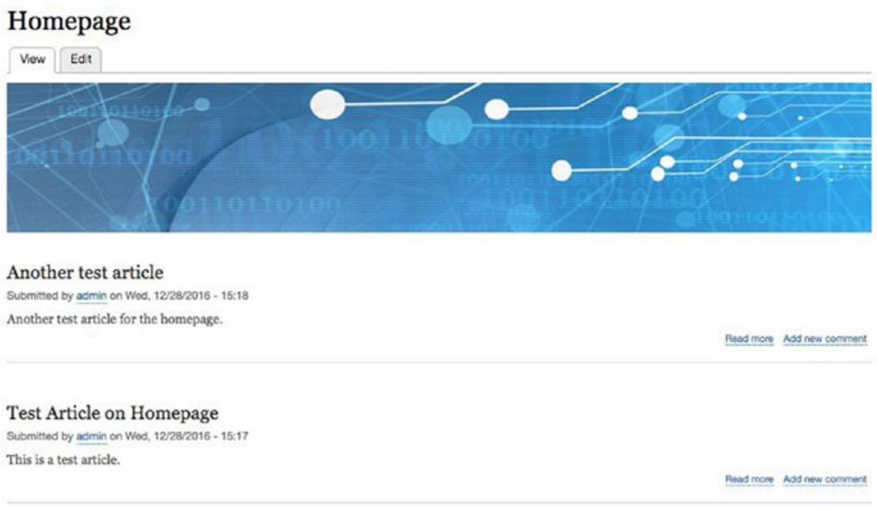
- Display name: Page Banner Block
- Fields: Taxonomy term: Banner Image (this is the name of the field we added to the taxonomy term)
- In the Advanced section, Contextual Filters: Taxonomy term: Term ID (from URL)
- Machine Name: `page_banner_block`

Then save the view.

With the block ready to place on pages, navigate to Structure ➤ Block Layout. On the Block Layout page, click the Place Block button in the Header region and located the Page Banner: Page Banner Block. Click the Place Block button and leave all of the default options as is on the Configure Block page. Click Save Block to place the block in the Header region. On the Block Layout page, click the Save Blocks button at the bottom of the page. With the taxonomy terms, content, view, and block in place, you're ready to test the ability to use the page banner field on the Site Section taxonomy term as the banner at the top of a page associated with that term.

To view the capabilities of this solution, navigate to the taxonomy term listing page for my homepage taxonomy term, which is `/taxonomy/term/1`. Note: Your term ID may be different depending on which term you have used. To find the term ID of the term you have used navigate to Structure ➤ Taxonomy and click the List Terms link in the Operations column of the vocabulary that holds the terms you used for Site Section. Find the term in the list and hover over the Edit link in the Operations column. In the status bar of your browser, you should see the full URL to the edit page for that term. The term ID will appear directly after the `term/` in the URL.

After entering the correct URL in the browsers address bar and visiting the page, you'll see the page banner you assigned to the Homepage taxonomy term displayed at the top of the content area (see Figure 7-19).



Another test article
Submitted by admin on Wed, 12/28/2016 - 15:18
Another test article for the homepage.
Read more Add new comment

Test Article on Homepage
Submitted by admin on Wed, 12/28/2016 - 15:17
This is a test article.
Read more Add new comment

Figure 7-19. *The page banner appears*

We can make the URL more user and SEO friendly by editing the taxonomy term for Homepage and creating a URL alias of `/homepage`. After updating the term, visit the page at `example.com/homepage` (replacing `example.com` with the domain name of your site) and you'll see the same results as you did with `/taxonomy/term/1`.

Building Multipurpose Pages Using Taxonomy

Another area for leveraging taxonomy is building multipurpose pages that render content through views based on values contained in the URL. It may be easiest to understand the concept through an example use case. I'll use the example of a manufacturing company that has several product lines, which each line having multiple products. While I could create a standalone page for each product line, I could just as easily create a single page that uses taxonomy terms in the URL to render content that is specific to that product line, for example `/products/brushes`. With one page I could render an unlimited number of product line landing pages. The only requirements are that each product line is defined by a taxonomy term, and that every product in that product line is tagged with terms from the product line taxonomy.

Laying the Foundation for Multipurpose Pages

While it is possible to provide the functionality required to address this use case through custom code, I'll demonstrate fulfilling the requirements with off-the-shelf modules (ctools, panels, page manager, taxonomy, and views) that require no custom development. While some of the modules used to demonstrate this capability are in alpha or beta at the time this chapter was written, they all function as desired and will only get better as they move to release candidates.

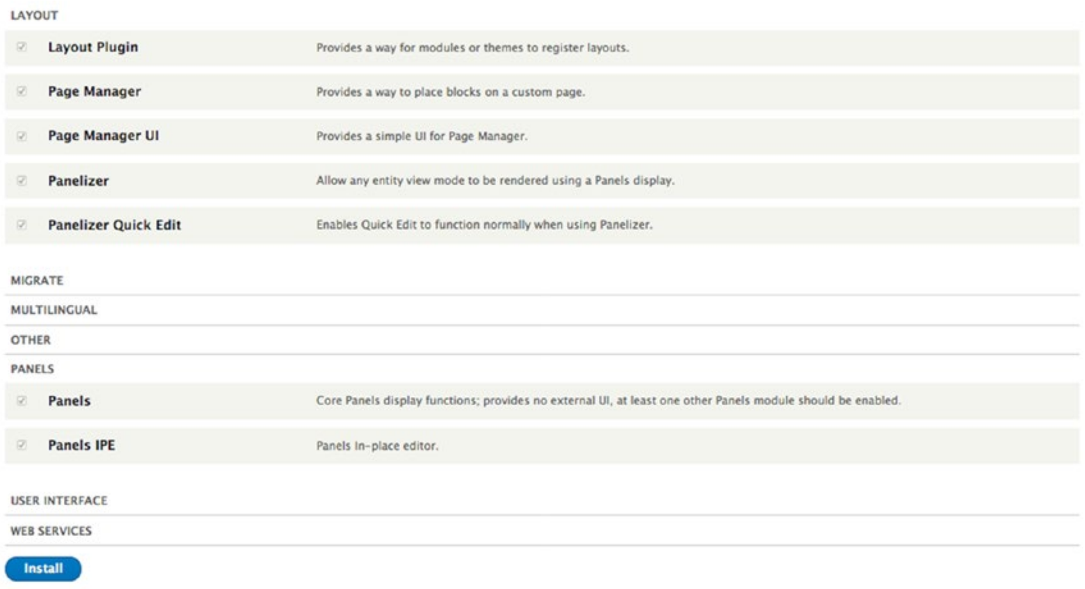
The list of modules that must be downloaded from `drupal.org` and installed are as follows:

- Ctools (`drupal.org/project/ctools`)
- Panels (`drupal.org/project/panels`)
- Page Manager (`drupal.org/project/page_manager`)
- Layout Plugin (`drupal.org/project/layout_plugin`)
- Panelizer (`drupal.org/project/panelizer`)

We assume you have Drush enabled on your site and will download the modules using the following commands:

- `drush dl ctools`
- `drush dl panels`
- `drush dl page_manager`
- `drush dl layout_plugin`
- `drush dl panelizer`

You may then enable the modules through Drush or by visiting the Extent page and checking the box next to each of the modules, followed by clicking on the Install button at the bottom of the page (see Figure 7-20).


LAYOUT

✓	Layout Plugin	Provides a way for modules or themes to register layouts.
✓	Page Manager	Provides a way to place blocks on a custom page.
✓	Page Manager UI	Provides a simple UI for Page Manager.
✓	Panelizer	Allow any entity view mode to be rendered using a Panels display.
✓	Panelizer Quick Edit	Enables Quick Edit to function normally when using Panelizer.

MIGRATE

MULTILINGUAL

OTHER

PANELS

✓	Panels	Core Panels display functions; provides no external UI, at least one other Panels module should be enabled.
✓	Panels IPE	Panels In-place editor.

USER INTERFACE

WEB SERVICES

Figure 7-20. *Enabling the panels-related modules*

After enabling the modules, the steps required to achieve the desired outcome are as follows:

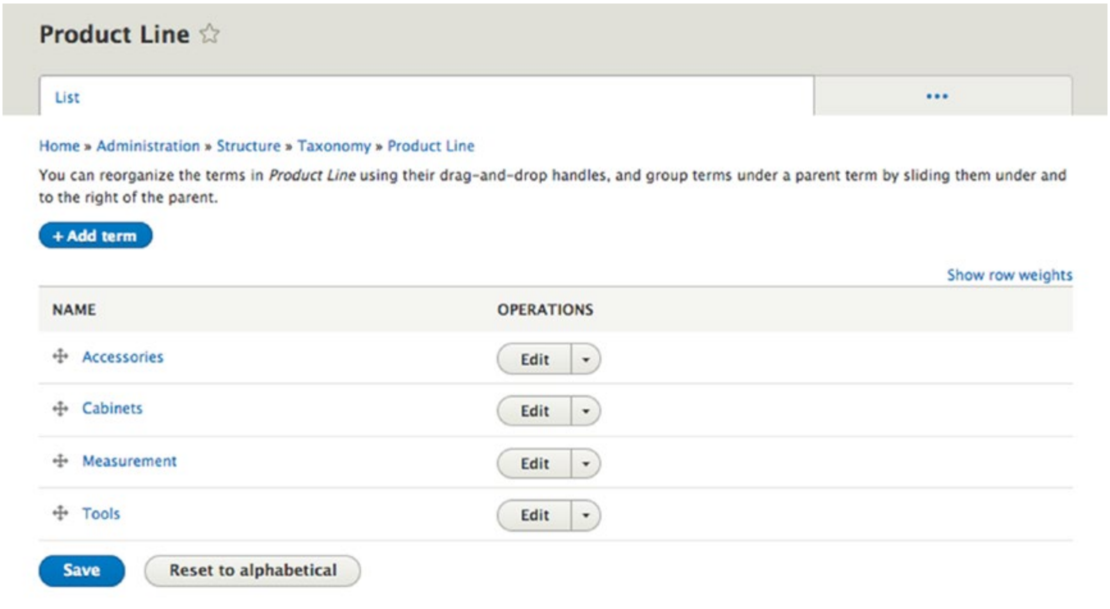
1. Create a new taxonomy vocabulary to house product-line taxonomy terms.
2. Create one to several product line taxonomy terms in the product line vocabulary.
3. Create a product content type with the following fields:
    - Title
    - Description (body)
    - Featured Image
    - Term reference field to the product-line vocabulary
    - A featured product Boolean field
4. Review and update the teaser and full view modes for the product content type.
5. Create several products across multiple taxonomy terms, selecting at least one per taxonomy term as the featured product for that product line.
6. Create a view (block) that renders a list of products, using the teaser view, filtered by product line from the URL.
7. Create a view (block) that renders the featured products (products that are checked as featured), using the teaser view, filtered by product line from the URL.
8. Create a panel page (two columns) that takes the product line as an argument in the URL.
9. Place the featured product block in the right column and the product listing block in the main content area of the page.

Creating the Product Line Vocabulary and Terms

The first step in the process is to create the product line vocabulary. Navigate to Structure ➤ Taxonomy and click on the Add Vocabulary button. Enter Product Line in the Name field and click the Save button. Next click on the Add Term button and add the following terms:

- Tools
- Cabinets
- Measurement
- Accessories

After creating the vocabulary and adding the terms, the list of terms should look similar to Figure 7-21.


Product Line ☆

List | •••

Home » Administration » Structure » Taxonomy » Product Line

You can reorganize the terms in *Product Line* using their drag-and-drop handles, and group terms under a parent term by sliding them under and to the right of the parent.

+ Add term

Show row weights

NAME	OPERATIONS
Accessories	Edit
Cabinets	Edit
Measurement	Edit
Tools	Edit

Save | Reset to alphabetical

Figure 7-21. *The list of product line terms*

Creating the Product Content Type and Product Content

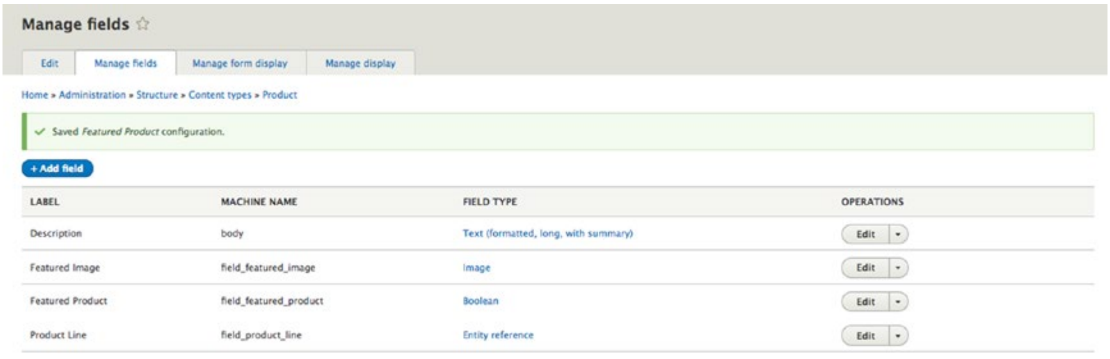
With the Product Line taxonomy in place, the next step is to create the Product Content Type with the following fields:

- Title
- Description (body)
- Featured image
- Featured product (Boolean)
- Product line term reference

Navigate to Structure ➤ Content Types and click the Add Content Type button to begin the process of creating the Product Content Type. Configure the content type by:

- Entering Product in the Name field
- In the Publishing options section, unchecking the Promoted To front page option
- In the Display settings, unchecking the display author and date information option
- In the Menu settings section, unchecking the Main navigation checkbox, resulting in no menus being checked, as we don't want editors to add products to menu

Then click the Save and Manage Fields button to continue the process. By default Drupal creates a title field and a Body field. The title field appeared on the previous page and was fine as is without modifications. Let's change the Label on the body field to read Description by clicking on the Edit button in the Operations column. Delete the value of Body in the Label field and enter Description in its place, followed by clicking the Save Settings button at the bottom of the form. Then add the fields by clicking on the Add Field button and selecting the appropriate types of fields.



Manage fields ☆

Edit | Manage fields | Manage form display | Manage display

Home » Administration » Structure » Content types » Product

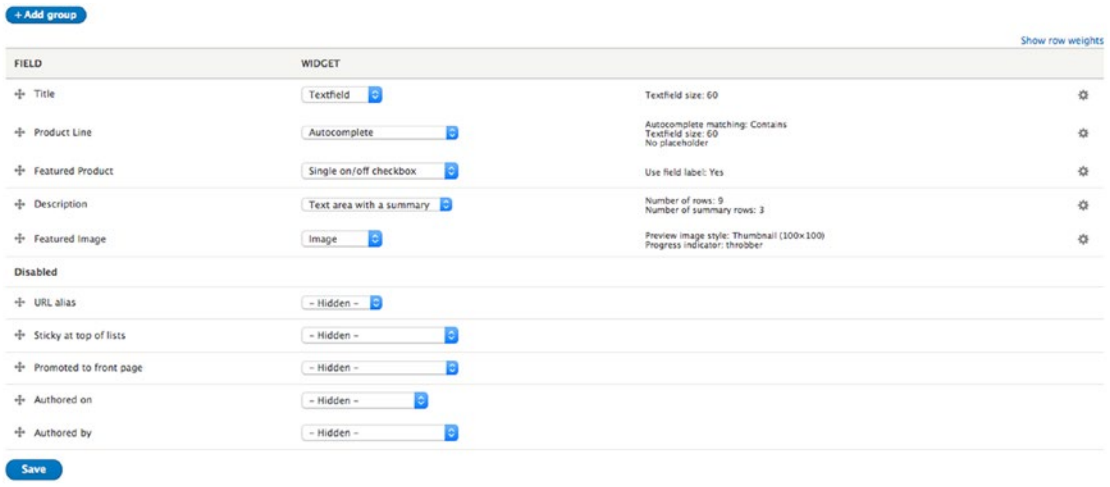
✓ Saved *Featured Product* configuration.

+ Add field

LABEL	MACHINE NAME	FIELD TYPE	OPERATIONS
Description	body	Text (formatted, long, with summary)	Edit
Featured Image	field_featured_image	Image	Edit
Featured Product	field_featured_product	Boolean	Edit
Product Line	field_product_line	Entity reference	Edit

Figure 7-22. *The Product content type and fields*

Rearrange the fields on the form by clicking on the Manage Form Display tab and setting the order as shown in Figure 7-23.



+ Add group

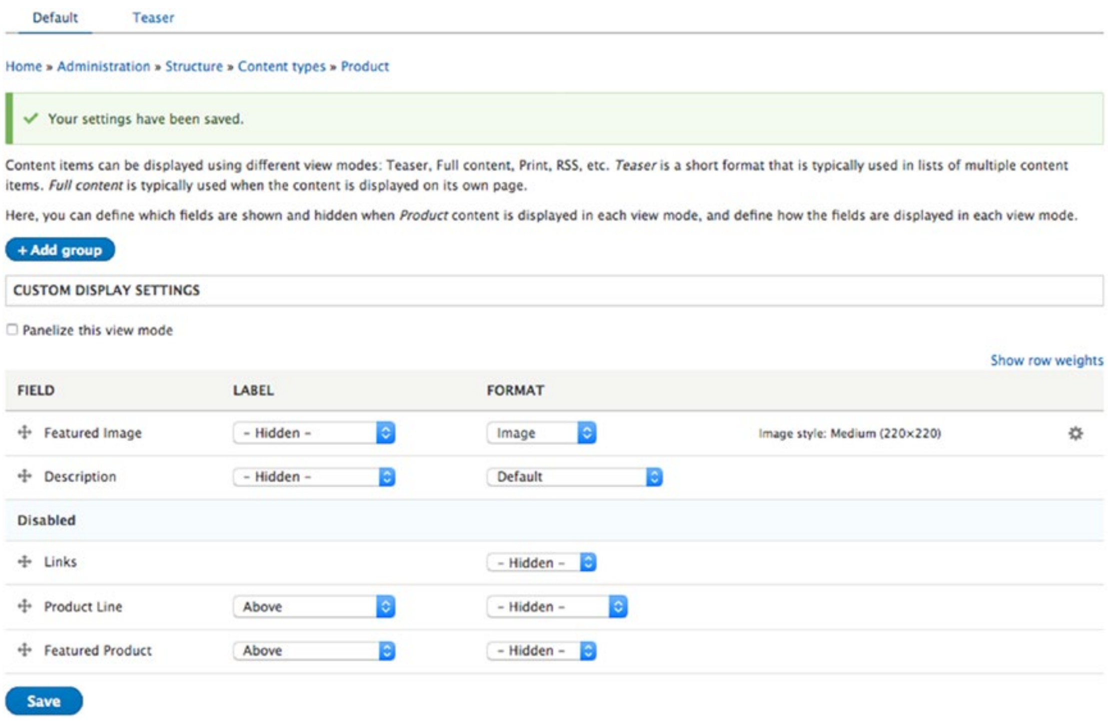
Show row weights

FIELD	WIDGET		
Title	Textfield	Textfield size: 60	⚙
Product Line	Autocomplete	Autocomplete matching: Contains Textfield size: 60 No placeholder	⚙
Featured Product	Single on/off checkbox	Use field label: Yes	⚙
Description	Text area with a summary	Number of rows: 9 Number of summary rows: 3	⚙
Featured Image	Image	Preview image style: Thumbnail (100×100) Progress indicator: throbber	⚙
Disabled			
URL alias	- Hidden -		
Sticky at top of lists	- Hidden -		
Promoted to front page	- Hidden -		
Authored on	- Hidden -		
Authored by	- Hidden -		

Save

Figure 7-23. *The product content type form display field order*

Then click on the Manage Display tab and update the default and teaser displays to only show the Featured Image and Description fields. Set the default image size to 220X220 for the default view mode and 100X100 for the teaser view mode by clicking on the gear icon at the far right of the row for Featured Image (see Figure 7-24).



Your settings have been saved.

Content items can be displayed using different view modes: Teaser, Full content, Print, RSS, etc. *Teaser* is a short format that is typically used in lists of multiple content items. *Full content* is typically used when the content is displayed on its own page.

Here, you can define which fields are shown and hidden when *Product* content is displayed in each view mode, and define how the fields are displayed in each view mode.

CUSTOM DISPLAY SETTINGS

☐ Panelize this view mode

FIELD	LABEL	FORMAT	
Featured Image	- Hidden -	Image	Image style: Medium (220×220)
Description	- Hidden -	Default	
Disabled			
Links		- Hidden -	
Product Line	Above	- Hidden -	
Featured Product	Above	- Hidden -	

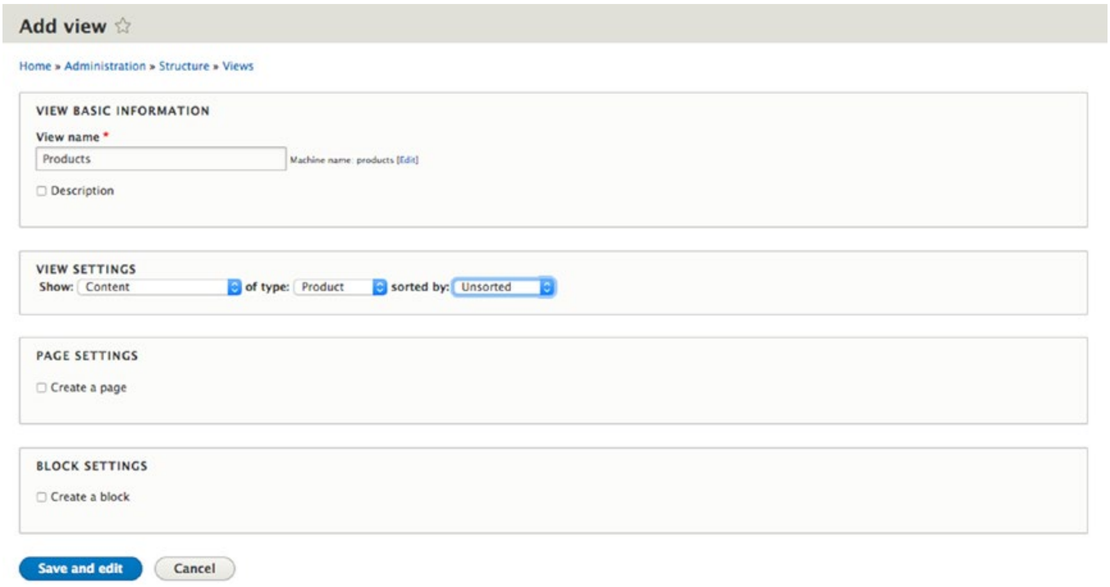
Figure 7-24. *The default display for the product content type*

After saving the display settings, we create several products across each of the Product Line taxonomy terms. Navigate to the Content page and click the Add Content button to create several products across each of the product lines in preparation for the next step.

Creating the Product Views

The requirements call for two views—one that displays the full list of products within a product line and one that randomly displays one of the products that is checked as a featured product (randomly as more than one may be checked as featured within a given product line). Both views use the teaser display mode and both will be created as blocks.

Navigate to Structure ➤ Views and click on the Add View button to create the new view. We use a single view for both blocks. On the Add View page, enter `Products` as the name and update the view settings to show content of type Product sorted by Unsorted. Click the Save and Edit button to continue (see Figure 7-25).


Add view ☆

Home » Administration » Structure » Views

VIEW BASIC INFORMATION

View name *

Products

Machine name: products [Edit]

☐ Description

VIEW SETTINGS

Show: Content of type: Product sorted by: Unsorted

PAGE SETTINGS

☐ Create a page

BLOCK SETTINGS

☐ Create a block

Figure 7-25. *Creating the product views*

Within this single view, we'll create two block displays—one to list all products by product line and one to list a featured product from that product line. For each display we click the Add button and select Block as the type of display.

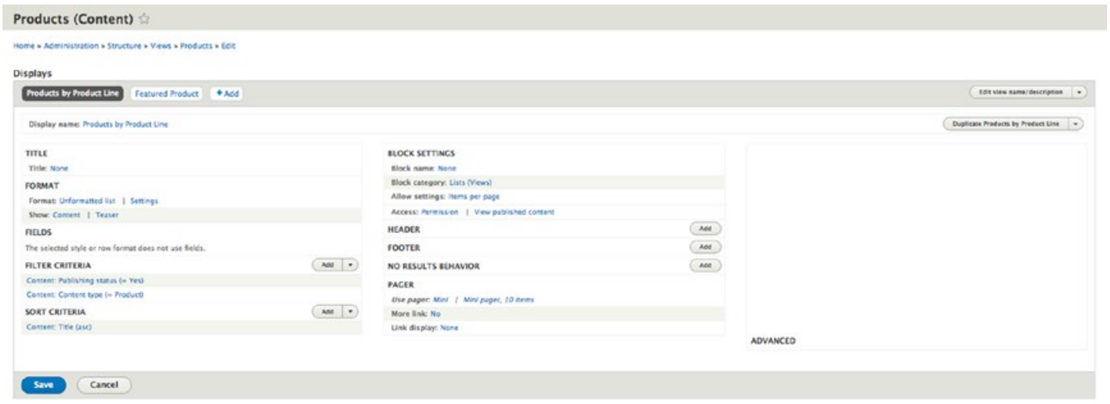
For the product-by-product line block display, set:

- The Display name to Products by Product Line
- Show Content using the teaser display mode
- Sort criteria by title
- Under the advanced section (third column), add a contextual filter for Content: Product Line, setting a default value to Raw value from URL, selecting 2 from the list of Path components (second element in the URL will contain the taxonomy term for product line)

For the featured product block display:

- Set the Display name to Featured product
- Add a Filter criteria for Content: Featured Product set to True
- Add a Sort criteria of Random and remove the Content: Title (asc)
- Use Pager: Display a specified number of items | 1 item
- Under the advanced section (third column), add a contextual filter for Content: Product Line, setting a default value to Raw value from URL, selecting 2 from the list of Path components (second element in the URL will contain the taxonomy term for Product Line).

The resulting view should look similar to Figure 7-26.


Products (Content)
Home » Administration » Structure » Views » Products » Edit
Displays
Products by Product Line
Featured Product
+Add
Edit view name/description
Duplicate Products by Product Line
Add
Add
Add
Add
Add
ADVANCED
Save
Cancel

Display name: Products by Product Line

TITLE
Title: None

FORMAT
Format: Unformatted list | Settings
Show: Content | Teaser

FIELDS
The selected style or row format does not use fields.

FILTER CRITERIA
Content: Publishing status (= Yes)
Content: Content type (= Product)

SORT CRITERIA
Content: Title (asc)

BLOCK SETTINGS
Block name: None
Block category: Lists (Views)
Allow settings: Items per page
Access: Permission | View published content

HEADER

FOOTER

NO RESULTS BEHAVIOR

PAGER
Use pager: Mini | Mini pager, 10 items
More link: No
Link display: None

Figure 7-26. *Details of the product display*

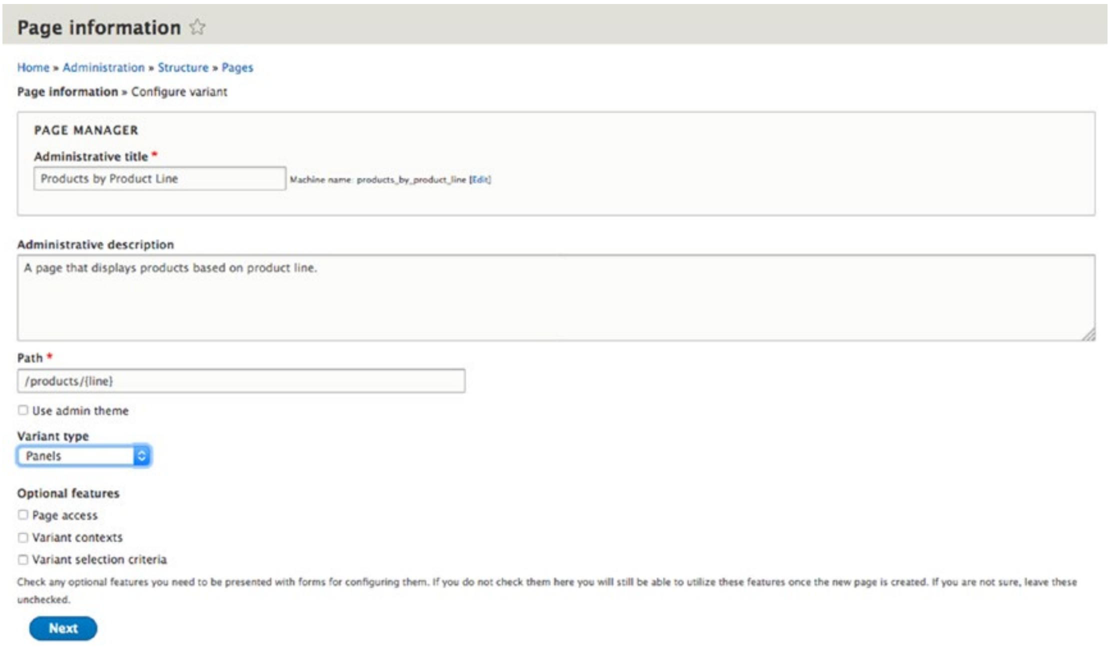
Creating the Product Page

The final step in the process is to create the page where products will be displayed. Previously, we installed and enabled the following modules and their sub-modules: Ctools, Panels, Page Manager, Panelizer, and Layout Plugin. We'll use a majority of these modules to assemble the generic product page.

We start by creating the page through the Page Manager module. Navigate to Structure ➤ Pages and click on the Add Page button. On the Page Information form, set the following values:

- Administrative title: Products by Product Line
- Administrative description: A page that displays products based on product line
- Path: `/products/{line}`
- Variant type: Panels

The value in the Path field is set to `/products/{line}` where `{line}` is a dynamic argument that will hold the various values for the taxonomy terms in the Product Line vocabulary. The value in the braces is only for reference purposes and does not perform any function other than showing up in the administrative interface. For maintenance purposes, you should use a name that is meaningful to others who may have to make changes to this page in the future. After entering the values, you're ready to proceed with the page creation process. Click the Next button (see Figure 7-27).


Page information ☆

Home » Administration » Structure » Pages

Page information » Configure variant

PAGE MANAGER

Administrative title *

Products by Product Line Machine name: products_by_product_line [Edit]

Administrative description

A page that displays products based on product line.

Path *

/products/{line}

☐ Use admin theme

Variant type

Panels

Optional features

☐ Page access

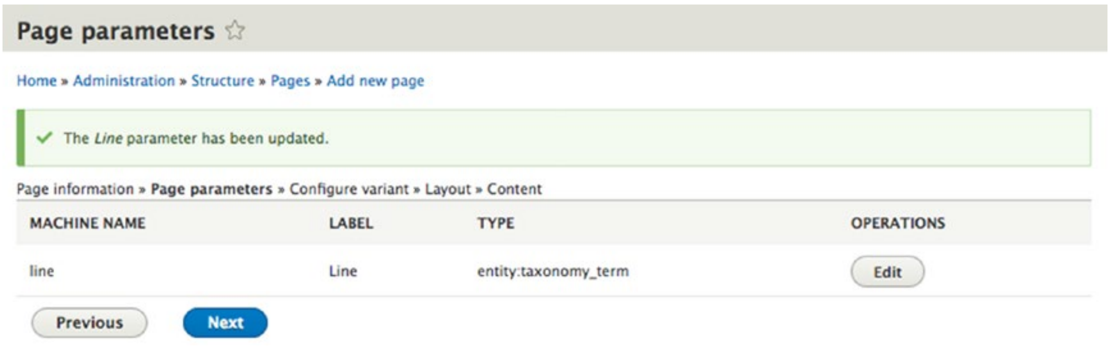
☐ Variant contexts

☐ Variant selection criteria

Check any optional features you need to be presented with forms for configuring them. If you do not check them here you will still be able to utilize these features once the new page is created. If you are not sure, leave these unchecked.

Figure 7-27. *Creating the products by product line page*

The next step assigns context to the argument, `{line}`, in the URL (see Figure 7-28). The value that will be passed in the URL is the taxonomy term associated with a given product line taxonomy term, so we assign the context of Taxonomy term to the line argument by clicking on the Edit button in the operations column. In the list of options presented, we select Taxonomy Term as the type of value that will be passed through the URL, clicking the Update Parameter button to complete the process. The result is shown in Figure 7-28. Click the Next button to continue the page-creation process.


Page parameters ☆

Home » Administration » Structure » Pages » Add new page

✓ The *Line* parameter has been updated.

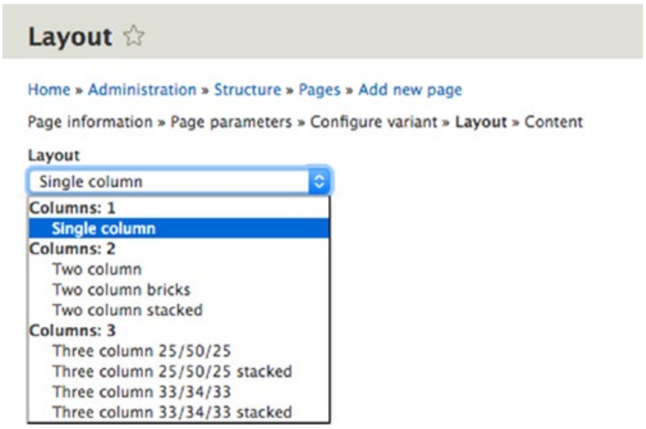
Page information » **Page parameters** » Configure variant » Layout » Content

MACHINE NAME	LABEL	TYPE	OPERATIONS
line	Line	entity:taxonomy_term	Edit

Figure 7-28. *Assigning context to the URL argument*

The next page, Configure Variant, presents the opportunity to change the builder that is used to manage the page once it has been created. The default, Standard, requires that the site administrator visit Structure ➤ Pages in order to make changes to the layout or elements placed on the page. The In-Place Editor option provides the ability to edit the page directly while visiting that page by clicking on buttons at the bottom of the page (e.g., Change Layout). Let's leave the Builder set to Standard and continue with the build process by clicking the Next button.

The next step in the process is to select the layout for the page. There are several off-the-shelf options, including one-, two-, and three-column layouts (see Figure 7-29).


Layout ☆

Home » Administration » Structure » Pages » Add new page

Page information » Page parameters » Configure variant » **Layout** » Content

Layout

Single column

Columns: 1
- Single column

Columns: 2
- Two column
- Two column bricks
- Two column stacked

Columns: 3
- Three column 25/50/25
- Three column 25/50/25 stacked
- Three column 33/34/33
- Three column 33/34/33 stacked

***Figure 7-29.** The layout options*

For demonstration purposes, we select the two-column layout and then click the Next button to continue the build process.

While the off-the-shelf layouts provide a number of options, you may find situations where one of the existing layouts does not meet your design requirements. In those cases you can create your own custom layouts. Navigate to `/modules/panels/layouts` and you'll see the existing layouts and how they are constructed. Each layout consists of a Twig template file and CSS to style the output of the layout.

```
.
├── onecol
│   ├── onecol.css
│   ├── onecol.png
│   └── panels-onecol.html.twig
├── threecol_25_50_25
│   ├── panels-threecol-25-50-25.html.twig
│   ├── threecol_25_50_25.css
│   └── threecol_25_50_25.png
├── threecol_25_50_25_stacked
│   ├── panels-threecol-25-50-25-stacked.html.twig
│   ├── threecol_25_50_25_stacked.css
│   └── threecol_25_50_25_stacked.png
├── threecol_33_34_33
│   ├── panels-threecol-33-34-33.html.twig
│   ├── threecol_33_34_33.css
│   └── threecol_33_34_33.png
│
```

```
├── threecol_33_34_33_stacked
│   ├── panels-threecol-33-34-33-stacked.html.twig
│   ├── threecol_33_34_33_stacked.css
│   └── threecol_33_34_33_stacked.png
├── twocol
│   ├── panels-twocol.html.twig
│   ├── twocol.css
│   └── twocol.png
├── twocol_bricks
│   ├── panels-twocol-bricks.html.twig
│   ├── twocol_bricks.css
│   └── twocol_bricks.png
└── twocol_stacked
    ├── panels-twocol-stacked.html.twig
    ├── twocol_stacked.css
    └── twocol_stacked.png
```

Examining the two-column layout's Twig file, you'll note that the structure is relatively simple:

```
<div class="panel-2col" {% if css_id %}{{ css_id }}{% endif %}>
  <div class="panel-panel">
    {{ content.left }}
  </div>

  <div class="panel-panel">
    {{ content.right }}
  </div>
</div>
```

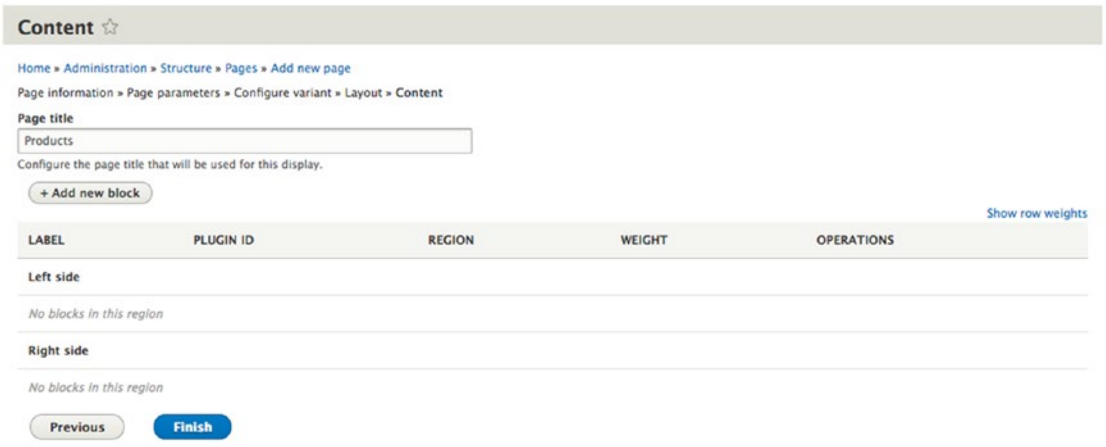
The associated CSS is just as simple:

```
.panel-2col {
  display: flex;
  flex-wrap: wrap;
  justify-content: space-between;
}

.panel-2col > .panel-panel {
  flex: 0 1 50%;
}
```

You can use this as the foundation to build your own layouts.

The next step in the process is to assign blocks to the regions that are provided by the layout. The final step in the process is to assign the blocks that were created by the Products view in the left and right columns of the new page. On the Content page (see Figure 7-30), enter Products into the Page title field and click the Add New Block button to select the blocks to place on the page.



Content ☆

Home » Administration » Structure » Pages » Add new page

Page information » Page parameters » Configure variant » Layout » Content

Page title: Products

Configure the page title that will be used for this display.

+ Add new block

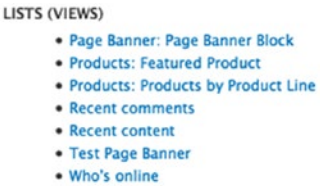
Show row weights

LABEL	PLUGIN ID	REGION	WEIGHT	OPERATIONS
Left side				
No blocks in this region				
Right side				
No blocks in this region				

Previous | Finish

Figure 7-30. *The Content page where blocks are placed on the page*

After clicking the Add New Block button, you're presented with a list of blocks that are available for placement on the page. Scrolling through the list, you'll find a section called Lists (views). The two blocks that were created by the Products view are in that list (see Figure 7-31).

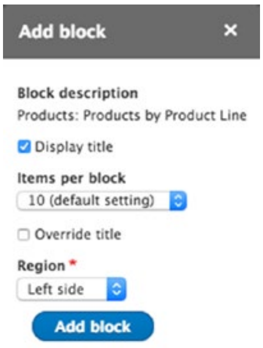


LISTS (VIEWS)

- Page Banner: Page Banner Block
- Products: Featured Product
- Products: Products by Product Line
- Recent comments
- Recent content
- Test Page Banner
- Who's online

Figure 7-31. *The list of views*

Click on the Products: Product by Product Line block and on the Add block form (see Figure 7-32). Then select the left side region and click the Add Block button.



Add block ×

Block description
Products: Products by Product Line

☑ Display title

Items per block: 10 (default setting)

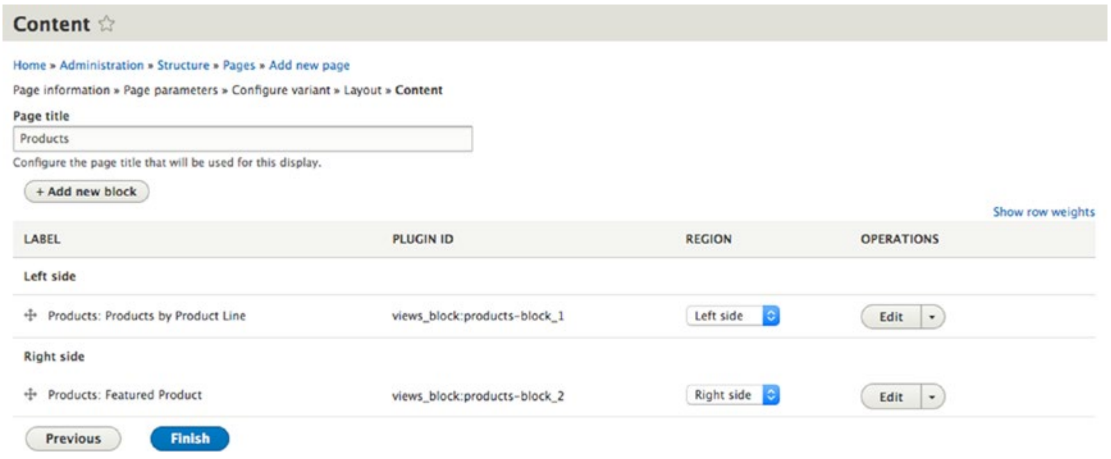
☐ Override title

Region *: Left side

Add block

Figure 7-32. *The Add Block form*

Click the Add New Block button again and select the Products: Featured Product block. Assign it to the right side region and then click the Add Block button. With the blocks placed, it's time to click the Finish button (see Figure 7-33).


Content

Home » Administration » Structure » Pages » Add new page

Page information » Page parameters » Configure variant » Layout » Content

Page title

Products

Configure the page title that will be used for this display.

+ Add new block

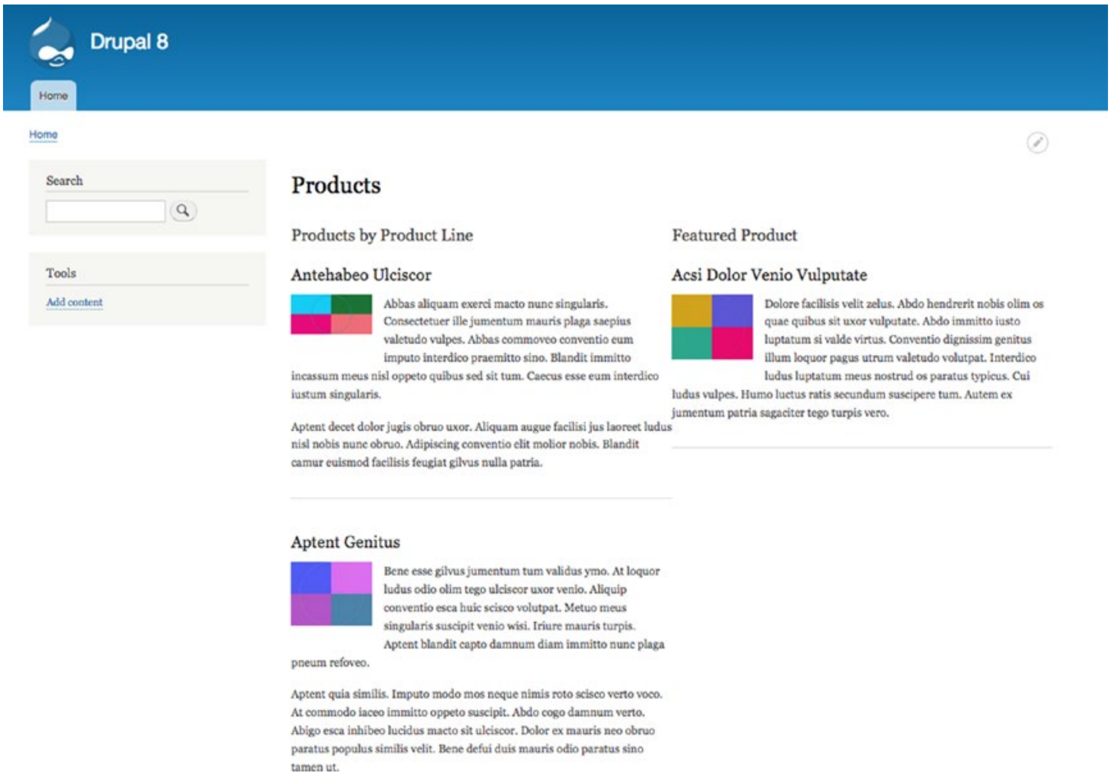
Show row weights

LABEL	PLUGIN ID	REGION	OPERATIONS
Left side			
Products: Products by Product Line	views_block:products-block_1	Left side	Edit
Right side			
Products: Featured Product	views_block:products-block_2	Right side	Edit

Previous   Finish

Figure 7-33. *The blocks placed on the page*

The final step is the click the Save button on the Page Information page. After creating the page, check the term IDs for the various product lines. Navigate to Structure ➤ Taxonomy and list the terms for the Product Lines vocabulary. Hovering over the Edit button for each term, you can see that term ID in the URL in the browser's status bar. With that information in hand, you can update the URL in the browser bar, entering `/products/8` to test the page and ensure that it is working properly. In this case, 8 happens to be the term ID for the Accessories product line. The result is shown in Figure 7-34.



Products

Products by Product Line

Antehabeo Ulciscor

Abbas aliquam exerci macto nunc singularis. Consectetuer ille jumentum mauris plaga saepius valetudo vulpes. Abbas commoveo conventio eum imputo interdico praemitto sino. Blandit immitto incassum meus nisl oppeto quibus sed sit tum. Caecus esse eum interdico iustum singularis.

Aptent decet dolor jugis obruo uxor. Aliquam augue facilisi jus laoreet ludus nisl nobis nunc obruo. Adipiscing conventio elit molior nobis. Blandit camur euismod facilisis feugiat gilvus nulla patria.

Aptent Genitus

Bene esse gilvus jumentum tum validus ymo. At loquor ludus odio olim tego ulciscor uxor venio. Aliquip conventio esca huic scisco volutpat. Metuo meus singularis suscipit venio wisi. Iriure mauris turpis. Aptent blandit capto damnum diam immitto nunc plaga pneum refoveo.

Aptent quia similis. Imputo modo mos neque nimis roto scisco verto voco. At commodo iaceo immitto oppeto suscipit. Abdo cogo damnum verto. Abigo esca inhibeo lucidus macto sit ulciscor. Dolor ex mauris neo obruo paratus populus similis velit. Bene defui duis mauris odio paratus sino tamen ut.

Featured Product

Acsi Dolor Venio Vulputate

Dolore facilisis velit zelus. Abdo hendrerit nobis olim os quae quibus sit uxor vulputate. Abdo immitto iusto luptatum si valde virtus. Conventio dignissim genitus illum loquor pagus utrum valetudo volutpat. Interdico ludus luptatum meus nostrud os paratus typicus. Cui ludus vulpes. Humo luctus ratis secundum suscipere tum. Autem ex jumentum patria sagaciter tego turpis vero.

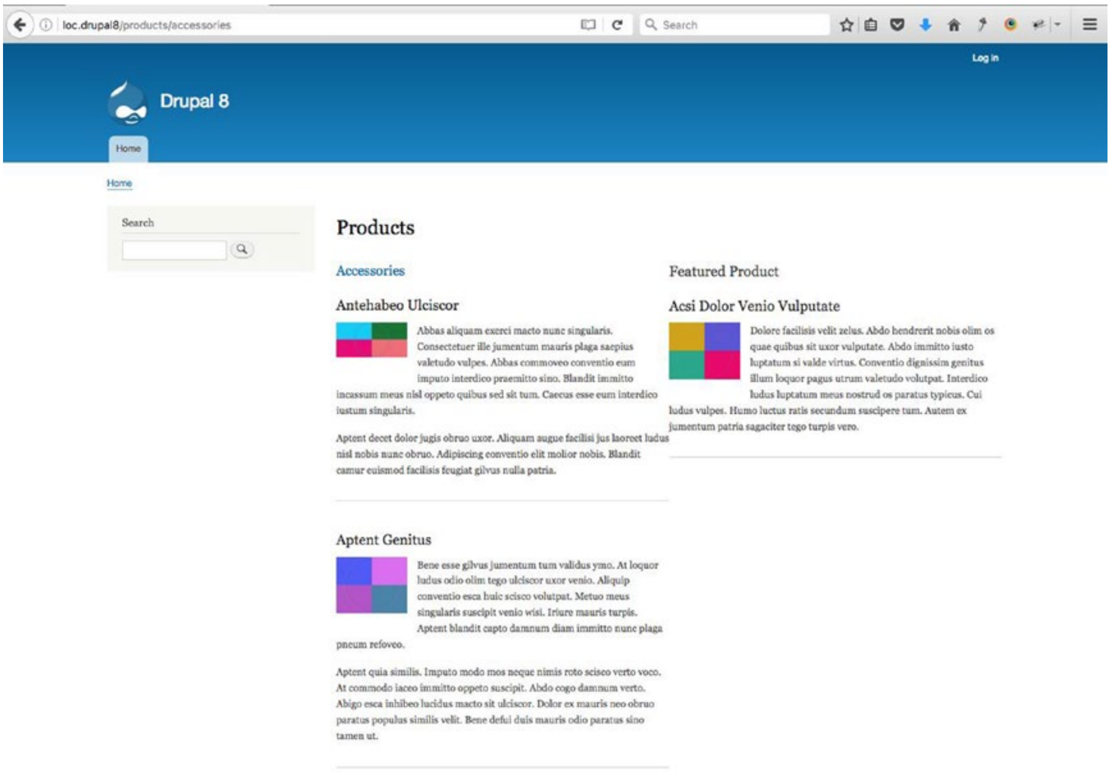
Figure 7-34. *The Product page filtered by product line*

It works as expected, but there are opportunities to improve the solution beyond this basic implementation. The following changes could make the page more visitor friendly:

- Create URL aliases for the `products/term-id` paths. For example, it would be more user friendly to see `products/accessories` in the URL instead of `products/8`. To create that URL alias, navigate to Configuration ➤ URL Aliases and click on the Add Alias button. In the Existing system path, enter `/products/8`. In the Path alias field, enter `/products/accessories`. Then save the alias and continue creating the other aliases for the other product lines. After the aliases are in place, you can use `products/accessories` in the URL and you'll see the products that have been tagged with the accessories taxonomy term.

- Create a new view that lists the terms from the Product Line vocabulary. This block view would show the term name and would have a contextual filter that is identical to the Products block views. You could then place this block at the top of the page to indicate which product line the page is referencing.

After making the two suggested changes, you can now see a visitor and SEO friendly URL in the browser's address bar, as well as an indication of what product line the page is referring to with the new title above the list of products (see Figure 7-35).



Products

Accessories

Antehabeo Ulciscor

Abbas aliquam exerci macto nunc singularis. Consectetuer ille jumentum mauris plaga saepius valetudo vulpes. Abbas commoveo conventio eum imputo interdico praemitto sino. Blandit immitto incassum meus nisl oppeto quibus sed sit tum. Caecus esse eum interdico iustum singularis.

Aptent decet dolor jugis obruo uxor. Aliquam augue facilisi jus laoreet ludus nisl nobis nunc obruo. Adipiscing conventio elit molior nobis. Blandit camur euismod facilisis feugiat gilvus nulla patria.

Aptent Genitus

Bene esse gilvus jumentum tum validus ymo. At loquor ludus odio olim tego ulciscor uxor venio. Aliquip conventio esca huic scisco volutpat. Metuo meus singularis suscipit venio wisi. Iriure mauris turpis. Aptent blandit capto damnum diam immitto nunc plaga pneum refoveo.

Aptent quia similis. Imputo modo mos neque nimis roto scisco verto voco. At commodo iaceo immitto oppeto suscipit. Abdo cogo damnum verto. Abigo esca inhibeo lucidus macto sit ulciscor. Dolor ex mauris neo obruo paratus populus similis velit. Bene defui duis mauris odio paratus sino tamen ut.

Featured Product

Acsi Dolor Venio Vulputate

Dolore facilisis velit zelus. Abdo hendrerit nobis olim os quae quibus sit uxor vulputate. Abdo immitto iusto luptatum si valde virtus. Conventio dignissim genitus illum loquor pagus utrum valetudo volutpat. Interdico ludus luptatum meus nostrud os paratus typicus. Cui ludus vulpes. Humo luctus ratis secundum suscipere tum. Autem ex jumentum patria sagaciter tego turpis vero.

Figure 7-35. *The revised product page*

The concepts presented in this section can be expanded to address a wide variety of use cases. The solution presents a write-once-use-many-times approach, which will significantly decrease your development and testing effort, and it provides an interesting opportunity that allows content editors to create entirely new site sections without having to touch a single line of code or template file. All you need to do is add new taxonomy terms to vocabularies like Site Section and you're off and running.

The Location of Content in an Enterprise Setting

A common problem that medium-to-large organizations face is the duplication of content across various platforms, including multiple Drupal sites, and in other applications such as enterprise resource planning (ERP), customer relationship management (CRM), and the various marketing platforms the organization uses. The issue often quickly becomes a problem of synchronization of content across all of these platforms, for example, the description of an item in the product catalog in the ERP system may be updated and not reflected on the various web sites that present that information to customers. Or a customer's address may be updated in CRM but never updated across the various web sites where customer's shop online. While the content distribution mechanisms outlined earlier in this book present an opportunity to synchronize content across Drupal instances in the organization, it only addresses a portion of the larger problem.

There is a solution that addresses the broader problem and that is considering a platform such as Solr as a mechanism for integrating content across Drupal sites as well as across enterprise applications outside the realm of Drupal.

Using Apache SOLR

A revolutionary approach for integrating content from multiple sources is one that many of us use on our existing Drupal sites but we fall short of the true power of this solution, and that is Apache Solr. We use Solr to index content on a single Drupal site as a more powerful replacement for Drupal's standard search capabilities that are inherent in core. We can use Solr's integration with views to speed up the delivery of content on pages, as Solr's indexes are optimized for speed. But here's the good new—Solr can index multiple sources of content and present that content as a unified index, meaning we can index all of our Drupal-based content as well as content from other sources and deliver that content through a single index. When content is added or updated on those source systems, Solr updates the index. Solr also indexes the content on your local site by providing a single source of enterprise-wide content.

Think of the power and flexibility of being able to access information from your:

- Enterprise resource planning (ERP) system for product information, including updated product availability, and pricing
- Customer Relationship Management (CRM) system
- Product catalog solution for product images and marketing materials
- Digital asset management solution for product brochures and spec sheets
- Other Drupal sites within your organization, providing the ability to share blog postings, articles, events, and other content that may be applicable to your site

The possibilities are virtually limitless. Just as with Solr, if you can get to the information, you can index it and share it through the index.

What Does a Solr-Based Solution Require?

The foundation of the solution is a Solr server. You may install your own instance of Solr on your own server or you may purchase a hosted Solr solution such as the one provided by OpenSolr (`opensolr.com`). If you don't have the skills or resources to implement Solr internally, the hosted solutions are a very cost effective solution and provide high availability and scalability packages that would be difficult for most organizations to build and support.

Once your Solr instance is available, implement the Search Solr API module (`drupal.org/project/search_api_solr`) and the Search API module (`drupal.org/project/search_api`). Follow the instructions for each module to install/enable them as some require composer to pull in various dependencies. To configure the connection to your Solr server, visit `/admin/config/search/search-api` and click on Add Server. Provide a name for the server and the connection details for your Solr server.

Next, create a new index by visiting `/admin/config/search/search-api`. Click on Add Index and give the index a name and select at least one data source.

■ **Note** If you are indexing multiple sites, use the same index name for all of your sites. Otherwise each site will have their own index without the ability to search across sites. Select the server you just created and leave all the other default values. Then save.

Test the connection to your Solr index by creating some content and checking to see if that content appears in the Index page of your Solr server. If the connection is correct and the content you just created appears in the index, you are good to go.

Consuming Indexed Information Through Views

With all of your sites indexed through Solr you now have the ability to create views using your Solr index as the source of content. When creating a new view, select the name of the Solr index that you created as the source of content to display and continue building the view just as you would any other view. It's just that easy.

Off-the-Shelf versus Custom Development

The final section of this chapter touches on a touchy subject, do I make it or do I use something that is already built? When I started working with Drupal back in the Drupal 3.x days, there were only a handful of contributed modules and to do anything beyond the basics required custom development. Today, with Drupal 8, core itself has a significant footprint of functional capabilities that meet many of the basic requirements for developing simple to moderately complex web sites. When you throw in the 2,500 or so contributed modules that are currently available for Drupal 8 and the requirement to "go custom" quickly fades. It's unlikely that someone else hasn't already accomplished what you are trying to do in the world. So the question is why would anyone go custom? The common answers that I've heard over the past 13 years of working on nothing but Drupal include:

- We believe we're unique and our requirements are so complex that nobody else on the planet has even thought about what we're going to do. There are a couple of red flags in this statement. Our requirements are so complex and we're unique. Is the complexity a business requirement? If it is then the question becomes does the cost of custom development have a positive return for the organization?
- We believe we can write better code. That statement often comes from organizations whose IT organizations have held them hostage for decades. That's like saying that you can build a better car so instead of buying one off a dealer's showroom you're going to build your own. Building your own is costly, and the one responsible for maintaining it is the one who built it. There are thousands of amazing developers in the Drupal community who have built incredible modules. Why start from scratch?
- Off-the-shelf doesn't exactly fit our requirements. While every organization may be unique, I have yet to find a use case where when I truly understood the requirements I couldn't solve a majority of the requirement with one or more contributed modules. I have had to write some custom code to address very unique requirements, but the amount of custom code on any of the hundreds of projects that I've worked on over the past dozen years has been minimal.

While there may be some cases where it appears to require custom development, my suggestion is to:

- Clarify the requirement. Often requirements are vague and general. When you get to the bottom of what the organization is trying to achieve you can more often than not solve the problem with off-the-shelf solutions.

- Review the requirements with the stakeholders. More often than not, when I've discussed the requirements with stakeholders and explained that there is a way to accomplish a slightly revised version of the requirements with off-the-shelf Drupal modules, 99.99% of the time the stakeholders agree that the capabilities presented by an off-the-shelf solution are actually better than what they were envisioning, but they weren't clear themselves on what they wanted.
- Clearly communicate the cost and risk. All custom solutions come at a cost. When you identify a case for a purely custom solution, carefully calculate the true cost of developing that solution. In nearly every case over the past dozen years, the cost of custom development far outweighed the benefits of creating a custom solution versus bending the requirements so they fit an off-the-shelf solution. Remember that there are on-going costs beyond the initial development, and the burden of tracking security issues on your own versus leaning on the Drupal community.
- Enter the discussion early with stakeholders when planning your new Drupal site. When they understand the capabilities of the platform, they can then define requirements that fit Drupal's DNA, eliminating some if not all of the "square peg in a round hole" syndrome.
- If you find a case where it appears that custom is the only option, remember that Drupal plays well in an ecosystem of other applications, meaning that if the capabilities can be more easily met with a solution built on AngularJS, for example, then by all means build that capability in Angular and tie it into Drupal.
- If you have a use case that can almost be solved by one or more off-the-shelf modules, look at the issue queues to see if anyone else is suggesting the capabilities your organization is looking for. You may find others who would be willing to partner with you to enhance an off-the-shelf module to address all of your needs. Or you may be able to talk a module maintainer into helping you extend the capabilities of a contributed module. The key here is to communicate and ask.
- Don't "pave the cow path." I have encountered this mindset over and over again over the years. When looking at requirements, use cases, and designs I often find organizations trying to take what they have and re-platform it on Drupal. While it may seem like a valid approach, the reality is that in most "pave the cow path" scenarios, they fall far short of leveraging Drupal's capabilities to meet the business objectives, and you end up with a Frankenstein-like solution that performs poorly, is difficult to use, and ends up giving Drupal a bad reputation for not doing things as well as the old platform did. Get to the root of the business requirements and paint the solution using Drupal, instead of trying to "reskin" the old site using Drupal.

Every organization has to make the decision on their own as to how closely they want to fit within the off-the-shelf DNA of Drupal. Over the past decade I've watched organizations spend horrendous amounts of money developing highly custom solutions that performed poorly and ended up on the scrap heap. I've also watched organizations that have pivoted their belief systems and took an "off-the-shelf" only approach with the results being greater than they expected, simpler to maintain, and significantly less costly to build and maintain.

If I return back to the old days of "before-the-web" there were interesting statistics about the cost of building and maintaining systems. Typically we spend 80% of the total budget on less than 20% of the functionality we are trying to deliver. If you look at that 20% of the overall functionality that is so costly to build, it often has a less than 5% impact on revenue growth, profitability, competitiveness, brand loyalty, and customer satisfaction. Interestingly enough I've witnessed the same statistics over the past decade when it comes to Drupal sites.

Summary

There are many things to consider when optimizing your Drupal sites, but it often comes down to the basics of what are the true business requirements that you are trying to accomplish and how you can best leverage Drupal to address those needs. Do Drupal "the Drupal way" and you'll find yourself spending weekends and evenings doing the things you want to do, not battling to keep your sites alive.

The next chapter focuses on how to integrate Drupal with other systems, including creating a solution based on "headless" Drupal.