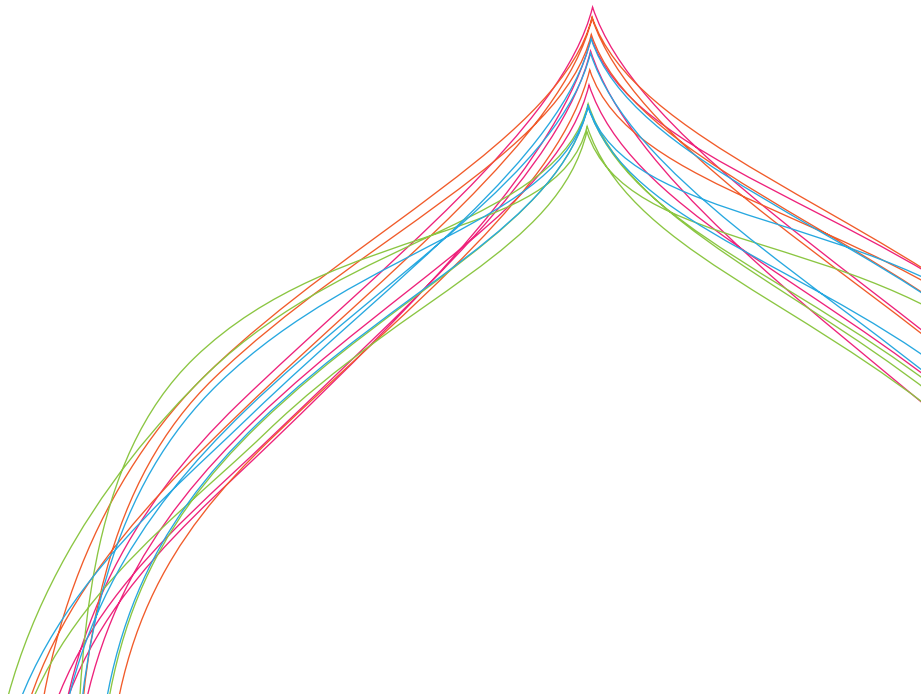
Acquia®

The Ultimate Guide to Drupal 8

Angela Byron, Director, Community Development, Acquia, Inc.



THE DIGITAL EXPERIENCE COMPANY

Table of Contents

Introduction

Whether you're a site builder, module or theme developer, or simply an end user of a Drupal website, Drupal 8 has tons in store for you. This ebook will enumerate the major changes in Drupal 8 for end users, for site builders, for designers and front-end developers, and for back-end developers.

Note that since Drupal 8 is still under active development, some of the details here may change prior to its release. Drupal 8 is now feature-frozen, so most information should remain relevant. Where applicable, Drupal 7 contributed equivalents of Drupal 8 features will be noted.

Angela Byron is an open source evangelist whose work includes reviewing and committing Drupal core patches, supporting community contributors, coordinating with the Drupal.org infrastructure team, and evangelizing Drupal. She is also a Drupal 8 core committer.

Angela is the lead author of O'Reilly's first Drupal book, entitled Using Drupal*. Angie is known as "webchick" on drupal.org.*

Authoring Experience

A major area of focus in Drupal 8 was around the out-of-the-box experience for content authors—the folks who actually use a Drupal website every day. Here are some of the changes you'll see.

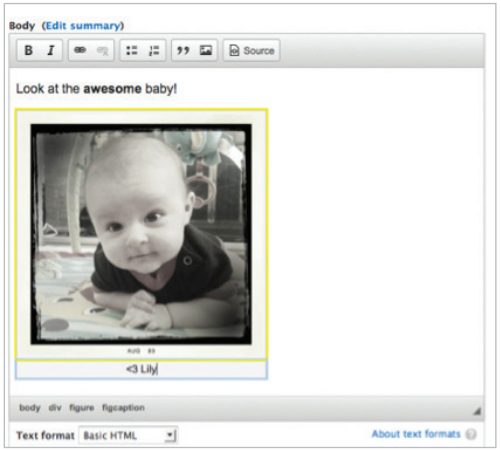
Spark

Spark is an Acquia initiative created by Dries Buytaert to **improve Drupal core's default authoring experience**. The Acquia development team for Drupal core performed analysis of both proprietary and open source competitors to Drupal and worked hard over the course of the release in collaboration with other Drupal core contributors. They helped make enhancements to Drupal core, all the while creating **back ports of key Drupal 8 UX improvements for Drupal 7** that can be used today.

WYSIWYG Editor

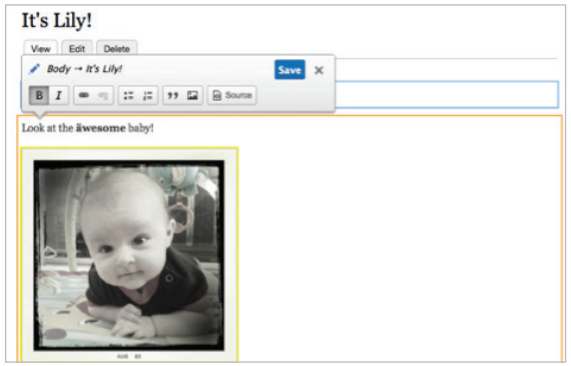
Gone are the days of typing HTML by hand (in the default install, at least). Drupal 8 ships with the **CKEditor** WYSIWYG editor. In addition to supporting what you'd expect in a WYSIWYG editor—buttons for bold, italic, images, links, and so on—it supports extras, such as easily editable image captions, thanks to CKEditor's new **Widgets** feature, developed specifically for Drupal's use. **Ensuring that we keep the structured content benefits of Drupal in our WYSIWYG implementation** was a priority.

Drupal 8 also sports a drag-and-drop admin interface for adding and removing buttons in the WYSIWYG toolbar, which automatically syncs the allowed HTML tags for a given text format, vastly improving usability. Buttons are contained in "button groups" with labels that are invisible to the naked eye, but that can be read by screen readers, providing an amazing, accessible editing experience for website visitors.



Drupal 8's Editor module wraps the WYSIWYG integration, so other libraries can be tightly integrated as well in contrib.

In-place Editing

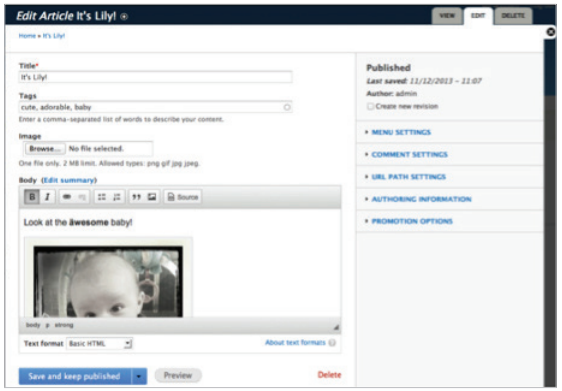


In Drupal 7, if you need to make a correction on a website—for example, a typo, or a missing image—you must use a back-end form, which is visually separated from the front-end website where content will appear. The Preview button doesn't help, because the results of preview are shown in the administrative theme (twice, in case you missed it the first time).

Drupal 8's new in-place editing feature allows editors to click into any field within a piece of content and edit it right on the front-end of the site, without ever visiting the back-end form. Full node content, user profiles, custom blocks, and more are all in-place editable as well.

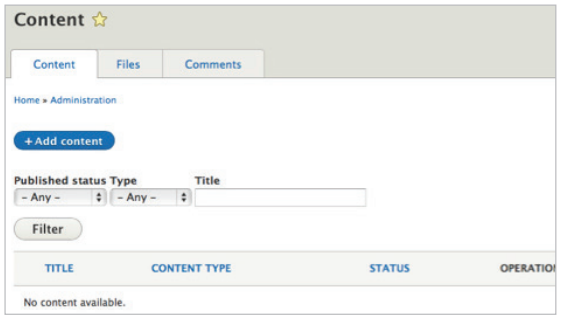
This in-place editing feature has been backported to Drupal 7 as the **Quick Edit** module (formally **Edit** module).

Redesigned Content Creation Page



A **community-led effort from Drupal's Usability team** resulted in a redesigned content creation page in Drupal 8. It contains two columns: one for the main fields (the actual "content" part of your content) and another for the "extras"—optional settings that are used less often. The hope is that the new design will create a less overwhelming experience for content authors and allow them to focus more on the task at hand.

Refreshed Admin Theme



Although still **undergoing development**, you'll find the administrative theme in Drupal 8 a refresh of Drupal 7's, with a **new style guide for the "Seven" admin theme**.

Draft Support in Core

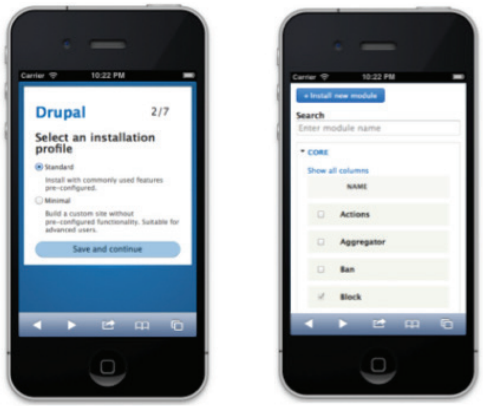
API support was added as an under-the-hood change in core to support content draft revisions. This change should make publishing workflow modules, such as **Workbench**, much easier in Drupal 8 and beyond.

Mobile Improvements

In addition to authoring experience improvements, another huge area of focus for Drupal website end users is on features that make Drupal 8 more mobile-friendly out of the box to keep up with a global explosion of mobile devices worldwide.

Mobile First

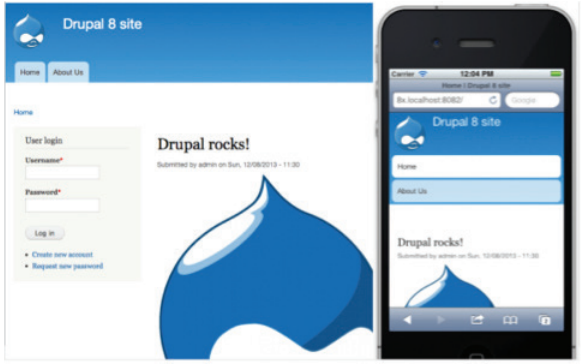
You'll find that Drupal 8 has been designed with mobile in mind, from the installer to the modules page. Even new features, such as in-place editing, are designed to work on teensy screens. Give Drupal 8 a try in your device of choice, and **let us know** what you think.



Also, you'll find a search box on the modules page. Check out **Module Filter** for a similar experience in Drupal 7.

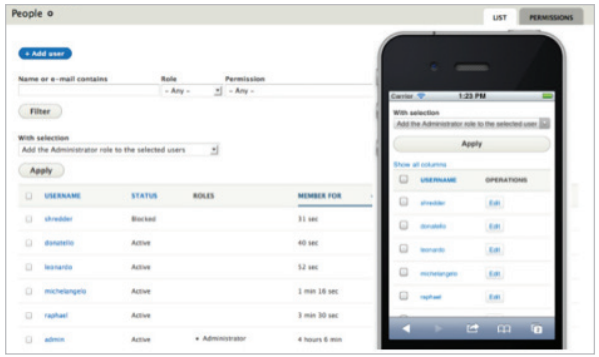
Responsive-ize ALL Things (Themes, Images, Tables...)

To support the unimaginable array of Internet-enabled devices coming in the next 5+ years, Drupal 8 incorporates responsive design into everything it does.



For starters, all core themes are now responsive and automatically reflow elements, such as menus and blocks, to fit well on mobile devices (if the viewport is too narrow, horizontal elements will switch to a vertical orientation instead). Images that show up large on a desktop shrink down to fit on a tablet or smartphone, thanks to built-in support for responsive images.

Drupal 8 also provides support for responsive tables, so table columns can be declared with a high, medium, or low importance. On wide screens, all the columns show, but as the screen size narrows, the less important columns start dropping off so everything fits nicely. This API is also built into the Views module, so you can configure your own responsive admin screens.

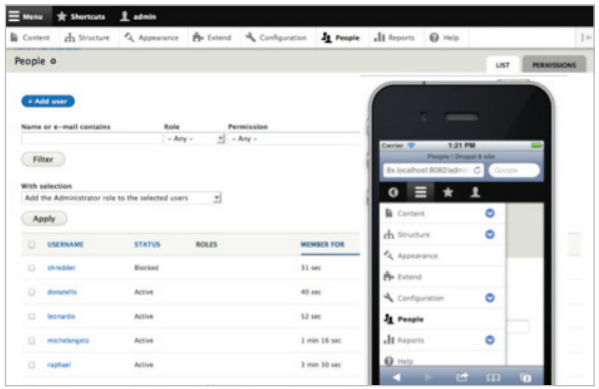


The **Responsive Bartik** and **Responsive Tables** modules can make Drupal 7 behave similarly. Numerous responsive base themes for Drupal 7, including **Omega** and **Zen**, help you build a responsive design for your website.

Mobile-friendly Toolbar

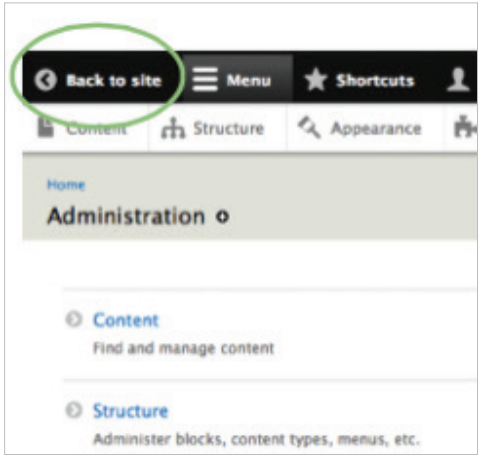
Drupal 8 now sports a shiny new administrative toolbar that automatically expands and orients itself horizontally on wide screens and collapses down to icons and orients vertically for smaller screens. Like all new front-end features in Drupal 8, this one got tons of accessibility love, so it's easy for screen reader users to jump around to various parts of the site.

If you're interested in this feature for Drupal 7, check out the **Mobile Friendly Navigation Toolbar** module.



Front-end Performance

One of the biggest factors that can make or break the mobile experience is the raw performance of a website. As a result, a lot of work went into minimizing Drupal 8's front-end footprint. In many cases, native JavaScript replaced jQuery, and out-of-the-box Drupal 8 loads zero JavaScript files for anonymous visitors. Additionally, lighter-weight alternatives that are mobile friendly replaced JavaScript-intensive features, such as the Overlay module: A simple "Back to site" link in the admin toolbar is visible while in an administrative context. See **Escape Admin** for a Drupal 7 equivalent.



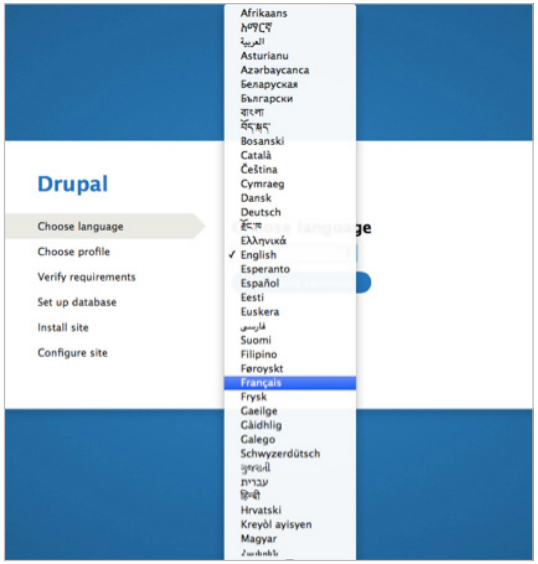
Multilingual++

The **Multilingual Initiative** (D8MI), led by Acquia's own **Gábor Hojtsy** with participation of over **1,000 contributors**, is a major development focus for Drupal 8. Check out Gábor's excellent **Drupal 8 Multilingual Tidbits** series if you're interested in all the details about D8MI.

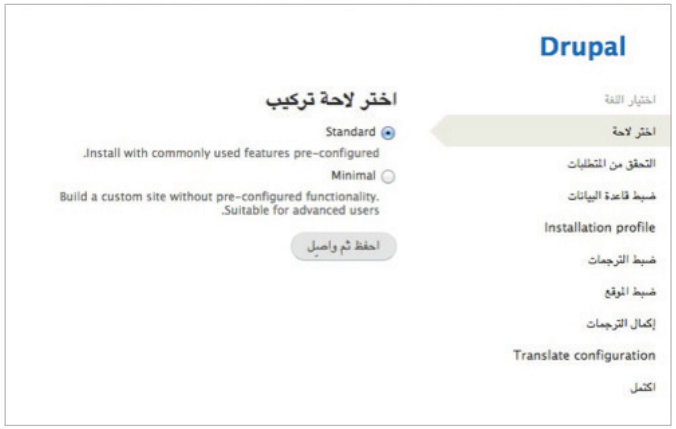
Multilingual First

You'll see Drupal 8's multilingual roots immediately at the beginning of the installer.

Drupal 8 auto-detects the language from your browser and then auto-selects that option in the drop-down for your convenience. But even better, if you install Drupal in a non-English language (or later add a new language to your site), Drupal 8 automatically downloads the latest interface translations from **https://localize.drupal.org**, so you can perform your entire site installation and setup in your native language.



This is in contrast to Drupal 7, which presented users with a wall of text (in English), explaining how to place downloaded files into particular directories in order to proceed.



This works for right-to-left languages, such as Arabic, too. (Drupal 8 is still in development, so some translations may be missing some strings.)

Drupal 8 does away with the concept of English as a "special" language. If you select a language other than English here, the English option will no longer show in your site configuration unless explicitly turned on. Also, you can make English "translatable" so that you can convert strings, such as "Log in / Log off" to "Sign in / Sign off."

Fewer Modules, Packing a Bigger Punch

Building a multilingual site in Drupal 7 requires about 30 contributed modules, and a lot of tricky configuration. In Drupal 8, all this functionality (and more) has been streamlined into just four modules, together making Drupal 8 more multilingual-friendly than all the Drupal 7 contributed modules combined.

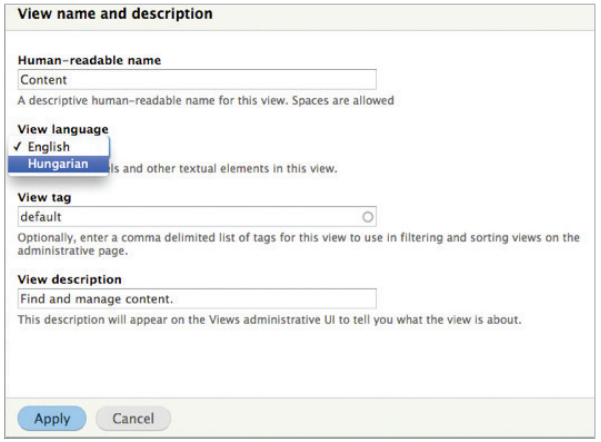
- **Language** provides Drupal 8's underlying language support. It is the base module and is required by the other multilingual modules.
- **Configuration Translation** makes things like blocks, menus, views, and so on, translatable. (Similar to Internationalization in Drupal 7).
- **Content Translation** makes things, such as nodes, taxonomy terms, and comments translatable. (It is not the same as core's Content Translation module in Drupal 7; it is much more akin to **Entity Translation**.)
- **Interface Translation** makes Drupal's user interface itself translatable. (It is the same as the Locale core module in Drupal 7.)

Why four modules and not just one, you ask? Because single-language, non-English sites are also a valid use case, and even multilingual sites may or may not need some of these features (for example, a desire to always keep user-generated content in its native language). This sort of granularity allows site builders to choose whatever combination meets their sites' specific use cases.

Language Selection on ALL Things

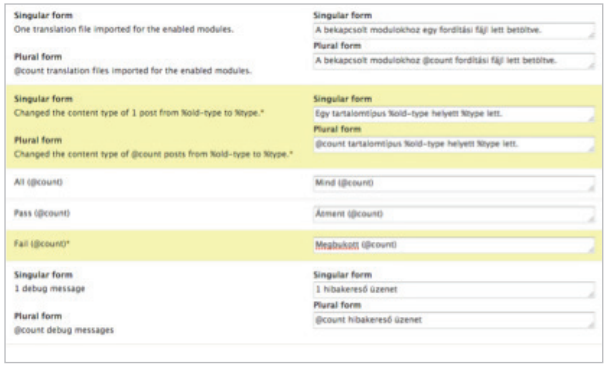
Everything from system configuration settings to site components, such as blocks, views, and menus, to individual field values on content are translatable.

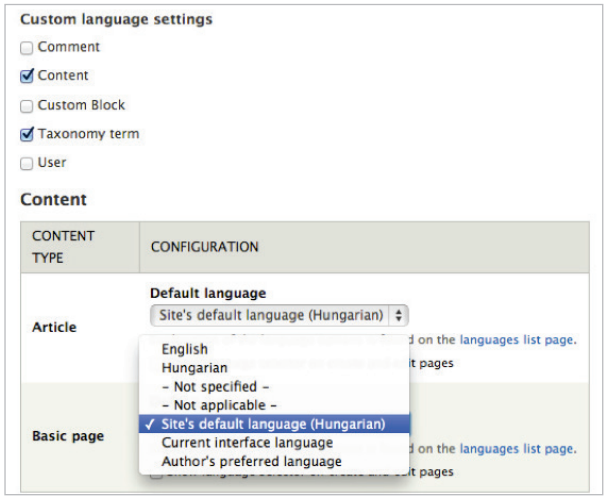
For content entities (comments, nodes, users, taxonomy terms, and so on), you have even more options, such as the ability to configure the visibility of the language selector, and whether newly created content defaults to the site's default language, the content author's preferred language, or some other value.



More Streamlined Translation UIs

Tons of effort went into improving the user experience of Drupal 8's multilingual functionality. You'll see much more streamlined translation and well-integrated interfaces throughout.





Transliteration Support

One really handy addition to Drupal 8 is the inclusion of the **Transliteration** module in core. It automatically converts special characters such as "ç" and "ü" to "c" and "u" for friendlier machine names, file uploads, paths, and search results.

...And More!

Here are some extras for site builders that are worth mentioning:

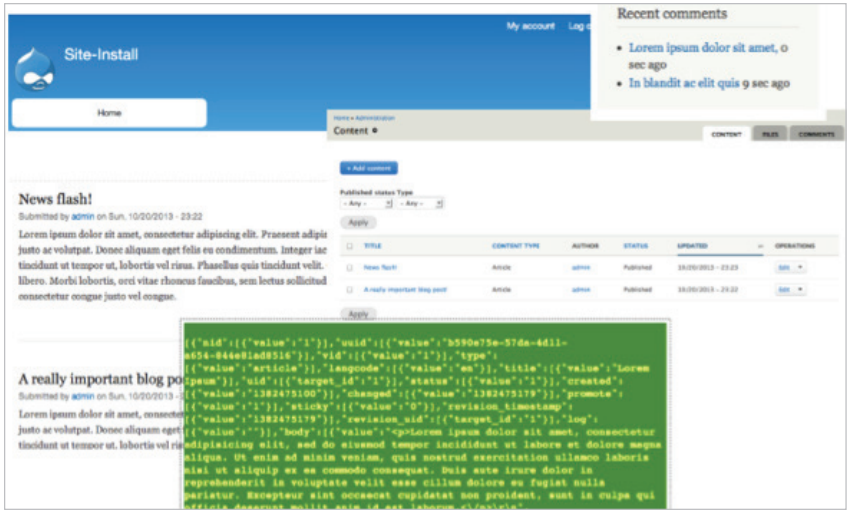
- Several of the pages in core that are using Views allow for much easier language-based customization, especially the admin views, where adding language filters, a language column, and so on, are easy to put together.
- Unlike the Drupal 7 Entity Translation module integration, Drupal 8 core's Content Translation module integrates well with Search in core, and search API gets more language information as well.
- The language selection system now supports a separate admin language, for easier management of multilingual sites for site admins.

Site Builders FTW

Although the **authoring experience improvements** and **mobile improvements** in Drupal 8 tend to focus on end users and content authors of Drupal websites, Drupal 8 also includes a huge push to improve the site building tools.

Views in Core!

The **Views** module, the **most frequently used module in Drupal contrib**, is now baked into Drupal 8 out of the box. Hooray! And not only is the Views module in core, but most of the main administrative listings such as Content, People, and Files, in addition to various sidebar blocks, several RSS feeds, and the default front page have also been converted to Views. This makes customizations of these elements—for example to add a "Full name" search to the People listing, or thumbnails next to items in the Content listing—just a few clicks away.



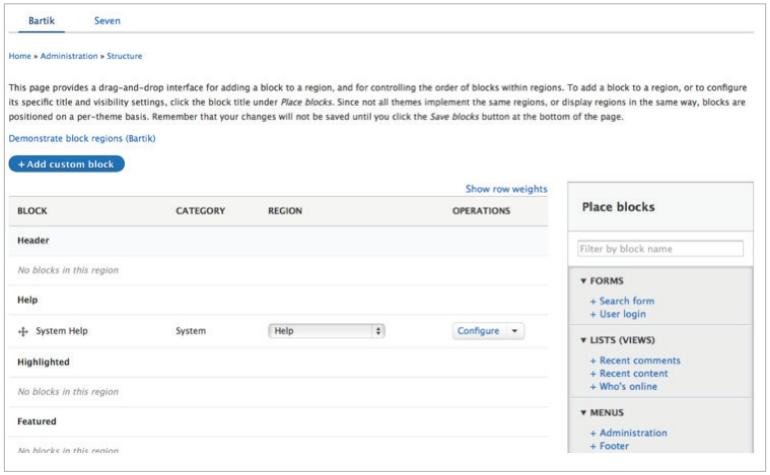
Everything you know and love from Views is included—and even a few extras such as mobile-friendly administration, some subtle user experience and accessibility improvements, the ability to create responsive table listings, and the ability to turn any listing into a REST export that can be consumed by a mobile application or other external service.

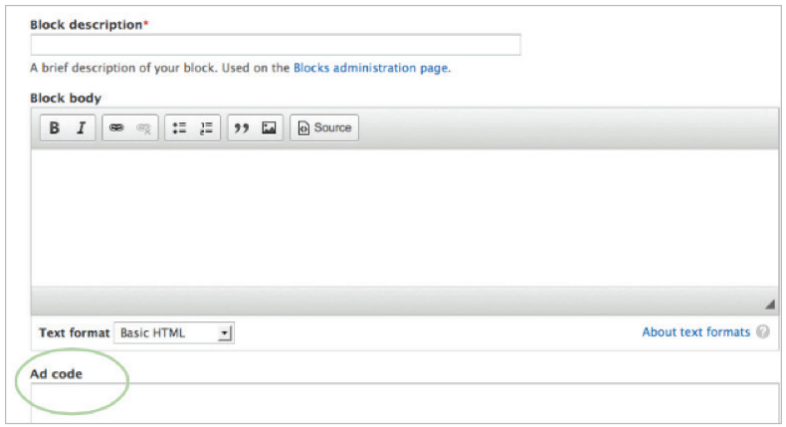
More and Better Blocks

In Drupal 8, you'll notice a few new features as they relate to blocks. First, just like with Views, several previously hard-coded site components have been converted to blocks, including breadcrumbs, site name, and slogan—with more in the works. This makes it easier to adjust page organization in the user interface.

Second, the limitation of being able to place a block into only one region is gone; you can place blocks and re-use them in multiple places, for example, a "Navigation" block in both the header and footer. No more need for the **MultiBlock** module!

And finally, you can now create custom block types, just as you can create custom content types, to allow for granular control over different styling, different fields, and more. This allows you to create, for example, an "Ad" block type with an "Ad code" field that can contain JavaScript snippets from a remote ad service.



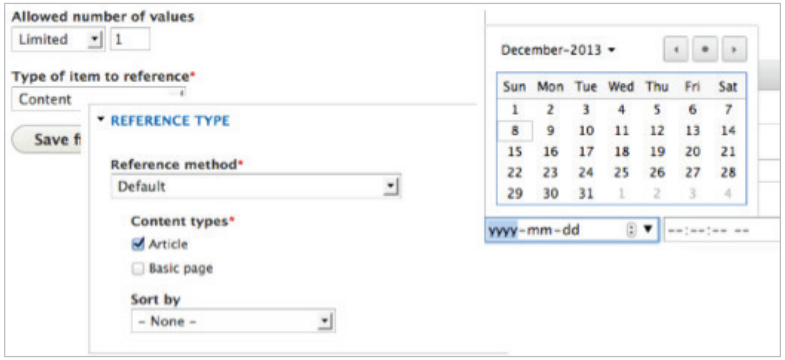


Improved and Expanded Entity and Field Features

Two of Drupal 7's most powerful site builder features—Entities and Fields—have been expanded in Drupal 8, making it easier than ever to build data models for the structured content you want to manage inside Drupal.

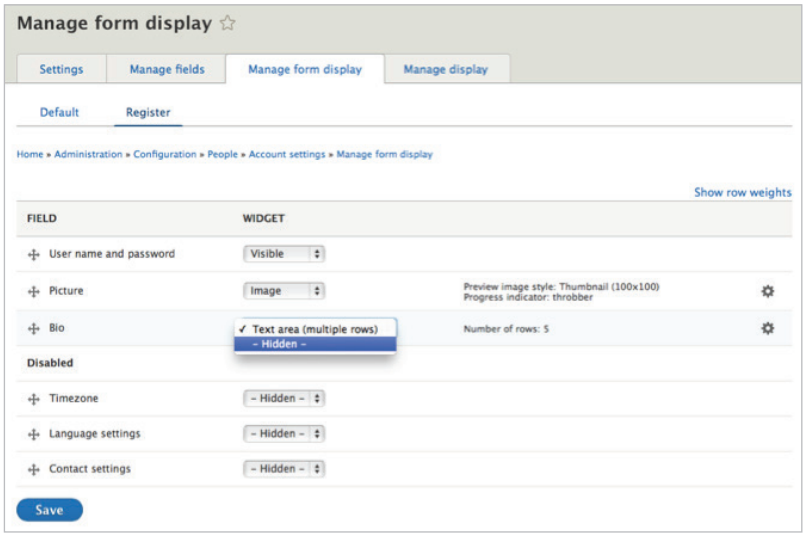
More Field Types

In addition to the Drupal 7 field types such as Taxonomy, Image, and File, Drupal 8 adds some powerful new fields such as Entity Reference and Date, along with commonly needed simple fields such as Phone, Email, and Link. Even the setting for whether comments are open or closed has been moved to a field, making any entity type commentable.



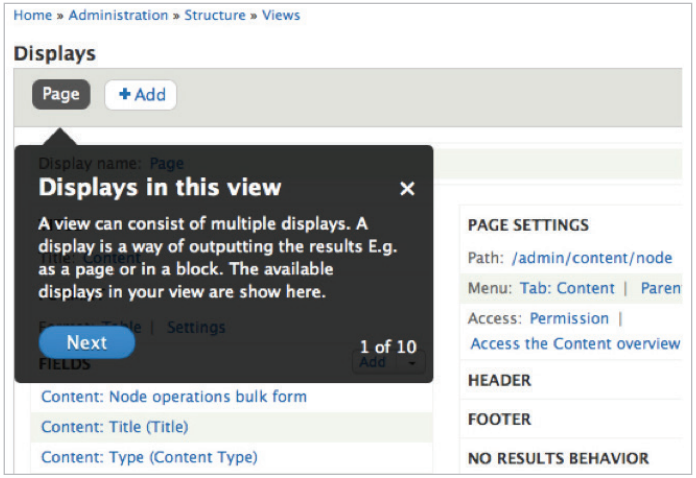
Form Modes

In addition to Drupal 7's "view modes" feature, which allows creating multiple display options for content in different contexts (for example, showing a thumbnail image on your content's teaser view and a full-size image on default view), Drupal 8 adds the notion of "form modes" to do the same for data-entry forms. Here's an example of configuring the user registration form differently from the user edit form, so you can hide the more esoteric fields and provide a simpler user experience.



Take a Tour

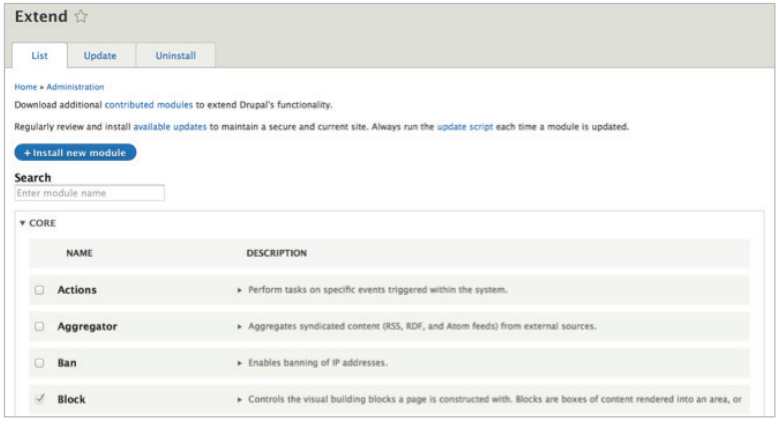
Drupal 8's new tour module gives site builders contextual, step-by-step tooltips with overviews of its administrative interfaces, help to introduce new terminology, and walk through the steps involved in configuring components of the site.



Both Less and More, Module-wise

You'll find Drupal 8 missing some modules that shipped with Drupal 7, namely Blog, Dashboard, Open ID, Overlay, PHP filter, Poll, Profile, and Trigger (as well as the Garland theme). You'll find several new modules in which functionality has been split out into more granular chunks, such as Menu Links/Menu UI, Block/Custom Block, Ban/History/Actions (formally baked into User/Node/System module), and so on.

Heather James's **"Drupal 8 Site Building Preview—Less Is More"** has a great rundown of the state of modules, including contrib modules that are now rendered obsolete due to the functionality that ships with Drupal 8 core.



The bottom line: Drupal core will ship with enough functionality out of the box that for the first time site builders should be able to create fairly sophisticated sites without having to install 30+ contributed modules. Hooray!

Migration Path

Although the UI is not yet in Drupal 8 core, the major version upgrade path has been replaced with a migration path, courtesy of a D8 port of the **Migrate** and **Migrate Drupal-to-Drupal** modules. Both a migration path from Drupal 6 (already in Drupal 8.x) and Drupal 7 (under active development) will be supported in Drupal 8's final release. The main difference to you as a site builder is that instead of keeping your site offline for hours while a variety of scripts attempt to upgrade your production database schema in-place, you'll keep your Drupal 6/7 site running while you build the new Drupal 8 site and keep running the migration path (provided by core/contrib/custom modules) until everything is moved over satisfactorily—doing a simple webroot/DNS swap at the end with next to zero downtime.

For more on Drupal 8's improved major version upgrade process, check out Moshe Weitzman's **"Drupal 8—Improved Upgrade Process"** blog from December 2013.

Front-end Developer Improvements

Drupal 8 contains a lot of nifty front-end developer improvements, including HTML5, libraries, accessibility enhancements, new themes and UI elements, and faster performance to name a few.

HTML5

All of Drupal's output has been converted to use semantic HTML5 markup, as opposed to XHTML in Drupal 7. This means you'll find tags such as `<nav>`, `<header>`, `<main>`, and `<section>` in Drupal's default templates, as part of an overarching effort to clean up Drupal's default markup.

HTML5 also brings new form input types, such as `date`, `tel`, and `email`, that can provide targeted user interfaces on mobile devices (for example, only showing the number pad when entering phone numbers) to help streamline data entry. Drupal's Form API provides these additional types so you can easily create these new types of fields. The Drupal 7 equivalent can be found in the **Elements** module.

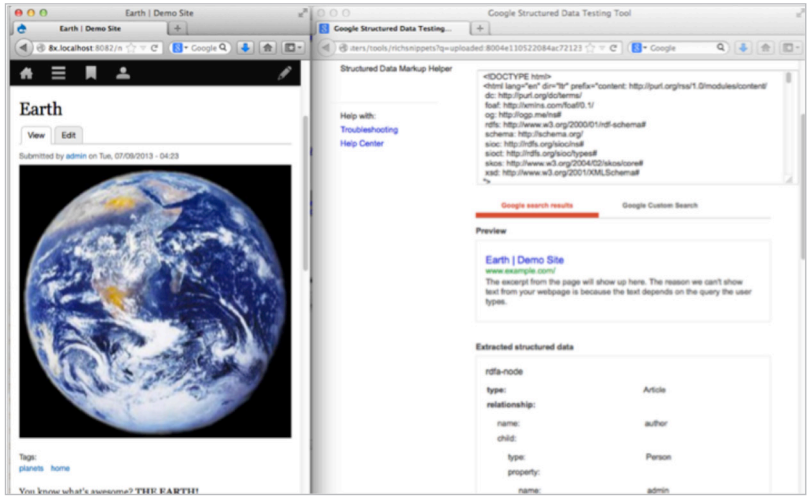
Additionally, you'll find HTML5/CSS3 replacements for several things that previously needed custom workarounds: resizing on text areas and first/last/odd/even classes, replaced by CSS3 pseudo selectors; and collapsible fieldsets largely replaced by the `<details>` element.

New Front-end Libraries and Helpers

Although Drupal has shipped with jQuery since version 5 and shipped with jQuery UI since Drupal 7, Drupal 8 brings with it an expanded array of front-end libraries—for example, **Modernizr** (which makes it easy to detect if a browser supports touch or HTML5/CSS3 features), **Underscore.js** (a lightweight JS helper library), and **Backbone.js** (a model-view-controller JavaScript framework). Together, these additional libraries allow for creating mobile-friendly, rich front-end applications in Drupal, and they're used by several of the **Authoring Experience** and **Mobile** feature improvements to Drupal 8.

Native Schema.Org Output

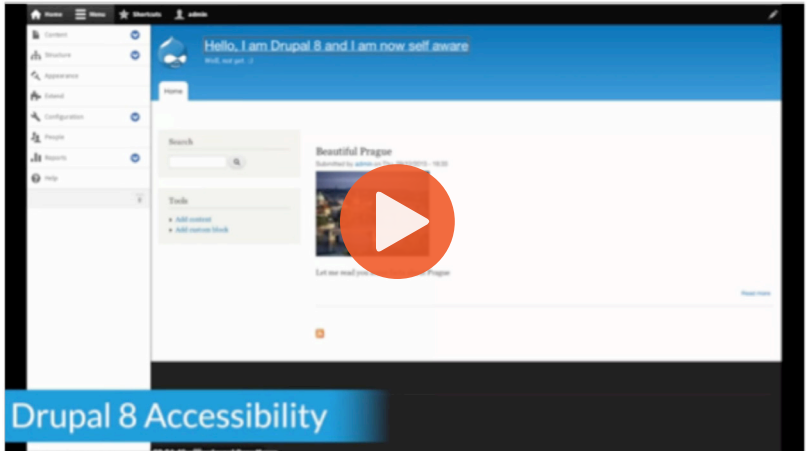
In a great boon for search-engine optimization, Drupal 8's RDFa module now outputs **schema.org** markup. This makes the task much easier for search engines such as Google, Yahoo!, Bing, and Yandex to extract data, such as who the author of a given piece of content is, in order to add meaning behind the content.



Even More Improved Accessibility

Drupal 8 has expanded on **Drupal 7's existing stellar accessibility record** with even more improvements. Drupal 8 extensively uses **WAI-ARIA attributes** to provide meaning on rich, front-end applications, such as the in-place editor and responsive toolbar. On the back-end, Drupal 8 provides a variety of **new Accessibility tools for JavaScript** (JS), which allows module developers to create accessible applications easily. An ongoing effort seeks to **provide automated testing for accessibility features via the Quail library**.

The following **video**, extracted from **Dries's DrupalCon Prague Keynote**, demonstrates how these new accessibility features appear to assistive technology users.



New Theme System: Twig

Drupal 8 introduces **Twig**, which takes the place of the PHPTemplate-based theme system in Drupal 7 and below. Twig, like many similar templating engines from other projects, allows designers with HTML/CSS knowledge to modify markup without needing to be a PHP expert. For example, instead of needing to understand the syntax differences between deeply-nested arrays versus objects and when to use each, a simple `{{ foo.bar }}` statement does the trick. Simple conditional and looping logic can be contained in `{% ... %}` tags.

The following is an excerpt from page.html.twig (the equivalent of page.tpl.php in Drupal 7), showing off some Twig features and some HTML5 tags and native ARIA support as well:

```
<?php
  <main role="main">
    <a id="main-content"></a>{# link is in html.html.twig #}

    <div class="layout-content">
      {{ page.highlighted }}

      {{ title_prefix }}
      {% if title %}
        <h1>{{ title }}</h1>
      {% endif %}
      {{ title_suffix }}

      {{ tabs }}

      {% if action_links %}
        <nav class="action-links">{{ action_links }}</nav>
      {% endif %}

      {{ page.content }}

      {{ feed_icons }}
    </div>{# /.layout-content #}

    {% if page.sidebar_first %}
      <aside class="layout-sidebar-first"
       role="complementary">
        {{ page.sidebar_first }}
      </aside>
    {% endif %}

    {% if page.sidebar_second %}
      <aside class="layout-sidebar-second"
       role="complementary">
        {{ page.sidebar_second }}
      </aside>
    {% endif %}

  </main>
?>
```

How do you provide those variables if you can no longer use PHP in templates directly? With `THEME_preprocess_HOOK()` functions, you do it the same way you've always done (although they are in a file named `THEME.theme` instead of `template.php`). Twig effectively forces a separation of presentation and business logic, and all variables going into template files are automatically escaped, far reducing the risk of dangers like XSS vulnerabilities and making theming in Drupal 8 more secure than ever before.

Another nice tidbit from Twig is that if you turn on debug mode using `debug: true;` in your site's `services.yml` file, helpful code comments will be displayed throughout Drupal's generated markup to inform you where to find the template for the markup you're trying to change, and which particular "theme suggestion" is being used to generate the markup. For example:

```
<div class="content">

<!-- THEME DEBUG -->
<!-- THEME HOOK: 'node' -->
<!-- FILE NAME SUGGESTIONS:
   * node--1--full.html.twig
   * node--1.html.twig
   * node--article--full.html.twig
   * node--article.html.twig
   * node--full.html.twig
   x node.html.twig
-->
<!-- BEGIN OUTPUT from 'core/themes/bartik/templates/
 node.html.twig' -->
<article data-history-node-id="1" data-quickedit-entity-
 id="node/1" role="article" class="contextual-region node
 node--type-article node--promoted node--view-mode-full
 clearfix" about="/node/1" typeof="schema:Article">
   ...
</article>

<!-- END OUTPUT from 'core/themes/bartik/templates/node.
 html.twig' -->

</div>
```

It's a bit like having the fabulous **Theme developer** module baked into core!

Fast by Default

Acquia's own llama-loving performance guru Wim Leers posited that the best way to make the Internet as a whole faster is to **make the leading CMSes fast by default**. This means that CMSes need to enable their high-performance settings out of the box rather than require users to be savvy enough to find them in all their various locations. And in Drupal 8, that's exactly what we've done. You'll notice that Drupal 8 ships with features such as CSS and JavaScript aggregation already turned on for a much faster default installation. Huzzah!

What this means to you as a front-end developer is that by default Drupal is not immediately in a good place to start theming, unless you manually turn off those performance settings one by one (even hacking core's CSS directly will show absolutely no changes). Fortunately, Drupal 8 ships with a `sites/example.settings.local.php` file for exactly this purpose. It hard codes the performance settings to off, which is extremely useful in a development environment. Simply copy it, rename it as `sites/default/settings.local.php`, and uncomment the following lines in `sites/default/settings.php`:

```
<?php
# if (file_exists(__DIR__ . '/settings.local.php')) {
#   include __DIR__ . '/settings.local.php';
# }
?>
```

Your new `settings.local.php` file points to `development.services.yml`, which contains some disabled-by-default settings about Twig specifically, for example ones for turning on debug mode and turning off caching. Changing these settings to `true` will definitely make your dev site slower but will also make theming much easier, because you'll be able to see the results of your changes to Twig templates immediately, without having to clear the cache.

In other front-end performance-related news, while Drupal 8 will still ship with the latest versions of jQuery and jQuery UI, a lot of movement is going away from using libraries like this for run-of-the-mill JavaScript to keep front-end performance as quick as possible, which is especially important for mobile devices. The default install of Drupal 8 actually loads zero JavaScript for anonymous users!

Although more work needs to be done on performance optimizations, once it ships, Drupal 8 should provide a much faster front-end experience for site visitors. Hooray!

New UI Elements

Drupal 8 ships with several new UI elements that you can use in your own admin screens, including modal dialogs and drop buttons, which were part of the **Chaos tool suite (ctools)** module in Drupal 7 and below. Drupal 8 introduces the concept of "button types," "primary" (the default form action; in Seven theme styled as blue), and "danger" (styled as red links) to help users quickly make correct choices when confronted with multiple options on a form.

Theme Responsively

As mentioned in the **Mobile Improvements** article, Drupal 8 ships with numerous new responsive features, including responsive themes, toolbar, images, and tables.

To support these features, themes can now declare **Breakpoints** (the height, width, and resolution at which a design changes to meet shifting browsers and devices) that can be used by responsive features. (However, note that **Move breakpoint settings to theme and module *.info.yml files** is a patch actively being worked on that proposes changing the exact implementation.)

It's also **looking as though Drupal 8** will ship with support for **the new** `<picture>` **element**, which browers have started to support. This will make for a significant front-end performance improvement, particularly on mobile devices, because it allows delivering smaller images (typically the heaviest part of any page load) for smaller screens, saving data. (Thanks to **Marc Drummond** for this information.)

New Method of Selectively Adding JS/CSS to the Page

Also on the performance front—in the past, if you wanted to add CSS or JS to a particular page, you'd use the `drupal_add_css()` and `drupal_add_js()` functions, respectively. Not anymore! You now insert any JS/CSS assets in the `#attached` property of a render array. For example:

seven.theme

```
function seven_form_node_form_alter(&$form, &$form_state) {
...
  $form['#attached'] = array(
    'css' => array(drupal_get_path('module', 'node') . '/css/
    node.module.css'),
  );
...
}
```

Although this will work OK for one-off assets that don't have any dependencies, the more common and recommended approach is to register one or more JS/CSS assets (along with their dependencies) as a library in your `MODULE/THEME.libraries.yml` and then add a reference to the library in the `#attached` property. For example:

seven.libraries.yml

```
maintenance-page:
 version: VERSION
 js:
   js/mobile.install.js: {}
 css:
   theme:
     maintenance-page.css: {}
 dependencies:
   - system/maintenance

install-page:
 version: VERSION
 js:
   js/mobile.install.js: {}
 css:
   theme:
     install-page.css: {}
 dependencies:
   - system/maintenance

drupal.nav-tabs:
 version: VERSION
 js:
   js/nav-tabs.js: {}
 dependencies:
   - core/matchmedia
   - core/jquery
   - core/drupal
   - core/jquery.once
   - core/jquery.intrinsic
```

seven.theme

```
<?php
function seven_preprocess_install_page(&$variables) {
  // ...
  $libraries = array(
    '#attached' => array(
      'library' => array(
        'seven/maintenance-page',
        'seven/install-page',
      ),
    ),
  );
  drupal_render($libraries);
}
?>
```

Although this isn't quite as convenient as a quick in-line call to `drupal_add_FOO()`, it does mean that these assets are now cacheable for improved performance, and easily re-usable among different parts of the code base.

R.I.P. IE 6, 7, and 8

On a bit of a melancholy note, the last big improvement for front-end developers is that at last, in a move applauded by web designers everywhere, Drupal 8 core has officially dropped support for IE 6, 7, and 8, enabling the use of jQuery 2.0 and other code that assumes modern HTML5/CSS3 browser support. (Note there's also talk of **dropping support for Android 2.3 and below** for the same reason.)

As a parting gift, **html5shiv** (an HTML5 polyfill for less capable browsers) is included in D8 core so at least IE 8 and below aren't completely ignored, and the **IE8** project in contrib is available for those who absolutely must have IE8-compatible versions of core front-end features. For the rest of us, we're looking forward to snappier front-end code that doesn't have to worry about limitations in 5+ year old browsers. Hooray!

And More?

Because Drupal 8 is still under active development, some aspects of the APIs are not nailed down yet (and markup doesn't freeze until RC). Here are a few of the big remaining front-end efforts out there, which could use code and reviews:

- Althought not complete yet (hint: Here's a great way to learn Twig), there is an ongoing effort to **convert all core theme functions to Twig templates** so that theming works the same way no matter what is output on the page, except in very rare cases where theme functions are still needed for performance. This should make theming much more approachable, because it would negate the requirement for designers to learn PHP to make trivial markup alterations in almost all cases.
- **Convert page elements (title, tabs, actions, messages) into blocks** and **Move `menu_block` module functionality into core** would get rid of one-off variables in favor of consistent placement/theming (as well as caching) of page elements everywhere as blocks.
- The **Dream Markup** movement arose to strip all the nasty cruft (extra `<div>`s and whatnot) from Drupal's markup. This movement just got some fresh interest poured into it at DrupalCon Austin, resulting in **a proposal to strip down all Drupal core's default markup** to remove any extraneous cruft, and provide a base theme with helpful classes/wrappers for equivalence with the current status quo. Interesting times.
- Also out of DrupalCon Austin, the **Headless Drupal** group formed out of a desire to make it easier to use completely custom front-ends, for example in Angular JS on top of Drupal.

Back-end Developer Improvements

Drupal 8 gives you lots of back-end developer improvements, including an API for configuring your system. All entities are now classed as objects. You also get improved caching, better integration with third-party services, and lots of built-in web services features. It just keeps getting better.

New Configuration Management System

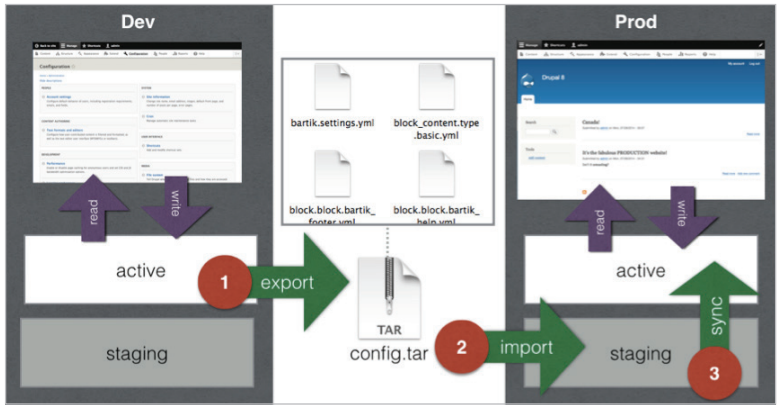
Probably the most looked-forward-to change in Drupal 8, for both developers and site builders, is the new configuration management system. In Drupal 7 and below, both content and configuration were saved to the database (sometimes with a mix of both in the same table), making deploying configuration changes from one environment to another (for example, development to production) very tricky. A variety of workarounds emerged for this, including `hook_update_N()`, **Features module**, and of course the old standby: carefully writing the configuration changes you made in development on a napkin and then manually repeating them in production. However, all these were attempting to circumvent the fundamental problem that Drupal core didn't properly support configuration deployment natively—until Drupal 8, that is.

In Drupal 8, all configuration changes (both standard admin settings forms, such as site name, as well as any **ConfigEntity** including Views, user roles, and content types) run through a unified **configuration API**. Each environment has both an "active" store (where configuration settings are written to and read from on every page load) as well as a "staging" store to hold configuration changes from other environments that are about to be imported for review. For performance, the active store is in a config table in the database (somewhat analogous to the variables table in Drupal 7 and below)—though the storage location is swappable. The **Configuration Development module**, for example, writes active configuration out to YAML files in the file system, just as core does with the staging store.

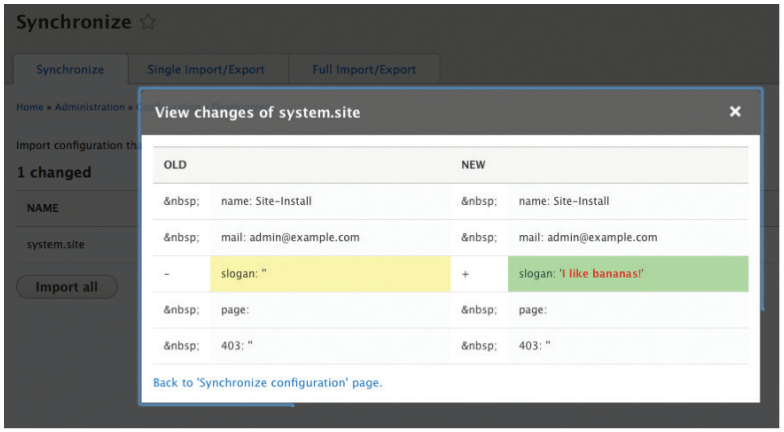
Drupal 8 also ships with a basic UI to do both single and full configuration imports and exports, and configuration can also be moved around via the command line with **Drush's `config-*` commands**, which is handy when using version control systems such as Git.

The basic workflow (after making whatever configuration changes to your Drupal 8 site) is:

1. On the development site, export your site's "active" configuration. You'll receive a tarball that consists of lots of YAML files.
2. On production, import the files, which places them into the config "staging" area.
3. In the configuration UI, view the list of what configuration settings have changed and view a "diff" of changes in advance.
4. If changes are acceptable, synchronize them, which will replace production's current active store with the contents of staging and become the new values that Drupal will use to build pages.



Of course, there are some settings that are specific to a given environment and that you don't want to be deployed across environments. One such example is a timestamp storing the last time cron ran. For these, there's a "sister" API to the configuration API named the **State API** for these more ephemeral settings.



What About Content Deployment?

While Drupal 8 core doesn't ship with support for migrating content such as nodes, users, and taxonomy terms between sites (although this could happen in a later feature release such as 8.1.0 or 8.2.0), one welcome addition to Drupal 8 has been the introduction of UUIDs (universally unique identifiers) to every piece of content, such as b2423870-b19b-45e7-8407-076aee906870. These UUIDs can be used to determine whether a piece of content exists on a given destination site, regardless of whether the content's numeric ID conflicts, making content imports/exports infinitely easier. Keep your eyes on the **Deploy module** for a **Drupal 8 version** that provides this feature. If you're still on Drupal 7, you can get similar functionality to what core offers via the **Universally Unique IDentifier module**.

Entities, Entities, Everywhere!

Entities were a key new feature and concept in Drupal 7, abstracting the ability to add fields to other types of content than just nodes, such as users and taxonomy terms. However, the Drupal 7 core API was severely limited and required using modules, such as the **Entity API module,** to further flesh out basic functionality, such as saving and deleting.

In Drupal 8, the Entity API has been completely rehauled to not only fill the gaps in functionality from Drupal 7 but also to greatly improve developer experience. All entities are now classed objects that implement a standard **EntityInterface** (no more guessing which of the 100 entity hooks you're required to implement), with baked-in knowledge about the active language. Compare and contrast:

```
<?php
# Drupal 7 code.
$node->title
$node->body[$langcode][0]['value']
```

with

```
# Drupal 8 code.
$node->get('title')->value
$node->get('body')->value
?>
```

Nearly anything you can create more than one of has been converted to an entity, bringing greater consistency to Drupal development. There are two kinds of these entities: Config entities and Content entities. What's the difference?

Content entities also sport some nifty new features compared to Drupal 7, such as revisions on not just nodes but also custom blocks and the ability to add comments to any content entity (you can even have comments on comments). The **"Site Builder Improvements"** article has more information about other entity-related features.

Content Entities

- Customized fields
- Stored in database tables (by default)
- Mostly created on front-end

Examples

- Nodes
- Custom Blocks
- Users
- Comments
- Taxonomy Terms
- Menu Links
- Aggregator Feeds/Items

Config Entities

- Deployed to different environments
- Stored in configuration system
- Mostly created on back-end

Examples

- Content Types
- Custom Block Types
- User Roles
- Views
- Taxonomy Vocabularies
- Menus
- Image Styles

Wither hook_schema()?

What does this mean for you as a developer? It means that between the Entity API and the Configuration/State API, there is almost never a reason to create and manage your own database tables by hand in Drupal 8. By using these standard APIs, you'll benefit from writing less brittle code and benefit from portability to other databases such as MongoDB.

Web Services

A major focus for Drupal 8, both to enable the creation of Drupal-backed mobile applications and to facilitate cross-site communication and better integration with third-party resources, is a native REST API built into Drupal 8 and provided by the **RESTful Web Services** suite of modules. This allows for fine-grained configuration of which resources should be available (nodes, taxonomy, users, and so on), what HTTP methods are allowed against those resources (for example, GET, POST, PATCH, DELETE), and which formats and authentication are used to access those resources. See the contributed **REST UI module** that provides an interface for this configuration. For each allowed HTTP method, you can set permissions on which role(s) on your site may access the resources via that method. This allows anonymous users to GET but only administrators to POST, for example.

Once the various RESTful Web Services modules are configured properly, you can get a big clump of machine-readable data representing your site content, such as

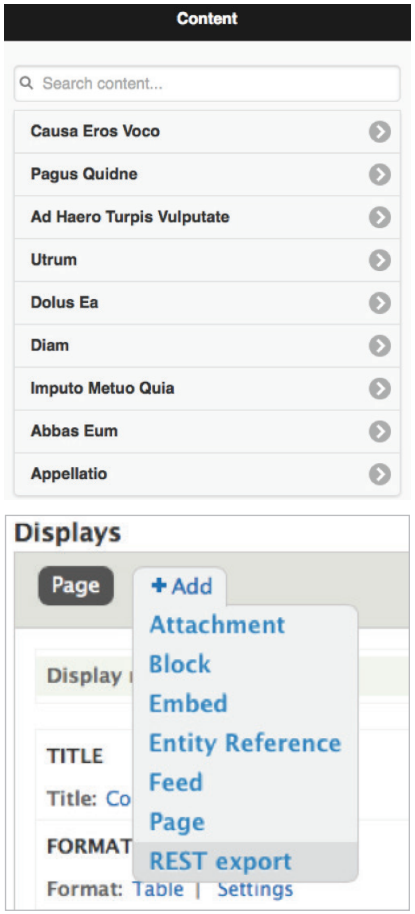
```
...
   [title] => Array
    (
        [0] => Array
            (
                [value] => Hello, world!
                [lang] => en
            )

    )
...
[body] => Array
    (
        [0] => Array
            (
                [value] => <p>This is my <strong>awesome
                </strong> article.</p>
                [format] => basic_html
                [summary] =>
            )

    )
...
```

What good is that? Plenty! Here's one example of retrieving information from Drupal 8 in JSON and displaying it in a standalone **jQuery Mobile app**.

Drupal 8 ships with a new library named **Guzzle** with **easy syntax** to retrieve and post data to Drupal or to talk to third-party Web Services, such as Twitter or Github.

Another Web Services feature in Drupal 8 offered by the RESTful Web Services module is the ability to add a “REST export” display to any view.

This means you can easily create JSON or XML feeds of custom dynamic content from your Drupal site, just by clicking them together!



Improved Caching

And on a final happy note, caching in Drupal 8 has been improved across the board.

- **Entity cache** module is now in core.
- **Cache tags** allow for much more granular cache clearing when content or settings are updated on the site.
- All caching features such as CSS/JS aggregation are turned on out of the box, making Drupal 8 **fast by default**.
- While we’re still working hard on **improving D8’s performance overall**, this extra caching should help most page loads come up lickety-split.

Better, Right Down to the Core

Drupal 8 made some major API changes that embrace the way the rest of the world works.

"Proudly Found Elsewhere"

As a counterpoint to **"Not Invented Here"**, "Proudly Found Elsewhere" represents a mind-shift among Drupal core developers to find the best tool for the job and incorporate it into the software, versus creating something custom and specific to Drupal, which only we benefit from.

You'll see this philosophy change in many aspects of Drupal 8. Among the external libraries we've pulled in are **PHPUnit** for unit testing, **Guzzle** for performing HTTP (web service) requests, a variety of **Symfony** components (**Create your own framework on top of the Symfony2 Components** is an excellent tutorial for learning more about those), and **Composer** for pulling in external dependencies and class autoloading, and more.

But this philosophy change also extends to the code base itself. We made **big architecture changes** in Drupal 8 to embrace the way the rest of the world is writing code: decoupled, object-oriented (OO), and embracing modern language features of PHP, such as namespaces and traits.

Getting OOP-y with It

Let's look at a few code samples to illustrate Drupal 8's "Proudly Found Elsewhere" architecture in action.

Drupal 7: example.info

```
name = Example
description = An example module.
core = 7.x
files[] = example.test
dependencies[] = user
```

All modules in Drupal need a `.info` file to register themselves with the system. The example above is typical of a Drupal 7 module. The file format is "INI-like" for ease of authoring, but also includes some "Drupalisms" such as the `array[]` syntax so standard PHP functions for reading/writing INI files can't be used. The `files[]` key, which triggers Drupal 7's custom class autoloader to add code to the registry table, is particularly Drupalish, and module developers writing OO code must add a `files[]` line for each file that defines a class, which **can get a little bit silly**.

Drupal 8: example.info.yml

```
name: Example
description: An example module.
core: 8.x
dependencies:
  - user
# Note: New property required as of Drupal 8!
type: module
```

In embracing "Proudly Found Elsewhere," info files in Drupal 8 are now simple **YAML** files—the same as those used by other languages and frameworks. The syntax is very similar (mostly `:` instead of `=` everywhere, and arrays formatted differently), and it remains very easy to read and write these files. The awkward `files[]` key is gone, in favor of the **PSR-4** standard for automatic class autoloading via **Composer**. The English version of that sentence is that by following a specific class naming/folder convention (modules/example/src/ExampleClass.php), Drupal can pick up OO code automatically without requiring manual registration.

Drupal 7: hook_menu()

example.module

```
<?php
/**
 * Implements hook_menu().
 */
function example_menu() {
  $items['hello'] = array(
    'title' => 'Hello world',
    'page callback' => '_example_page',
    'access callback' => 'user_access',
    'access arguments' => 'access content',
    'type' => MENU_CALLBACK,
  );
  return $items;
}

/**
 * Page callback: greets the user.
 */
function _example_page() {
  return array('#markup' => t('Hello world.'));
}
?>
```

This is a pretty basic "hello world" module in Drupal 7, which defines a URL at `/hello` that when accessed checks to make sure the user has "access content" permissions before firing the code inside `_example_page()` which prints "Hello world." to the screen as a fully themed page. The `hook_menu()` is an example of what is pejoratively known as an ArrayPI, a common pattern in Drupal 7 and earlier. The problem with ArrayPIs is that they are difficult to type (for example, have you ever forgotten the `return $items` and then spent the next 30 minutes troubleshooting a problem?), have no IDE autocompletion for what properties are available, and the documentation must be manually updated as keys are changed and added. The **documentation for** `hook_menu()` shows that it also suffers from trying to do too many things. It's used for registering path-to-page/access callback mappings, but it's also used to expose links in the UI in a variety of ways: swapping out the active theme, and much more.

Drupal 8: Routes + Controllers

example.routing.yml

```
example.hello:
  path: '/hello'
  defaults:
    _content: '\Drupal\example\Controller\Hello::content'
  requirements:
    _permission: 'access content'
```

src/Controller/Hello.php

```
<?php
namespace Drupal\example\Controller;

use Drupal\Core\Controller\ControllerBase;

/**
 * Greets the user.
 */
class Hello extends ControllerBase {
  public function content() {
    return array('#markup' => $this->t('Hello world.'));
  }
}
?>
```

In Drupal 8's new **routing system**, the path-to-page/access-check logic now lives in a YAML file using the same syntax as the **Symfony routing system**. The page callback logic now lives in a "Controller" class (as in the standard **model-view-controller** pattern) in a specially named folder, per the PSR-4 standard. It's declared in the example module's namespace to allow the example module to name its classes whatever it wants without worry of conflicting with other modules that might also want to say Hello (Drupal is very friendly, so it's possible!). And finally, the class pulls in the logic from the `ControllerBase` class in via the use statement and `extends` it, which gives the `Hello` controller access to all `ControllerBase`'s convenient methods and capabilities, such as `$this->t()` (the OO way of calling the `t()` function). And, because `ControllerBase` is a standard PHP class, all its methods and properties will autocomplete in IDEs, so you aren't guessing at what it can and can't do for you.

Drupal 7: hook_block_X()

block.module

```
<?php
/**
 * Implements hook_block_info().
 */
function example_block_info() {
  $blocks['example'] = array(
    'info' => t('Example block'),
  );
  return $blocks;
}

/**
 * Implements hook_block_view().
 */
function example_block_view($delta = '') {
  $block = array();
  switch ($delta) {
    case 'example':
      $block['subject'] = t('Example block');
      $block['content'] = array(
        'hello' => array(
          '#markup' => t('Hello world'),
        ),
      );
      break;
  }
  return $block;
}
?>
```

Here's an example of a typical way in which you define "pluggable thingies" in Drupal (blocks, image effects, text formats, and so on): some kind of `_info()` hook, along with one or more other hooks to perform additional actions (view, apply, configure, and more). In addition to these largely being ArrayPIs, this time they're actually even worse "mystery meat" APIs, because the overall API itself is completely undiscoverable except by very careful inspection of various modules' **.api.php files** (provided they exist, which is not a guarantee) to discover which magically named hooks you need to define to implement this or that behavior. Some are required, some aren't. Can you guess which is which?

Drupal 8: Blocks (and many other things) as Plugins

In Drupal 8, these "mystery meat" APIs have now largely moved to the new **Plugin system**, which looks something like this:

src/Plugin/Block/Example.php

```
<?php
namespace Drupal\example\Plugin\Block;

use Drupal\Core\Block\BlockBase;

/**
 * Provides the Example block.
 *
 * @Block(
 *   id = "example",
 *   admin_label = @Translation("Example block")
 * )
 */
class Example extends BlockBase {
  public function build() {
    return array('hello' => array(
      '#markup' => $this->t('Hello world.')
    ));
  }
}
?>
```

Most of this is very similar to the Controller example; a plugin is a class that in this case extends from a base class (**`BlockBase`**) that takes care of some underlying things for you. The Block API itself is housed in the **`BlockPluginInterface`**, which the `BlockBase` class implements.

Note that interfaces in general expose and document various APIs in a discoverable and IDE-friendly way. A great way to learn about the new APIs in Drupal 8 is by browsing through the interfaces that are provided.

The comments above the class are called **annotations**. At first it might seem strange for PHP comments to be used for specifying metadata that affects the logic of the software, but this technique is now widely used by many modern PHP libraries and accepted by the PHP community. Its benefit is that it keeps the class metadata in the same file as and right next to the class definition.

Drupal 7: Hooks

In Drupal 7 and earlier, the extension mechanism used is the concept of **hooks**. As an API author, you can declare a hook using functions like `module_invoke_all()`, `module_implements()`, `drupal_alter()`, and so on. For example:

```
<?php
  // Compile a list of permissions from all modules for
  // display on admin form.
  foreach (module_implements('permission') as $module) {
    $modules[$module] = $module_info[$module]['name'];
  }
?>
```

If you wanted a module to respond to this event, you would create a function named `modulename_hookname()`, and declare its output in a way that the hook implementation expected. For example:

```
<?php
/**
 * Implements hook_permission().
 */
function menu_permission() {
  return array(
    'administer menu' => array(
      'title' => t('Administer menus and menu items'),
    ),
  );
}
?>
```

Although this is a clever extension hack that is mostly the result of Drupal's age (Drupal started in 2001 when PHP3 was all the rage and when there was little support for OO code and the like), there are several tricky bits:

- This "name a function in a special way" extension mechanism is very much a Drupalism, and developers coming to Drupal struggle to understand it at first.
- At least four different functions can trigger a hook: `module_invoke()`, `module_invoke_all()`, `module_implements()`, `drupal_alter()`, and more. This makes finding all the available extension points in Drupal very difficult.
- No consistency exists between what hooks expect. Some are info style hooks that want an array (sometimes an array of arrays of arrays of arrays), others are info-style hooks that respond when a particular thing happens like cron run or a node is saved. You need to read the documentation of each hook to understand what input and output it expects.

Drupal 8: Events

Although hooks are definitely still prevalent in Drupal 8 for most event-driven behavior (the info style hooks have largely moved to YAML or Plugin annotations), the portions of Drupal 8 that are more closely aligned to Symfony (for example, bootstrap/exit, routing system) have largely moved to **Symfony's Event Dispatcher system**. In this system, events are dispatched at runtime when certain logic occurs, and modules can subscribe classes to the events to which they want to react.

To demonstrate this, let's take a look at Drupal 8's configuration API, helpfully housed in **core/lib/Drupal/Core/Config/Config.php**. It defines a variety of CRUD methods such as `save()`, `delete()`, and so on. Each method dispatches an event when finished with its work, so other modules can react. For example, here's `Config::save()`:

```
<?php
  public function save() {
    // <snip>Validate the incoming information.</snip>

    // Save data to Drupal, then tell other modules this was
    // just done so they can react.
    $this->storage->write($this->name, $this->data);
    // ConfigCrudEvent is a class that extends from Symfony's
    // "Event" base class.
    $this->eventDispatcher->dispatch(ConfigEvents::SAVE, new
    ConfigCrudEvent($this));
  }
?>
```

As it turns out, at least one module needs to react when the configuration is saved: the core Language module. Because if the configuration setting that was just changed was the default site language, it needs to clear out compiled PHP files so the change takes effect.

To do this, Language module does three things:

1. Registers an event subscriber class in its language.services.yml file (this is a configuration file for the **Symfony Service Container** for registering reusable code):

```
language.config_subscriber:
  class: Drupal\language\EventSubscriber\ConfigSubscriber
  tags:
    - { name: event_subscriber }
```

2. In the **referenced class**, implements the **EventSubscriberInterface** and declares a `getSubscribedEvents()` method, which itemizes the events that it should be alerted to and provides each with one or more callbacks that should be triggered when the event happens, along with a "weight" to ensure certain modules that need to can get the first/last crack at the object can (heads-up: Symfony's weight scoring is the opposite of Drupal's):

```
<?php
class ConfigSubscriber implements EventSubscriberInterface {
  static function getSubscribedEvents() {
    $events[ConfigEvents::SAVE][] = array('onConfigSave', 0);
    return $events;
  }
}
?>
```

3. Defines the callback method, which contains the logic that should happen when the configuration is saved:

```
<?php
  public function onConfigSave(ConfigCrudEvent $event) {
    $saved_config = $event->getConfig();
    if ($saved_config->getName() == 'system.site' &&
     $event- >isChanged('langcode')) {
      // Trigger a container rebuild on the next request.
     PhpStorageFactory::get('service_container')
     ->deleteAl();
    }
  }
?>
```

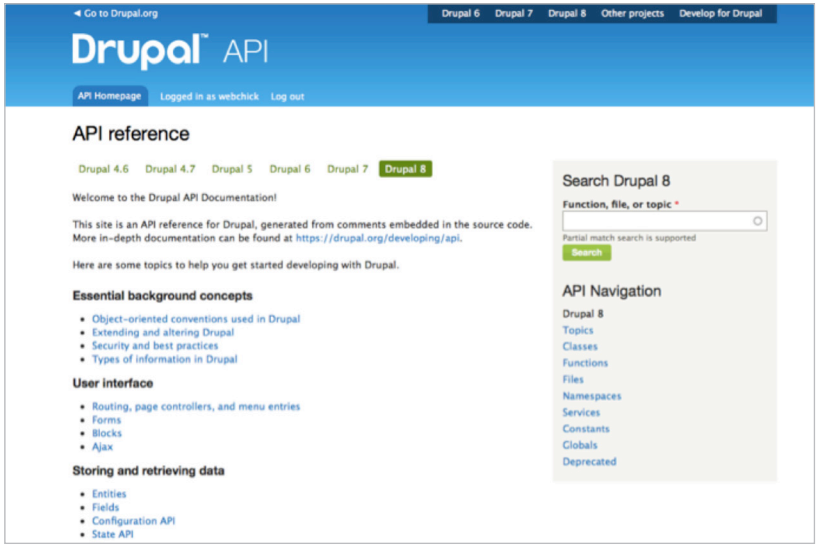
Overall this buys us a more explicit registration utility so that a single module can subscribe multiple classes to individual events. This allows us to avoid situations in the past where we had `switch` statements in hooks or multiple unrelated sets of logic competing for the developer's attention in a single code block. Instead, this provides the ability to separate logic into separate and distinct classes. This also means that our event logic is lazy loaded when it needs to be executed, not just sitting in PHP's memory at all times.

Debugging events and finding their implementations is also pretty straightforward. Instead of a handful of procedural PHP functions that may or may not have been used to call your hook, the same Event Dispatcher is used throughout the system. In addition to this, finding implementations is as simple as grepping for the appropriate Class Constant, for example, `ConfigEvents::SAVE`.

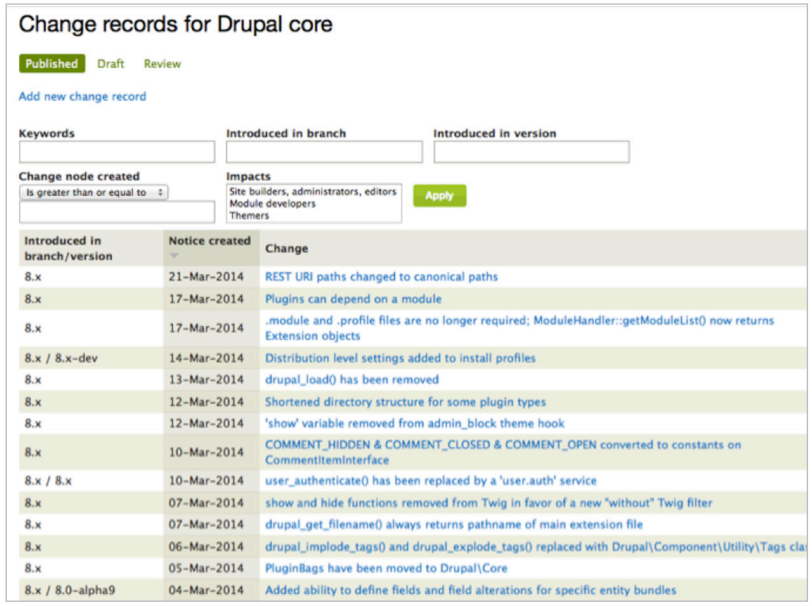
Logically, the event system rounds out the transition to an OO approach. Plugins handle info-style hooks and hooks that were called subsequent to an info hook. YAML stands in place for many of our explicit registration systems of old, and the event system covers event-style hooks and introduces a powerful subscription methodology for extending portions of Drupal core.

...and much, much more!

You can find a great introduction to Drupal 8's API changes at the revamped **D8 landing page of api.drupal.org** where you'll find a list of grouped topics to orient you to Drupal 8.



You can also see **https://drupal.org/list-changes** for the full list of API changes between Drupal 7 and Drupal 8. Maybe grab a drink first. Each API change record includes before/after code examples to help you migrate, as well as pointers to which issue(s) introduced the change and why.



So Much Typing

It's true that moving to modern, OO code generally involves more verbosity than procedural code. To help you over the hurdles, check out the following projects:

- **Drupal Module Upgrader:** If you're looking to port your modules from Drupal 7 to Drupal 8, look no further than this project. It can either tell you what you need to change (with pointers to the relevant change notices) or automatically convert your code in-place to Drupal 8. You can learn more about DMU in this **podcast interview with the maintainer**.
- **Console:** For new modules, this project is a Drupal 8 scaffolding code generator that will automatically generate .module/.info files, PSR-4 directory structures, YAML, and class registrations for routes, and more!
- Most Drupal 8 core developers swear by the **PhpStorm IDE**, and the latest version has lots of **great features for Drupal developers**. If you're one of the top contributors to the Drupal ecosystem, **you can get it for *free***! (Note that this isn't product placement. You should join #drupal-contribute at any time of the day or night and see if you can go longer than an hour without someone mentioning PhpStorm.)

Your Burning Questions

When will Drupal 8 be released? What do I need to do to upgrade? I have so many questions.

Why Should I Care about Drupal 8?

Drupal started first and foremost as a tool for developers and provided a set of APIs to allow building website elements in code, such as content entry forms, admin pages, and sidebar blocks. In later releases of Drupal, and particularly in Drupal 7, the emphasis was on making Drupal approachable for less technical users, providing user interfaces for foundational tasks (installation, data modeling, information architecture, landing pages, and so on). Most Drupal sites today download and configure a number of contributed projects for features such as a WYSIWYG editor, Views, and so on. And with that combination of core + contrib, Drupal runs **some of the biggest and most important sites on the web**.

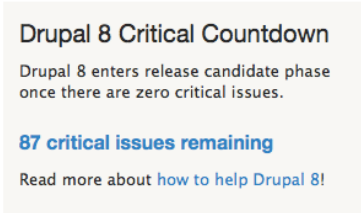
Drupal 8 builds on the success of Drupal 7 by incorporating more expected functionality out of the box, such as authoring experience improvements, complete multilingual functionality, and numerous site builder features. Drupal 8 is more in line with the web landscape of today, with its mobile-first approach and revamped front end. And, true to its developer roots, it offers numerous back-end features and modernized, OO code base. All around, Drupal 8 is a more powerful release with capabilities for content authors, site builders, developers, and designers alike. It is built in a future-proof way so that it can act as a solid foundation for projects no matter what technologies, devices, or services come out next.

That said, Drupal 7 is a stable, robust, and mature platform that will be supported for several more years to come. And much of the functionality within Drupal 8 is available in some form in Drupal 7 (a later answer digs into more details). Drupal 8 will be great, but so is Drupal 7 if you just can't wait. And either way, it's still a great idea to start learning about Drupal 8 now, so you can be prepared when it suits your future project needs.

Wow, Drupal 8 Sounds Great! What's Standing in the Way of It Being Released?

Once the number of **critical issues (bugs and tasks)** hits zero, a Drupal 8 "release candidate" will be created. Once a release candidate has been out in the wild with no new critical issues reported, Drupal 8.0.0 will be tagged and released, to much enthusiastic applause!

You can see how close or far away Drupal 8 is from shipping at any given time by over to **www.drupal.org/drupal-8.0/get-involved**. The sidebar block shows the number of critical issues left to go.



What Happens after Drupal 8 Is Released?

Parties. Lots and lots of parties.

But then, starting with Drupal 8.0.0, the Drupal project is **moving to a new release cycle**, which in addition to standard monthly bug fix and security releases (8.0.1, 8.0.2...) introduces semi-annual minor releases of core (8.1.0, 8.2.0, and so on. These releases can include new features, backward-compatible API improvements, and more. After several minor versions, a "Long-Term Support" (LTS) release of Drupal 8 will be created and Drupal 9 development will begin.

This means Drupalistas will no longer have to wait *N* years for new core functionality; we can iterate on features and APIs every few months until the platform reaches maturity. It also means that those who are more risk-averse and want stability over shiny things can stick to LTS releases and only move every several years (even leap-frogging major versions). Hooray!

When Should I Actually Start *Using* Drupal 8?

The answer to that depends on who you are:

- If you're a module developer, you should start caring about Drupal 8 right now. It's still possible to provide useful feedback on APIs and ensure Drupal 8 ships with everything you need to get your projects ported. But bear in mind that some of Drupal 8's APIs will still be changed before release if needed to fix critical issues, so you may still need to make code adjustments post-RC.
- If you're a documentation author, translator, or designer, note that Drupal 8's user interface, interface text, and markup are not finalized until the first release candidate, so you'd want to wait until RC1 to focus heavily on user-facing documentation, translations, or themes (though by all means, adventurous contributors should start now to provide feedback while we can still fix things).
- If you're an early adopter Drupal user with developers on staff who don't mind porting modules and fixing core bugs along the way, and have a launch date in late 2015 or 2016, you may want to start building your D8 sites once Drupal 8 hits a late beta or a RC. This would be a particularly good idea if you need some of the features Drupal 8 offers.
- Most users will want to use Drupal 8 a few months after Drupal 8's release, when various contributed modules are ported. Keep your eyes on the **Drupal project usage graph**. When the D7 and D8 lines cross, it may be a good time for you to make the jump, because it means there are more D8 users than D7, so most of the hard work has been done for you already.

Well, Dang. So What Should I Do in the Meantime?

Use Drupal 7. Drupal 7 is a stable, mature, robust, powerful, well-supported framework which will be maintained with bug fixes until after the LTS release of Drupal 8, and supported with security fixes until Drupal 9's LTS release (several years from now). And a number of the great features in Drupal 8 are available in Drupal 7 as well, with contributed modules.

Drupal 8 Core Feature	Drupal 7 Contrib Equivalent
WYSIWYG	CKEditor: https://drupal.org/project/ckeditor
In-Place Editing	Quick Edit: https://drupal.org/project/quickedit
Responsive Toolbar	Mobile Friendly Navigation Toolbar: https://drupal.org/project/navbar
Responsive Front-End Theme	Omega, Zen, Adaptive, Aurora, etc. base themes
Responsive Admin Theme	Ember: https://drupal.org/project/ember
Responsive Images	Picture: https://drupal.org/project/picture
Responsive Tables	Responsive Tables: https://drupal.org/project/responsive_tables
Simplified Overlay	Escape Admin: https://drupal.org/project/escape_admin
Multilingual	Internationalization: https://drupal.org/project/i18n Entity Translation: https://drupal.org/project/entity_translation (and several additional modules)
Better Blocks	Bean: https://drupal.org/project/bean
Configuration Management	Features: https://drupal.org/project/features (provides exportable files that can be used in deployments)
Web Services	RESTful Web Services: https://drupal.org/project/restws

What about the Upgrade Path?

Oh you had to ask, didn't you?

- For your site's content (users, articles, and so on) and many configuration settings (variables, block settings, and so on), Drupal 8 will provide a migration path from both Drupal 6 (already in core) and Drupal 7 (currently under construction) to Drupal 8 that will cover core modules. (Contributed and custom modules will need to write their own **migration path** to cover their data.) Basically, you'll keep your Drupal 6/7 site running while you build your Drupal 8 site and then run a script similar to the current update.php to move its contents over. When things look good, swap out the web roots. Almost no downtime!
- For your site's contributed modules, download and install the 7.x version of the **Upgrade Status** module, which shows a handy overview of your module's site and the current D8 porting status.
- For your site's custom modules, you need to port those yourself. The **Drupal Module Upgrader** project can help automate some of this and generate a report of other things to change. (However, it is not omniscient, so you will still need to fix some things by hand.)
- For your site's custom theme, which must be converted to Twig, check the **Twigifier** project that attempts to automate much of this work.

So, in short, your upgrade path depends a lot on the specifics of your site and how it's put together. In general, you'll have a much easier time moving to Drupal 8 if you stick to well-vetted contributed modules versus custom code. Plan out your current site builds accordingly.

For other tips on making your Drupal 6/7 site Drupal 8 ready, check out **www.acquia.com/blog/getting-your-site-ready-drupal-8**.

How Can I Help?

Want Drupal 8 to come out faster? It can, with your help.

- The most direct way is to **help fix critical issues**. Keep your eyes posted on **Drupal Core Updates**, which always has the latest spots that need particular attention.
- If you're new to Drupal core development, or want a pointer to some useful things to work on by a real person, check out **core mentoring hours** twice-weekly on IRC.
- Want to help with the Drupal 8 migration path? Check out the **IMP (Migrate in core) team**.
- Want to help with the Drupal 8 documentation? Check out the **Current documentation priorities**.
- Want to learn Drupal 8 APIs and help other developers in the process? Help port **Examples for Developers** to Drupal 8.
- Want to save yourself and others lots of time porting modules? Help write **Drupal Module Upgrader** routines.

Thank you!

Let's give a virtual round of applause to **more than 2,500 contributors to Drupal 8 so far! Now, join them!**

ACQUIA® THE DIGITAL EXPERIENCE COMPANY
acquia.com | 888.922.7842 | 1.781.238.8600 | 25 Corporate Drive, Burlington, MA 01803
SKU 00476-150109-EBOOK