



Ultimate Guide to **Drupal 8**

Revised and updated for Drupal 8.2, as of October, 2016

Angela Byron, Director, Community Development, Acquia

Jeffrey A. “jam” McGuire, Evangelist, Developer Relations, Acquia



Table of Contents

Introduction	<u>2</u>
Authoring Experience	<u>4</u>
Mobile Improvements	<u>7</u>
Multilingual++	<u>9</u>
Front-end Developer Improvements	<u>17</u>
Back-end Developer Improvements	<u>24</u>
Better, Right Down to the Core	<u>30</u>
Your Burning Questions	<u>40</u>

Introduction

Drupal 8 has a lot in store for you, whatever you do with Drupal. This ebook will enumerate the major changes, features, and updates in Drupal 8 – and specifically Drupal 8.2 – for service providers and end users, site builders, designers, theme- and front-end developers, and for module and back-end developers.

The Drupal community has moved to support innovation within major releases with a new “semantic versioning” release system. Starting with Drupal 8, a new “minor” version will be released every six months—Drupal 8.0.x, 8.1.x, 8.2.x, and so on, which can include backwards-compatible new features and functional improvements (which was not possible in Drupal 7 and below). This is alongside the expected bug fix, “patch” releases between minor versions; for example, Drupal 8.1.1, 8.1.2, and so on.

To further accelerate the pace of innovation, Drupal 8 includes [“experimental” core modules](#) alongside the semantic versioning concept. While these modules don’t necessarily conform to the rigorous requirements for “full” Drupal core modules, they allow core contributors to iterate quickly and get real-world feedback on important new functionality that may be fully supported in future versions of Drupal 8. Experimental core modules must become full modules within a year of their introduction (two minor releases) or be removed from core again. Drupal 8.2 includes 9 experimental core modules: “beta-stable” BigPipe, and “alpha-stable” Migrate, Migrate Drupal, Drupal Upgrade UI, Inline Form Errors, Place Block, Content Moderation, Settings Tray, and DateTime range.

While the focus of this ebook is firmly on Drupal 8 and the improvements in Drupal 8.2, for those familiar with Drupal 7.x, it includes some comparisons to Drupal 7 functionality and references to Drupal 7 equivalents of Drupal 8 features—for example, Drupal 7 contributed modules.



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.



Jeffrey A. "jam" McGuire — Evangelist, Developer Relations at Acquia — is a memorable and charismatic communicator with a strong following at the intersection of open source software, business, and culture. He is a frequent keynote speaker at events around the world. He writes and talks about technology, community, and more on weekly podcasts and as a blogger on [dev.acquia.com](#).

Authoring Experience

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



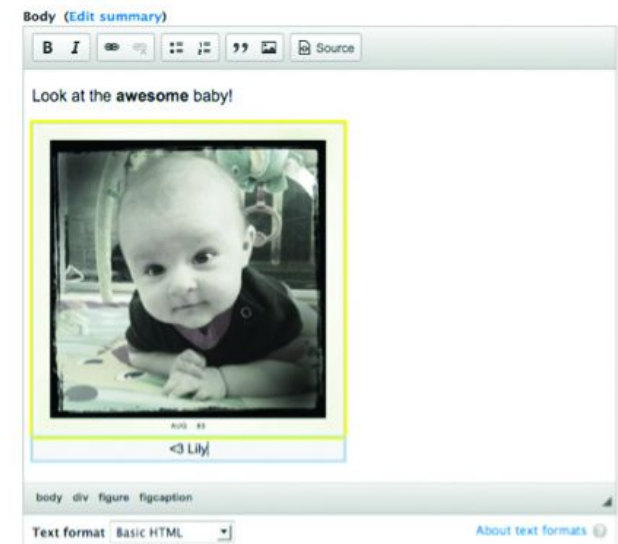
Spark

Spark is an Acquia initiative created by Dries Buytaert [to improve Drupal core's default authoring experience](#). The Acquia development team for Drupal core analysed both proprietary and open source competitors to Drupal and worked hard to make usability enhancements to Drupal core over the course of the release in collaboration with other Drupal core contributors. They also created [back ports of key Drupal 8 UX improvements for Drupal 7](#), which allowed them to be tested and improved under everyday, real use even before the release of Drupal 8.

WYSIWYG Editor

Drupal 8 ships with the [CKEditor](#) WYSIWYG editor in the default installation. 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. It is fully integrated into Drupal 8, from Drupal-styled dialogs, to utilizing Drupal's user roles and permissions, to image management, and it [ensures that we keep the benefits of Drupal's structured content concepts in our WYSIWYG implementation](#).

Drupal 8 also sports a drag-and-drop admin interface for customising the WYSIWYG toolbar; adding and removing buttons automatically syncs the allowed HTML tags for a given text format. 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 accessible editing experience for visually impaired users. Though core will only support CKEditor, Drupal 8's Editor module wraps around the WYSIWYG integration, so other text editors, libraries and contrib modules can be used and tightly integrated as well.



In-place Editing

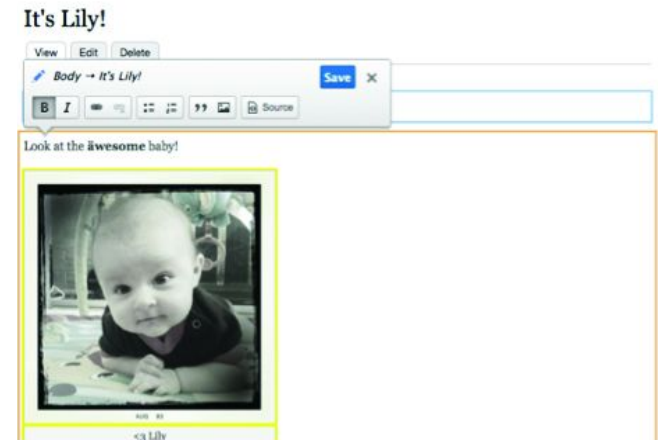
In Drupal 7, if you need to make a correction on a website—for example, a typo, or a missing image—you must use a back-end form, which is visually separated from the front-end website where content will appear. The Preview button doesn't help, because the results of preview are shown in the administrative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

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

To replace Drupal 7's default editing behavior, which required a more time-consuming visit to the administrative back-end of the site, this in-place editing feature has been backported to Drupal 7 as the [Quick Edit module](#).

Outside-in Usability!

Drupal 8.2 introduces a new-to-Drupal usability concept in two experimental core modules: “outside-in”, where configuration changes are made right from the front-end of the website. The new Place Block module allows you to place blocks on any page without having to navigate to the backend administration form. Block configuration has also become much easier with the experimental Settings Tray module. Selecting a block to edit opens a tray in a sidebar where you can edit the block's settings directly. You can configure menus that appear in menu blocks as well as change the site name and site slogan in-place. The settings tray system is expected to be available for integration with other modules soon.



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.

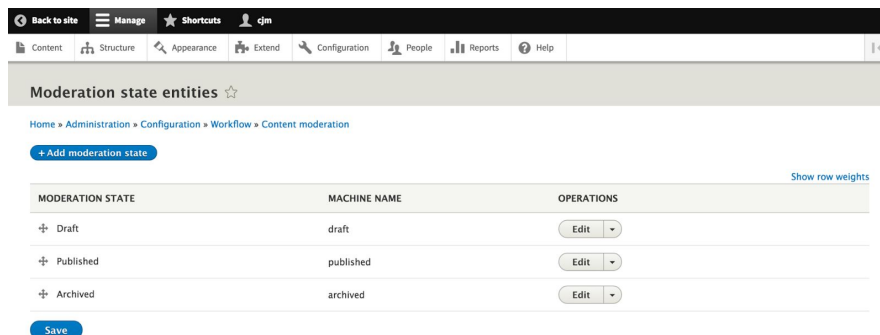
The new design lets content authors focus on the task at hand while having important publishing options just a click away.

Refreshed Admin Theme

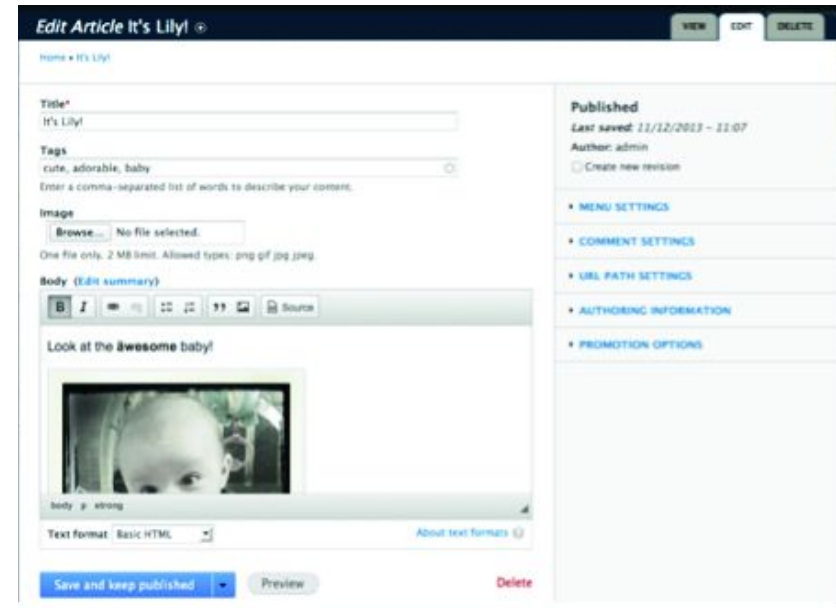
The administrative theme in Drupal 8 is a visually refreshed version of Drupal 7's, based on a formal [style guide](#) which can also be used by module developers and others concerned about backend usability.

Content Moderation

As part of the plan for incremental improvement throughout the Drupal 8 release cycle, Drupal 8.0.0 core included API support under the hood for a draft revision-state to help integrate publishing-workflow modules, like [Workbench](#). Drupal has always supported both published and unpublished content, but more granular workflow support was not available in Drupal core. The new experimental Content Moderation module, based on the contributed Workbench Moderation project and added to core in 8.2, allows defining content workflow states such as Draft, Archived, and Published, as well as which roles have the ability to move content between states.



MODERATION STATE	MACHINE NAME	OPERATIONS
✚ Draft	draft	Edit
✚ Published	published	Edit
✚ Archived	archived	Edit



Automatic revision saving

Drupal now enables revisions by default for new content types. This is important for systems-of-record and compliance, providing better accountability, creating a "safety net" for recovering from unintended changes, and for integrating with further future workflow features.

Mobile Improvements

A huge amount of work has gone into making Drupal 8 “mobile friendly.” Drupal 8 is able to support site visitors’ needs as they surf the web on their tablets and phones, as well as enabling authors and editors to actually work productively on their sites from mobile devices.



Mobile First

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 the smallest of screens. Give Drupal 8 a try on your device of choice, and [let us know what you think](#).

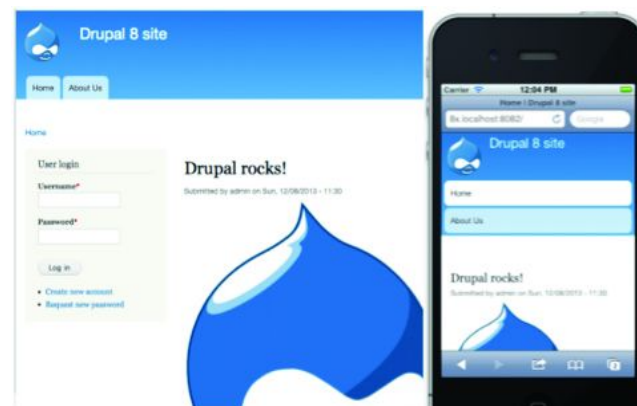
The new search box on the modules page adds to your Drupal-8-on-mobile experience by saving you a lot of scrolling when you need to get to the settings for a particular module. Check out [Module Filter](#) for a similar experience in Drupal 7.

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 with table columns that can be declared with 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](#), can help you build a responsive design for your website.

Mobile-friendly Toolbar

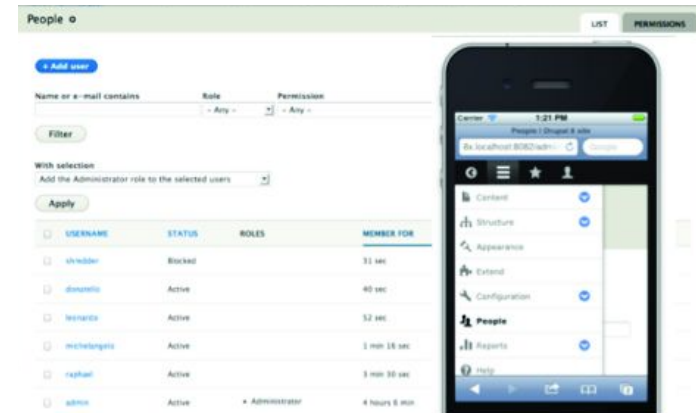
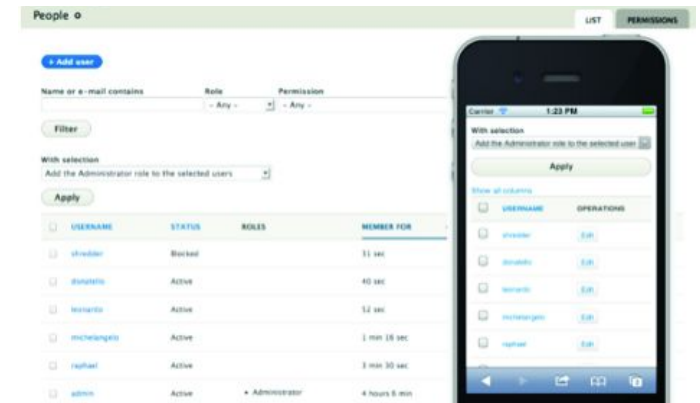
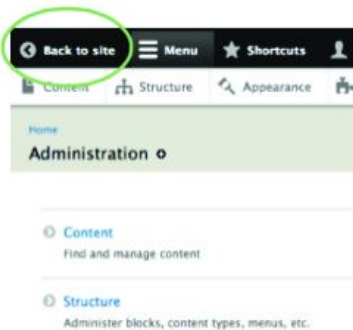
Drupal 8 sports a responsive administrative toolbar that automatically expands and orients itself horizontally on wide screens and collapses down to icons and orients itself vertically on smaller screens. Like all new front-end features in Drupal 8, this one was designed with accessibility in mind. The toolbar allows screen-reader users to easily navigate to various parts of a 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. Page loads were sped up by replacing jQuery with efficient, targeted, native JavaScript in many cases and out-of-the-box Drupal 8 loads no JavaScript files at all for anonymous visitors. Drupal 8's caching is a big advance over Drupal 7's. It includes entity caching and powerful, granular cache

tags which allow for targeted cache clearing so pages stay fast longer. Drupal 8.1 also introduced BigPipe page delivery as an “alpha-stability” [experimental core module](#). It has been upgraded to “beta-stability” in Drupal 8.2, meaning it is ready for use and testing by early adopters, and the API is stable enough for developers to begin using and extending it. See the Backend Developer Improvements chapter of this ebook for more on caching and BigPipe's effect on user experience thanks to faster perceived loading. Additionally, lighter-weight, mobile-friendly alternatives replaced JavaScript-intensive features like the Overlay module. Drupal 8 uses a simple “Back to site” link in the admin toolbar 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 the participation of over [1,000 contributors](#), was 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

Drupal 8 is a CMS built from the ground up for multilingual use. You can perform your entire site installation and setup in your language of choice. Right on the installer page, it auto-detects your browser's language and auto-selects that language for installation in the drop-down for your convenience. When you install Drupal in any language other than English (or later add a new language to your site), Drupal 8 automatically downloads the latest interface translations from localize.drupal.org in your language, too. This works for right-to-left languages, such as Arabic and Hebrew, too. Drupal's interface translations are dependent on local communities for accuracy and completeness, so some translations may be missing some strings.

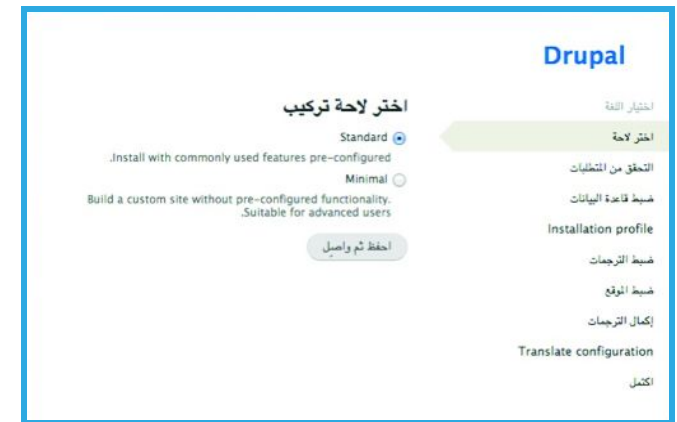
On localize.drupal.org, you can always contribute those yourself and help your language community take advantage of Drupal. Drupal 8 does away with the previous Drupal-concept of English as a “special” language. If you select a language other than English on installation, the English option will no longer show in your site configuration unless explicitly turned on. Also, you can make English itself “translatable” so that you can convert strings to something more tailored to your users. For example, you can change “Log in / Log off” to “Sign in / Sign off.”



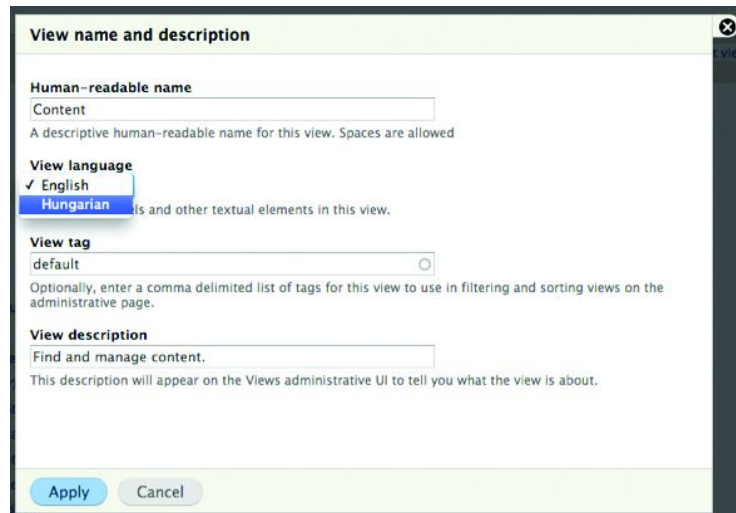
Fewer Modules, Packing a Bigger Punch

Making a site multilingual in Drupal 8 requires nothing more than activating one or more of just four modules, all shipped with Drupal 8 core. These four modules do everything and more than the roughly 30 contributed modules and lots of tricky configuration needed to make a Drupal 7 site multilingual.

- **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.
- **Content Translation** makes things such as nodes, taxonomy terms, and comments translatable.
- **Interface Translation** makes Drupal's user interface itself translatable.



Why four modules and not just one, you ask? This granularity allows site builders to choose whatever combination of features meet their site's specific use case, without forcing them to deal with the parts they don't need. For example, single-language, non-English sites are a valid use case, as are multilingual sites that may or may not need their content translated (e.g. to keep user-generated content in its native language), as are a plethora of combinations of interface languages and content translations for site admins, content authors, and end users.



Language Selection Everywhere

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, like configuring 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.

Streamlined Translation UIs

Drupal's international community put a lot of effort into the user experience of Drupal 8's multilingual functionality. You'll see well-integrated and streamlined translation interfaces throughout Drupal 8.

Transliteration Support

One really handy addition to Drupal 8 is the inclusion of the Transliteration module in core. It automatically converts special characters like “ç” and “ü” to “c” and “u” for friendlier, more human-readable 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.
- The Drupal 8 core Content Translation module is well-integrated with Drupal 8's core search and the Search API can also access language information for integration with search technologies like Apache Solr and Elasticsearch.
- The language selection system now supports one or more separate admin languages, for easier management of multilingual sites by site admins who might speak different languages.

The image displays a grid of eight screenshots from the Drupal 8 translation interface, arranged in two columns and four rows. Each screenshot shows a specific translation task with its singular and plural forms. The first row shows the translation of a translation file for enabled modules. The second row shows a content type change from 'Solid-type' to 'Ntype'. The third row shows the translation of a file path. The fourth row shows a debug message. The fifth row shows a file path translation. The sixth row shows a file path translation. The seventh row shows a file path translation. The eighth row shows a file path translation. The screenshots are organized into two columns, with the left column showing the original text and the right column showing the translated text.

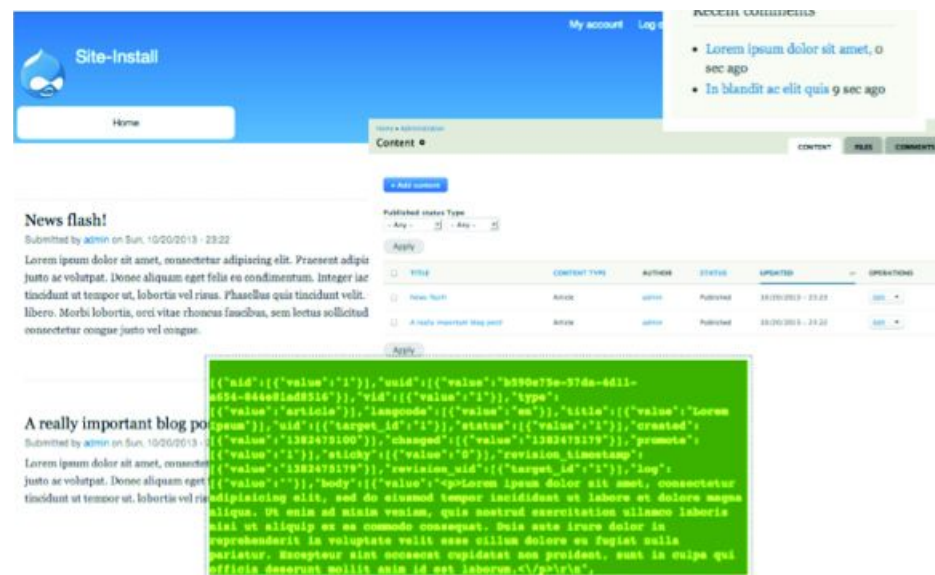
Original Text	Translated Text
Singular form One translation file imported for the enabled modules.	Singular form A bekapcsolt modulokhoz egy fordítási fájl lett betöltve.
Plural form @count translation files imported for the enabled modules.	Plural form A bekapcsolt modulokhoz @count fordítási fájl lett betöltve.
Singular form Changed the content type of 1 post from Solid-type to Ntype.*	Singular form Egy tartalomtípus Solid-típe helyett Ntype lett.
Plural form Changed the content type of @count posts from Solid-type to Ntype.*	Plural form @count tartalomtípus Solid-típe helyett Ntype lett.
All (@count)	Mind (@count)
Pass (@count)	Átment (@count)
Fail (@count)*	Meghibásított (@count)
Singular form 1 debug message	Singular form 1 hibakereső üzenet
Plural form @count debug messages	Plural form @count hibakereső üzenet

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 contributed module in Drupal](#), is now part of Drupal 8 core and is more tightly integrated into Drupal than ever before. Beyond providing a query-builder UI and serving up the results in a variety of formats for site visitors, baking Views into Drupal core allowed core developers to replace numerous previously hardcoded admin pages with Views listings. This removed thousands of lines of boilerplate code from Drupal core and more importantly, gives site builders the power to customise most of the main administrative listings (or even build brand new ones!). The Content, People, and Files admin pages, as well as various sidebar blocks, several RSS feeds, and the default front page have been converted to Views. Almost everyone who has built a site of any complexity in Drupal knows how to use Views. This makes customizations of these pages—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 in Drupal 8 core—and even a few extras such as mobile-friendly administration, some 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 replacing admin pages, several previously hard-coded site components have been converted to blocks, including breadcrumbs, site name, and slogan. This makes it easier to adjust page organization in the user interface, and enables in-place editing, and makes for easier theming.

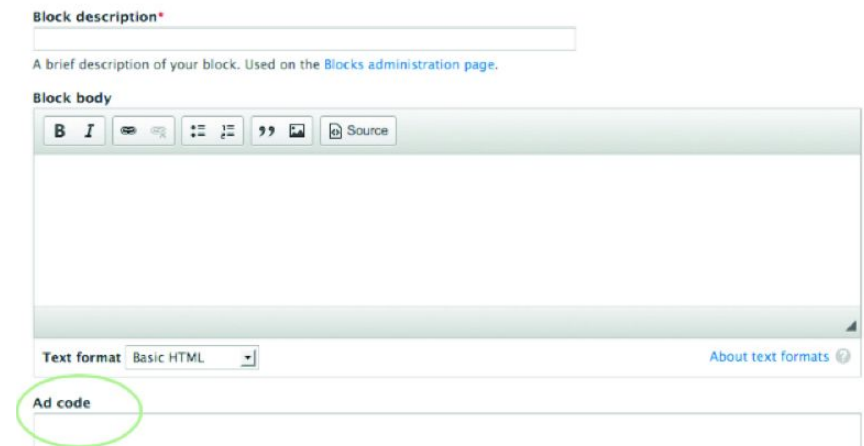
Reusable Blocks and Custom Block Types

A nice addition to Drupal 8 is the ability to re-use blocks. You can place a block in multiple places, for example, a “Navigation” block in both the header and footer.

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 and then add and place as many different blocks of that type on your site as you need.

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. Everything from content, to users, comments, and much more are all entities. You can add fields to all entities, including references to other entities. This makes it easier than ever to build data models for the structured content you want to manage using Drupal.



Block description*

A brief description of your block. Used on the [Blocks administration page](#).

Block body

Text format: Basic HTML

Ad code

More Field Types

To build those data models, Drupal 8.2 includes a plethora of fundamental, semantic field types like Taxonomy, Image, Phone, Email, Link, and File, as

well as some more powerful fields such as Entity Reference and Date Range. 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 the 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.

The screenshot displays the configuration interface for a field in Drupal 8. On the left, the 'Allowed number of values' is set to 'Limited' with a value of '1'. The 'Type of item to reference' is set to 'Content'. A 'Save' button is visible. The 'REFERENCE TYPE' section is expanded, showing 'Reference method' set to 'Default'. Under 'Content types', 'Article' is selected with a checked checkbox, and 'Basic page' is unselected. The 'Sort by' dropdown is set to '- None -'. On the right, a date picker is shown for 'December-2013', displaying a calendar grid with days of the week and dates. Below the calendar, the date format 'yyyy-mm-dd' is visible.

Take a Tour

Drupal 8's new Tour module lets site builders create contextual, step-by-step tooltip-style walkthroughs of your site. It can help with overviews of administrative interfaces, introduce new terminology, and walk through the steps involved in configuring components of your site.

Both Less and More, Module-wise

You'll find Drupal 8 missing some modules that shipped with Drupal 7,

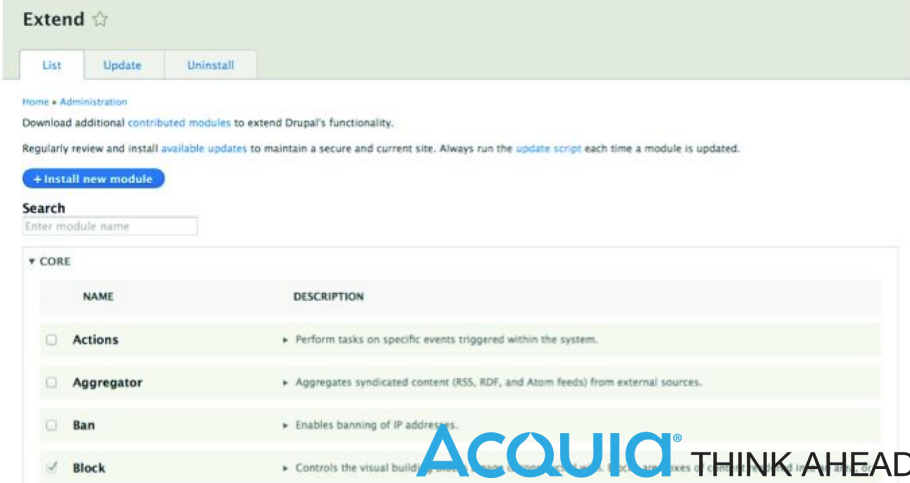
namely Blog, Dashboard, Open ID, Overlay, PHP filter, Poll, Profile, and Trigger (as well as the Garland theme). You'll find several new modules in which functionality has been split out into more granular chunks, such as Menu Links/Menu UI, Block/Custom Block, Ban/History/Actions (formally baked into 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 8 core ships with enough functionality out-of-the-box that site builders can create fairly sophisticated sites without having to install a plethora of contributed modules.

Migration Path

Drupal's 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. Since Drupal 8.1, there is also a Migration UI in core, which allows major Drupal version migrations without resorting to command-line tools. Both a migration path from Drupal 6 (already in Drupal 8.x) and Drupal 7 (partially in 8.x and under development) are supported. The migration path allows you to keep your Drupal 6/7 site running while you build your new Drupal 8 site, greatly minimizing downtime over the old



The screenshot shows the 'Extend' page in Drupal 8. At the top, there are tabs for 'List', 'Update', and 'Uninstall'. Below these, there's a section for 'Download additional contributed modules to extend Drupal's functionality.' and a note to 'Regularly review and install available updates to maintain a secure and current site.' A blue button labeled '+ Install new module' is visible. Below this is a 'Search' section with a text input field. The main content area is titled 'CORE' and contains a table with two columns: 'NAME' and 'DESCRIPTION'. The table lists several modules: 'Actions', 'Aggregator', 'Ban', and 'Block'. Each row has a checkbox in the 'NAME' column. The 'Block' module is checked. The 'DESCRIPTION' column provides a brief description for each module.

NAME	DESCRIPTION
<input type="checkbox"/> Actions	Perform tasks on specific events triggered within the system.
<input type="checkbox"/> Aggregator	Aggregates syndicated content (RSS, RDF, and Atom feeds) from external sources.
<input type="checkbox"/> Ban	Enables banning of IP addresses.
<input checked="" type="checkbox"/> Block	Controls the visual building blocks of the site.

update.php method.

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 improvements for front-end developers, including HTML5, additional helper 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 by default, as part of an overarching effort to clean up Drupal's default markup. This means you'll find tags such as <nav>, <header>, <main>, and <section> in Drupal's default templates and you'll find HTML5/CSS3 replacements for several things that previously needed custom workarounds: resizing on text areas and first/last/odd/ even classes is now covered by CSS3

pseudo-selectors; and collapsible fieldsets largely replaced by the by the <details> element.

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](#).

New Front-end Libraries and Helpers

Drupal has shipped with jQuery since Drupal 5 and jQuery UI since Drupal 7. Drupal 8 brings with it an update to the latest version of jQuery and an expanded array of front-end libraries. 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. These include:

- [Modernizr](#) (detects if a browser supports touch or HTML5/CSS3 features)
- [Underscore.js](#) (a lightweight JS-helper library)
- [Backbone.js](#) (a model-view-controller JavaScript framework).



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 to extract and index data from your site because the [schema.org](#) markup is semantic. That is, it adds meaning to your content. It shows, for example, that a given set of numbers are your restaurant's opening hours and another are your phone number; one set of words is your menu, and another the owner's name.



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 semantic meaning to elements in 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 allow module developers to create accessible applications easily.

The [video](#) to the right, extracted from [Dries's DrupalCon Prague Keynote](#), demonstrates how these new accessibility features appear to assistive technology users.



https://www.youtube.com/watch?v=8uhNFoOnz_g

New Theme System: Twig

Drupal 8 introduces [Twig](#), a very widely adopted theme system in the PHP world, to Drupal. Twig's syntax is simpler and Twig is more secure than the the PHPTemplate-based theme system in Drupal 7 and below that it replaces. It allows designers and themers with HTML/CSS knowledge to modify markup without needing to be a PHP expert and with almost no risk of their actions causing security issues on your site.

With Twig, themers no longer need to understand the syntax differences

```
<?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 %}
```

between deeply-nested arrays and objects, nor when to use each. In Twig, a simple `{{ foo.bar }}` statement does the trick. Simple conditional and looping logic can be contained in `{% ... %}` tags.

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.

These also allow you to create alternate templates and have them override the main one based on the specificity of their name (like CSS selectors) and use case. It's a bit like having the fabulous Theme Developer module baked into core! For example:

```
    {{ 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>
?>
```

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 CMS's need to have their high-performance settings enabled 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

```

<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>

```

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';
# }
?>

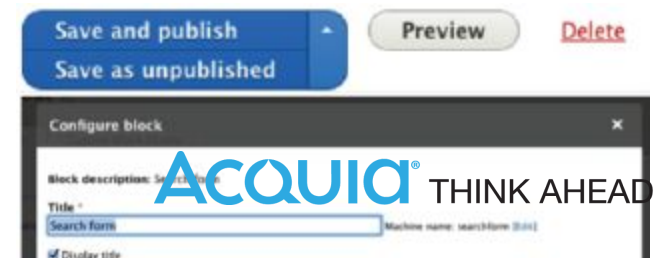
```

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 8.2 still ships with the latest, updated versions of jQuery and jQuery UI, a lot of movement is going away from using libraries like this in favor of 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 doesn't load any JavaScript at all for anonymous users!

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; styled blue in the default admin theme), and “danger” (styled as red links) to help users recognize and make correct choices when confronted with multiple options on a form.

Theme Responsively

As mentioned in the [Mobile Improvements](#) section of this ebook, 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.

Drupal 8 also ships with the new [Responsive Image module](#), which contains support for the HTML5’s [<picture> and <source>](#) elements, as well as the sizes, srcset and type attributes. 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.

New Method of Selectively Adding JS/CSS to the Page

Also helping out on the performance front, Drupal 8 has a new recommended best-practice for registering JS and CSS assets (along with their dependencies). Assets are defined in your MODULE/THEME.libraries.yml file as a series of properties that you then reference in the #attached property of an element or render array. 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 the Drupal-7-style quick in-line call to `drupal_add_FOO()`, it makes these assets cacheable for improved performance, and easily re-usable among different parts of the code base.

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

Another big improvement for front-end developers and designers is that 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.

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 on Drupal 8 websites. 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.



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.

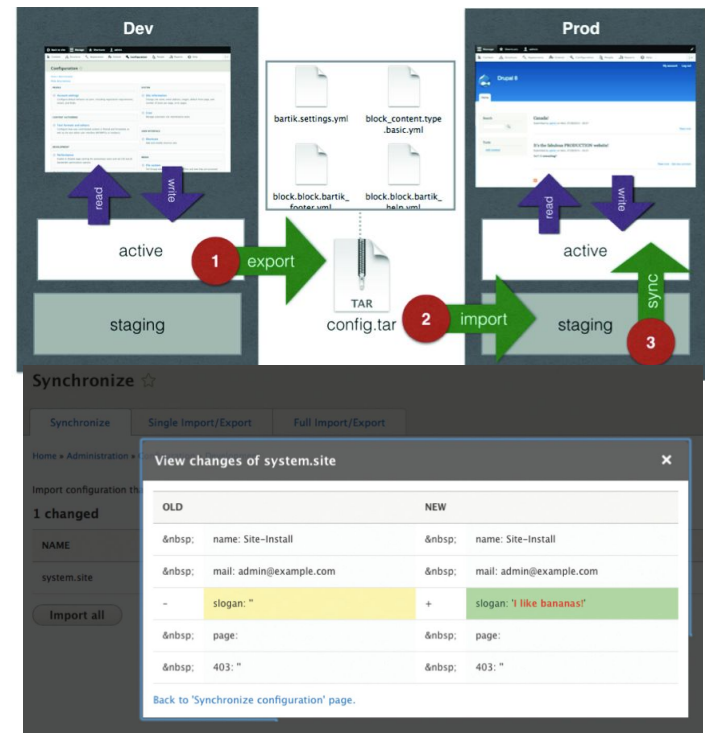
In Drupal 8, all configuration changes (both standard admin settings forms, such as site name, as well as any [ConfigEntity](#) including Views, user roles, and content types) run through a unified [configuration API](#). Each environment has a “sync” directory to hold configuration changes from other environments that are about to be imported for review. For performance, active configuration is stored in a config table in the database, though the storage location is swappable (e.g. to something like Redis or CassanDrupaldra).

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 configuration. You'll receive a tarball that consists of lots of YAML files.
2. On the production site, import the files, which places them into the config "sync" 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 the changes are acceptable, synchronize them. This will replace production's current active store with the contents of the sync directory 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 that you don't want to be deployed across environments; for example, the timestamp storing the last time cron ran. For these more ephemeral settings, there's a "sister" API to the configuration API named the [State API](#). There's also the Configuration Development module, which automatically exports/imports active configuration in files and is handy during development.



What About Content Deployment?

Drupal 8.2 core ships with improved (over Drupal 8.1) alpha-stability-support for migrating content such as nodes, users, and taxonomy terms between sites via the Migrate, Migrate Drupal, and the Migrate Drupal UI core experimental modules. Outside of full site migrations, one welcome addition to Drupal 8 that facilitates content deployment has been the introduction of UUIDs (universally unique identifiers) to every piece of content. These UUIDs can be used to determine whether a piece of content from a source site exists on a given destination site. This makes content imports/exports infinitely easier because even if both the source and destination sites have a node/100, for example, each will have a unique (obviously!) UUID. Contributed modules such as [Deploy module](#) can make use of this data for facilitating content transfers. If you're still on Drupal 7, you can get similar functionality to what Drupal 8 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.

In Drupal 8, the [Entity API](#) has been completely overhauled to greatly improve the developer experience and fill in the gaps in functionality compared to the Drupal 7 core API. 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 (to aid in translation and localization). Compare and contrast how this is done in D7 and D8:

Drupal 7

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

Drupal 8

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

Configuration and Content Entities

Nearly anything you can create more than one of in Drupal 8 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

- 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

Content entities sport some nifty new features compared to Drupal 7, like that revisions aren't just for nodes anymore (!) and the ability to add comments to any content entity (users, taxonomy terms... you can even have comments on comments!). The "[Site Builder Improvements](#)" chapter of this ebook has more information about other entity-related features.

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 is a native REST API built into Drupal 8 and supported by the [RESTful Web Services](#) suite of modules. Drupal 8.2 continues to expand Drupal's support for web services, enabling Drupal 8 to produce and consume web services for the creation of Drupal-powered decoupled and mobile applications, facilitate cross-site communication, and allow better integration with third-party resources, for example in micro-services-style applications.

The Drupal 8 REST API 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. The contributed [REST UI module](#) provides an interface for this configuration. You can define which role(s) on your site may access resources via each allowed HTTP method. For example, you can allow anonymous users to GET but only administrators to POST.

As of Drupal 8.2, it is possible to read (GET) configuration entities like vocabularies and content types as REST resources (previously not possible) and there are dedicated resources for user login, logout and registration. The authentication mechanism used by a REST Export Views Display is now configurable, and support was added for cross-origin resource sharing (CORS) to facilitate REST requests across different domain names. REST resource configuration is now also significantly simpler, and error messages much clearer.

Once the various RESTful Web Services modules are configured properly, you can expose machine-readable data representing your site content, like this:

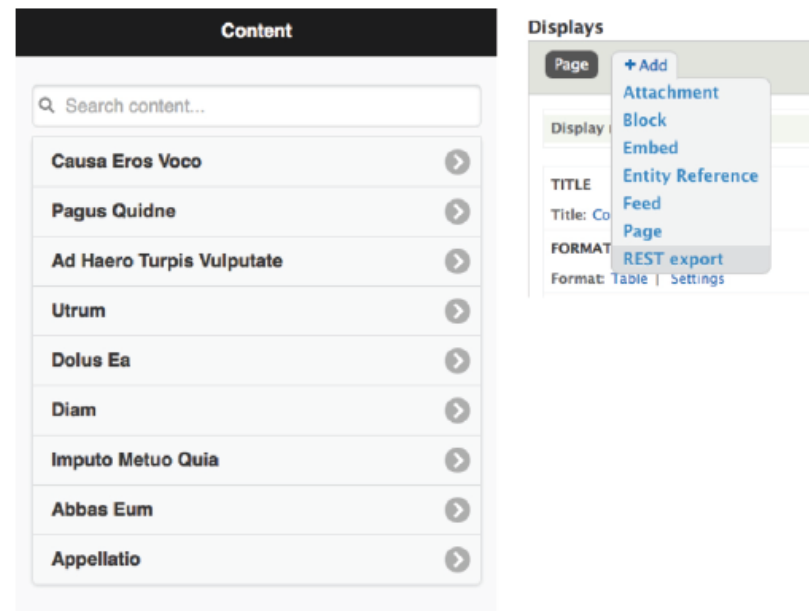
```
..
[title] => Array
(
[0] => Array
(
[value] => Hello, world!
[lang] => en
)
)
...
[body] => Array
(
[0] => Array
(
[value] => <p>This is my <strong>awesome
</strong> article.</p>
[format] => basic_html
[summary] =>
)
)
...
```

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 8.2 ships with an updated version of the [Guzzle PHP HTTP library](#), which gives us 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.

- **Fast by default:** All caching features such as CSS/JS aggregation are turned on out of the box.
- **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 and pave the way for further improvements in performance and user experience like BigPipe, [RefreshLess](#) and more.
- As of Drupal 8.2, caching of 404 responses and breadcrumbs have both been optimised to improve performance.

BigPipe

Activating the experimental “beta-stability” BigPipe Module in Drupal 8.2 core improves the user experience for your site visitors by reducing the perceived page loading time. Drupal 7 can’t really cache its output because it lacks metadata for caching. In Drupal 7 (and just about every other CMS or framework), personalization has always made things run slower. Using BigPipe in Drupal 8, it is no longer so. Drupal 8 includes cacheability metadata and knows which parts of every page are static and which dynamic. BigPipe then sends the unchanging parts of a page to the browser immediately while rendering and delivering the dynamic parts later, as soon as they are ready. Essentially, your site visitors see what they came to see almost immediately—the main content and images, for example—while the uncacheable, personalized page elements (such as a shopping cart block) are delivered once rendered.

The changes to Drupal 8’s rendering pipeline that make BigPipe possible are also what made ESI and Drupal 8’s Dynamic Page Cache possible.

Learn more about BigPipe in Drupal 8 [in this blog post and in this webinar – BigPipe: The Architecture Behind the Fastest Version of Drupal Yet – with Wim Leers.](#)

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 our earlier sense of self-sufficiency and rejection of third-party code, expressed as “[Not Invented Here](#)”, “Proudly Invented Elsewhere” represents a mind-shift among Drupal core developers. One of the great strengths of open source software is in not having to reinvent the wheel and being able to build better solutions “on the shoulders of giants.” It means opening our open source project and finding the best tool for the job, versus creating something custom and specific to Drupal. So we benefit from having a broader community of contributors and users, and when we make improvements to these open source tools, everyone who uses them benefits, too.

You’ll see this philosophy change 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.

In Drupal 8, the Entity API has been completely overhauled to greatly improve the developer experience and fill in the gaps in functionality compared to the Drupal 7 core API. All entities are now classed objects that implement a standard Entity Interface (no more guessing which of the 100 entity hooks you’re required to implement), with baked-in knowledge about the active language (to aid in translation and localization). Compare and contrast how this is done in D7 and D8:

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](#).

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 are formatted slightly 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()

```
**
* 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.”

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

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

src/Controller/Hello.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 been replaced by 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 (from back in 2001 when Drupal was new and when there was little support for OO code in PHP3), 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.
- There is no consistency between what various 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 (though 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) as well as brand new code in Drupal 8 like Migrate 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, located 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 can take 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):
2. In the [referenced class](#), it 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 priority scoring is the opposite of Drupal's weight system—higher-numbered methods are called first):

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

```
<?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 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));
}
?>
```

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.

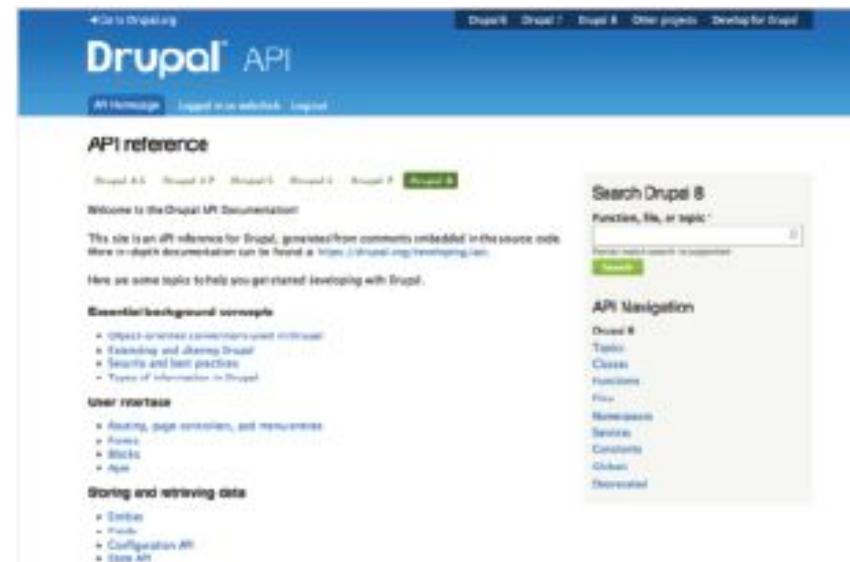
Debugging events and finding their implementations is also pretty straightforward. Instead of a handful of procedural PHP functions that may or may not have been used to call your hook, the same Event

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 takes the place of 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. 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! Learn more about the Drupal Console in this webinar with Jesus Olivas.
- 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.

Change records for Drupal core

[Add new change record](#)

Keywords

Introduced in branch

Introduced in version

Change mode created
 is greater than or equal to

Impacts
 Site builders, administrators, editors
 Module developers
 Themers

Introduced in branch/version	Notice created	Change
8.x	21-Mar-2014	REST URI paths changed to canonical paths
8.x	17-Mar-2014	Pugins can depend on a module
8.x	17-Mar-2014	<code>.module</code> and <code>.profile</code> files are no longer required; <code>ModuleHandler::getModuleList()</code> now returns Extension objects
8.x / 8.x-dev	16-Mar-2014	Distribution level settings added to install profiles
8.x	13-Mar-2014	<code>drupal_load()</code> has been removed
8.x	12-Mar-2014	Shortened directory structure for some plugin types
8.x	12-Mar-2014	'show' variable removed from admin_block theme hook
8.x	10-Mar-2014	COMMENT_HIDDEN & COMMENT_CLOSED & COMMENT_OPEN converted to constants on CommentItemInterface
8.x / 8.x	10-Mar-2014	<code>user_authenticate()</code> has been replaced by a 'user.auth' service
8.x	07-Mar-2014	show and hide functions removed from Twig in favor of a new 'without' Twig filter
8.x	07-Mar-2014	<code>drupal_get_filename()</code> always returns pathname of main extension file
8.x	06-Mar-2014	<code>drupal_implode_tags()</code> and <code>drupal_explode_tags()</code> replaced with <code>Drupal\Component\Utility\Tags</code> class
8.x	05-Mar-2014	PluginBases have been moved to Drupal\Core
8.x / 8.0-alpha9	04-Mar-2014	Add ability to define fields and field alterations for specific entity bundles

Your Burning Questions

When should I start using Drupal 8? 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 + contributed functionality, Drupal runs [some of the biggest and most important sites on the web](#).

Drupal 8 builds on the success of Drupal 7's approach by incorporating more expected "product" functionality out-of-the-box, such as authoring experience improvements, complete multilingual functionality, and numerous site builder features like the Settings Tray module and the Place Block module and more. 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.

How does Drupal's release cycle work now

Starting with Drupal 8.0.0, the Drupal project has moved 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 Drupalists will no longer have to wait 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 need to get your module updated to Drupal 8 ASAP. Each day, more Drupal 7 modules are being migrated over to Drupal 8 and new ones are being created for D8. The more solutions and functionality we have ready, the better off we will all be in the Drupal community.
- If you're a documentation author, translator, or designer, you can make an impact by working on user-facing documentation, translations, and themes.

It's a great time to start a Drupal 8 project. As of the release of Drupal 8.2, lots of Drupal 8 sites are online and a number of large Drupal 8 projects are underway. Many contributed modules necessary for complex projects have been released, upgraded for the new platform, and new ones are appearing every day.

So if I choose Drupal 7 now ... ?

Drupal 7 is still a stable, mature, robust, powerful, well-supported framework that will be maintained with bug fixes until after the LTS release of Drupal 8. It will be covered for official 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. However, you can expect most new functionality in contrib will soon only be released for Drupal 8, so you should factor that into your decision.

If you want to make the conservative choice for starting your Drupal 8 project, keep your eyes on the [Drupal project usage graph](#). When the D7 and D8 lines cross, it'll be a good time for you to make the jump, because it means there will be more D8 than D7 sites, so most of the hard work will have been done for you already.

Drupal 8 Core Feature	Drupal 7 Contrib Equivalent
WYSIWYG	CKEditor
In-Place Editing	Quick Edit
Responsive Toolbar	Mobile Friendly Navigation Toolbar
Responsive Front-End Theme	Omega, Zen, Adaptive, Aurora, etc. base themes
Responsive Admin Theme	Ember
Responsive Images	Picture
Responsive Tables	Responsive Tables
Simplified Overlay	Escape Admin
Multilingual	Internationalization Entity Translation (and several additional modules)
Better Blocks	Bean
Configuration Management	Features (provides exportable files that can be used in deployments)
Web Services	RESTful Web Services

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 provides 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 [Twigify](#) project that attempts to automate some 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've stuck to well-vetted contributed modules over custom code. If possible, make your plans accordingly.

For other tips on making your Drupal 6/7 site Drupal 8 ready, check out [Getting your site ready for Drupal 8](#) on the Acquia Developer Portal.

How Can I Help?

Want Drupal 8 to be even better? It can be, 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 3,500 contributors to Drupal 8 so far!](#) [Now, join them!](#)