Pro Drupal Development, Second Edition

Copyright © 2008 by John K. VanDyk

ISBN-13 (pbk): 978-1-4302-0989-8

ISBN-13 (electronic): 978-1-4302-0990-4

Printed and bound in the United States of America 9 8 7 6 5 4 3 2 1

Lead Editor: Matt Wade
Technical Reviewer: Robert Douglass
Editorial Board: Clay Andres, Steve Anglin, Ewan Buckingham, Tony Campbell, Gary Cornell, Jonathan Gennick, Matthew Moodie, Joseph Ottinger, Jeffrey Pepper, Frank Pohlmann, Ben Renow-Clarke, Dominic Shakeshaft, Matt Wade, Tom Welsh
Project Manager: Beth Christmas
Copy Editors: Heather Lang and Damon Larson
Associate Production Director: Kari Brooks-Copony
Production Editor: Laura Esterman
Compositor: Linda Weidemann, Wolf Creek Press
Proofreaders: April Eddy and Linda Siefert
Indexer: John Collin
Cover Designer: Kurt Krames
Manufacturing Director: Tom Debolski

The source code for this book is available to readers at http://www.apress.com.

CHAPTER 1

How Drupal Works

In this chapter, I'll give you an overview of Drupal. Details on how each part of the system works will be provided in later chapters. Here, we'll cover the technology stack on which Drupal runs, the layout of the files that make up Drupal, and the various conceptual terms that Drupal uses, such as nodes, hooks, blocks, and themes.

What Is Drupal?

Drupal is used to build web sites. It's a highly modular, open source web content management framework with an emphasis on collaboration. It is extensible, standards-compliant, and strives for clean code and a small footprint. Drupal ships with basic core functionality, and additional functionality is gained by enabling built-in or third-party modules. Drupal is designed to be customized, but customization is done by overriding the core or by adding modules, not by modifying the code in the core. Drupal's design also successfully separates content management from content presentation.

Drupal can be used to build an Internet portal; a personal, departmental, or corporate web site; an e-commerce site; a resource directory; an online newspaper; an image gallery; an intranet, to mention only a few possibilities. It can even be used to teach a distance-learning course.

A dedicated security team strives to keep Drupal secure by responding to threats and issuing security updates. A nonprofit organization called the Drupal Association supports Drupal by improving the `drupal.org` web site infrastructure and organizing Drupal conferences and events. And a thriving online community of users, site administrators, designers, and web developers work hard to continually improve the software; see `http://drupal.org` and `http://groups.drupal.org`.

Technology Stack

Drupal's design goals include both being able to run well on inexpensive web hosting accounts and being able to scale up to massive distributed sites. The former goal means using the most popular technology, and the latter means careful, tight coding. Drupal's technology stack is illustrated in Figure 1-1.

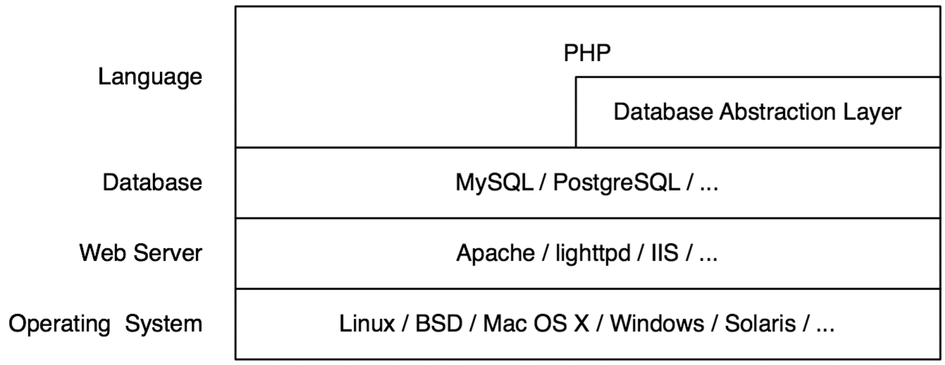
Language
PHP
Database Abstraction Layer
Database
MySQL / PostgreSQL / ...
Web Server
Apache / lighttpd / IIS / ...
Operating System
Linux / BSD / Mac OS X / Windows / Solaris / ...

Figure 1-1. *Drupal's technology stack*

The operating system is at such a low level in the stack that Drupal does not care much about it. Drupal runs successfully on any operating system that supports PHP.

The web server most widely used with Drupal is Apache, though other web servers (including Microsoft IIS) may be used. Because of Drupal's long history with Apache, Drupal ships with `.htaccess` files that secure the Drupal installation. *Clean URLs*—that is, those devoid of question marks, ampersands, or other strange characters—are achieved using Apache's `mod_rewrite` component. This is particularly important because when migrating from another content management system or from static files, the URLs of the content need not change, and unchanging URIs are cool, according to Tim Berners-Lee (`http://www.w3.org/Provider/Style/URI`). Clean URLs are available on other web servers by using the web server's URL rewriting capabilities.

Drupal interfaces with the next layer of the stack (the database) through a lightweight database abstraction layer. This layer handles sanitation of SQL queries and makes it possible to use different vendors' databases without refactoring your code. The most widely tested databases are MySQL and PostgreSQL, though support for Microsoft SQL Server and Oracle is increasing.

Drupal is written in PHP. Since PHP is an easy language to learn, there are many PHP programs written by beginners. The quality of beginner's code has given PHP a bad reputation. However, PHP can also be used to write solid code. All core Drupal code adheres to strict coding standards (`http://drupal.org/nodes/318`) and undergoes thorough review through the open source process. For Drupal, the easy learning curve of PHP means that there is a low barrier to entry for contributors who are just starting out, and the review process ensures this ease of access comes without sacrificing quality in the end product. And the feedback beginners receive from the community helps to improve their skills.

Core

A lightweight framework makes up the Drupal *core*. This is what you get when you download Drupal from `drupal.org`. The core is responsible for providing the basic functionality that will be used to support other parts of the system.

The core includes code that allows the Drupal system to bootstrap when it receives a request, a library of common functions frequently used with Drupal, and modules that provide basic functionality like user management, taxonomy, and templating as shown in Figure 1-2.

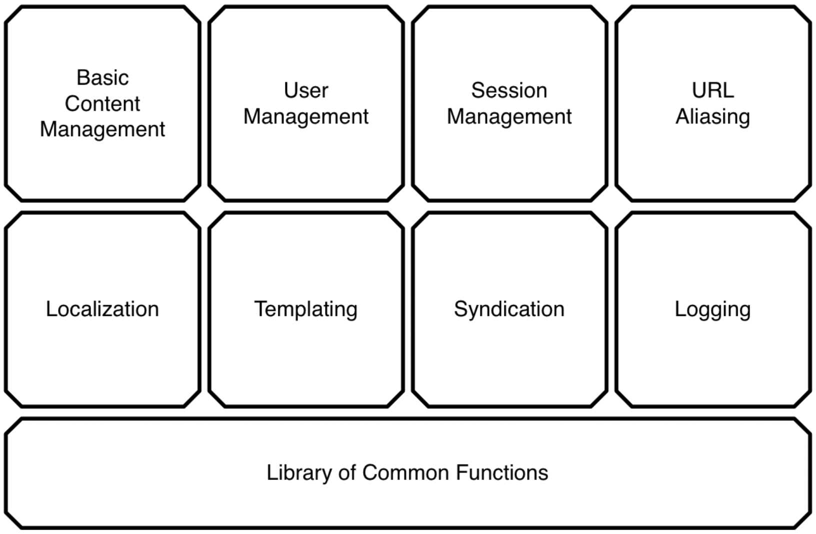

Figure 1-2. *An overview of the Drupal core (not all core functionality is shown)*

Administrative Interface

The administrative interface in Drupal is tightly integrated with the rest of the site and, by default, uses the same visual theme. The first user, user 1, is the superuser with complete access to the site. After logging in as user 1, you'll see an Administer link within your user block (see the "Blocks" section). Click that, and you're inside the Drupal administrative interface. Each user's block will contain different links depending on his or her access levels for the site.

Modules

Drupal is a truly modular framework. Functionality is included in *modules*, which can be enabled or disabled (some required modules cannot be disabled). Features are added to a Drupal web site by enabling existing modules, installing modules written by members of the Drupal community, or writing new modules. In this way, web sites that do not need certain features can run lean and mean, while those that need more can add as much functionality as desired. This is shown in Figure 1-3.

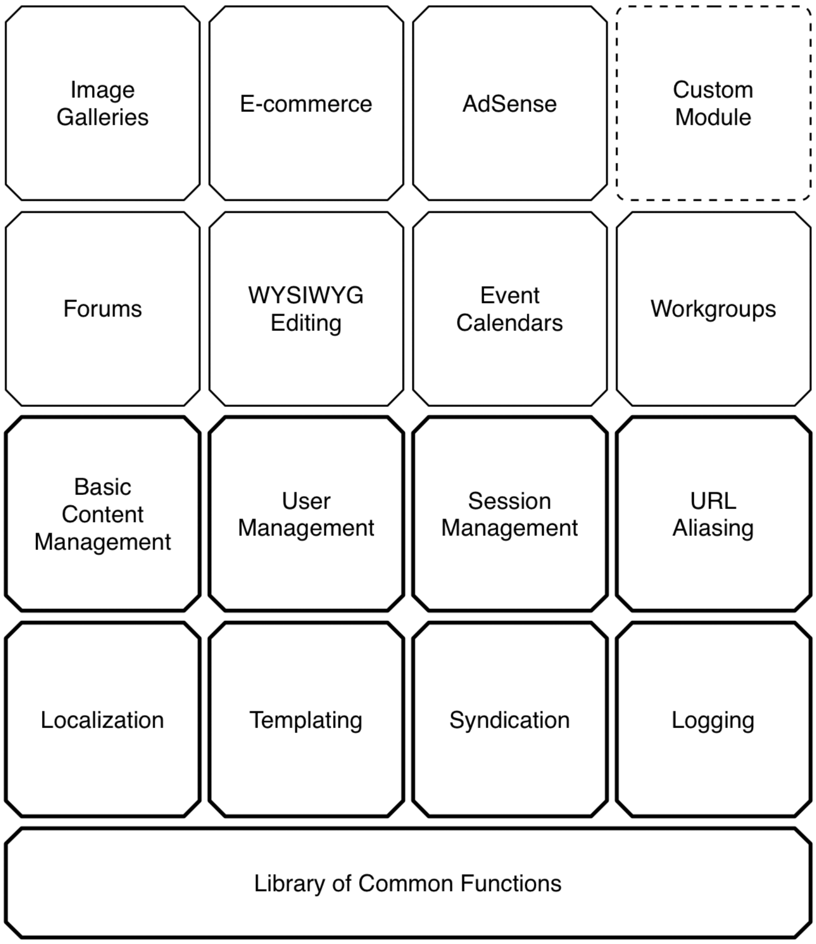

Figure 1-3. *Enabling additional modules gives more functionality.*

Both the addition of new content types such as recipes, blog posts, or files, and the addition of new behaviors such as e-mail notification, peer-to-peer publishing, and aggregation are handled through modules. Drupal makes use of the *inversion of control* design pattern, in which modular functionality is called by the framework at the appropriate time. These opportunities for modules to do their thing are called *hooks*.

Hooks

Hooks can be thought of as internal Drupal events. They are also called *callbacks*, though because they are constructed by function-naming conventions and not by registering with a listener, they are not truly being called back. Hooks allow modules to "hook into" what is happening in the rest of Drupal.

Suppose a user logs into your Drupal web site. At the time the user logs in, Drupal fires the user hook. That means that any function named according to the convention `module` *name* plus *hook name* will be called. For example, `comment_user()` in the comment module, `locale_user()` in the locale module, `node_user()` in the node module, and any other similarly named functions will be called. If you were to write a custom module called `spammy.module` and include a function called `spammy_user()` that sent an e-mail to the user, your function would be called too, and the hapless user would receive an unsolicited e-mail at every login.

The most common way to tap into Drupal's core functionality is through the implementation of hooks in modules.

■ **Tip** For more details about the hooks Drupal supports, see the online documentation at `http://api.drupal.org/api/6`, and look under Components of Drupal, then "Module system (Drupal hooks)."

Themes

When creating a web page to send to a browser, there are really two main concerns: assembling the appropriate data and marking up the data for the Web. In Drupal, the theme layer is responsible for creating the HTML (or JSON, XML, etc.) that the browser will receive. Drupal can use several popular templating approaches, such as Smarty, Template Attribute Language for PHP (PHPTAL), and PHPTemplate.

The important thing to remember is that Drupal encourages separation of content and markup.

Drupal allows several ways to customize and override the look and feel of your web site. The simplest way is by using a cascading style sheet (CSS) to override Drupal's built-in classes and IDs. However, if you want to go beyond this and customize the actual HTML output, you'll find it easy to do. Drupal's template files consist of standard HTML and PHP. Additionally, each dynamic part of a Drupal page (such as a box, list, or breadcrumb trail) can be overridden simply by declaring a function with an appropriate name. Then Drupal will use your function instead to create that part of the page.

Nodes

Content types in Drupal are derived from a single base type referred to as a *node*. Whether it's a blog entry, a recipe, or even a project task, the underlying data structure is the same. The genius behind this approach is in its extensibility. Module developers can add features like ratings, comments, file attachments, geolocation information, and so forth for nodes in general without worrying about whether the node type is blog, recipe, or so on. The site administrator can then mix and match functionality by content type. For example, the administrator may choose to enable comments on blogs but not recipes or enable file uploads for project tasks only.

Nodes also contain a base set of behavioral properties that all other content types inherit. Any node can be promoted to the front page of the web site, published or unpublished, or even searched. And because of this uniform structure, the administrative interface is able to offer a batch editing screen for working with nodes.

Blocks

A *block* is information that can be enabled or disabled in a specific location on your web site's template. For example, a block might display the number of current active users on your site. You might have a block containing links to the most popular content on the site, or a list of upcoming events. Blocks are typically placed in a template's sidebar, header, or footer. Blocks can be set to display on nodes of a certain type, only on the front page, or according to other criteria.

Often blocks are used to present information that is customized to the current user. For example, the user block only contains links to the administrative areas of the site to which the current user has access, such as the "My account" page. Regions where blocks may appear (such as the header, footer, or right or left sidebar) are defined in a site's theme; placement and visibility of blocks within those regions is managed through the web-based administrative interface.

File Layout

Understanding the directory structure of a default Drupal installation will teach you several important best practices such as where downloaded modules and themes should reside and how to have different Drupal installation profiles. A default Drupal installation has the structure shown in Figure 1-4.

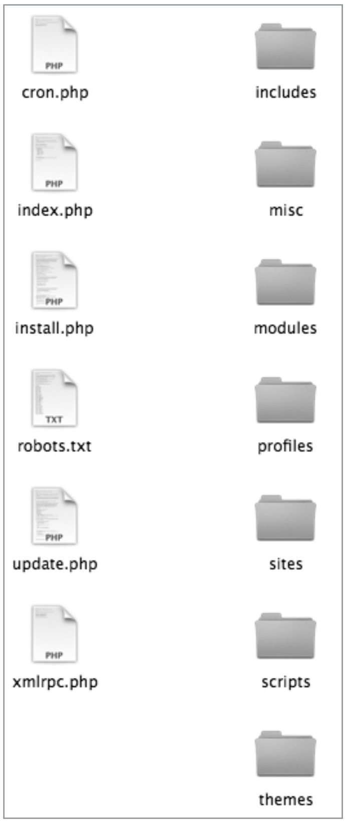

Figure 1-4. *The default folder structure of a Drupal installation*

Details about each element in the folder structure follow:

The `includes` folder contains libraries of common functions that Drupal uses.

The `misc` folder stores JavaScript and miscellaneous icons and images available to a stock Drupal installation.

The `modules` folder contains the core modules, with each module in its own folder. It is best not to touch anything in this folder (or any other folder except `profiles` and `sites`). You add extra modules in the `sites` directory.

The `profiles` folder contains different installation profiles for a site. If there are other profiles besides the default profile in this subdirectory, Drupal will ask you which profile you want to install when first installing your Drupal site. The main purpose of an installation profile is to enable certain core and contributed modules automatically. An example would be an e-commerce profile that automatically sets up Drupal as an e-commerce platform.

The `scripts` folder contains scripts for checking syntax, cleaning up code, running Drupal from the command line, and handling special cases with `cron`. This folder is not used within the Drupal request life cycle; these are shell and Perl utility scripts.

The `sites` directory (see Figure 1-5) contains your modifications to Drupal in the form of settings, modules, and themes. When you add modules to Drupal from the contributed modules repository or by writing your own, they go into `sites/all/modules`. This keeps all your Drupal modifications within a single folder. Inside the `sites` directory will be a subdirectory named `default` that holds the default configuration file for your Drupal site—`default.settings.php`. The Drupal installer will modify these original settings based on the information you provide and write a `settings.php` file for your site. The default directory is typically copied and renamed to the URL of your site by the person deploying the site, so your final settings file would be at `sites/www.example.com/settings.php`.

The `sites/default/files` folder doesn't ship with Drupal by default, but it is needed to store any files that are uploaded to your site and subsequently served out. Some examples are the use of a custom logo, enabling user avatars, or uploading other media associated with your new site. This subdirectory requires read and write permissions by the web server that Drupal is running behind. Drupal's installer will create this subdirectory if it can and will check that the correct permissions have been set.

The `themes` folder contains the template engines and default themes for Drupal. Additional themes you download or create should not go here; they go into `sites/all/themes`.

`cron.php` is used for executing periodic tasks, such as pruning database tables and calculating statistics.

`index.php` is the main entry point for serving requests.

`install.php` is the main entry point for the Drupal installer.

`update.php` updates the database schema after a Drupal version upgrade.

`xmlrpc.php` receives XML-RPC requests and may be safely deleted from deployments that do not intend to receive XML-RPC requests.

`robots.txt` is a default implementation of the robot exclusion standard.

Other files not listed here are documentation files.

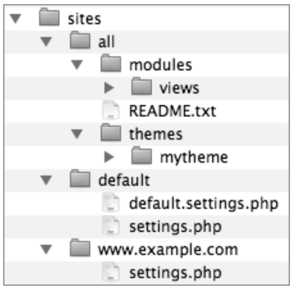
```
▼ sites
  ▼ all
    ▼ modules
      ▶ views
      README.txt
    ▼ themes
      ▶ mytheme
  ▼ default
      default.settings.php
      settings.php
  ▼ www.example.com
      settings.php
```

Figure 1-5. *The sites folder can store all your Drupal modifications.*

Serving a Request

Having a conceptual framework of what happens when a request is received by Drupal is helpful, so this section provides a quick walk-through. If you want to trace it yourself, use a good debugger, and start at `index.php`, which is where Drupal receives most of its requests. The sequence outlined in this section may seem complex for displaying a simple web page, but it is rife with flexibility.

The Web Server's Role

Drupal runs behind a web server, typically Apache. If the web server respects Drupal's `.htaccess` file, some PHP settings are initialized, and the URL is examined. Almost all calls to Drupal go through `index.php`. For example, a call to `http://example.com/foo/bar` undergoes the following process:

1. The `mod_rewrite` rule in Drupal's `.htaccess` file looks at the incoming URL and separates the base URL from the path. In our example, the path is `foo/bar`.
2. This path is assigned to the URL query parameter `q`.
3. The resulting URL is `http://example.com/index.php?q=foo/bar`.
4. Drupal treats `foo/bar` as the internal Drupal path, and processing begins in `index.php`.

As a result of this process, Drupal treats `http://example.com/index.php?q=foo/bar` and `http://example.com/foo/bar` exactly the same way, because internally the path is the same in both cases. This enables Drupal to use URLs without funny-looking characters in them. These URLs are referred to as clean URLs.

In alternate web servers, such as Microsoft IIS, clean URLs can be achieved using a Windows Internet Server Application Programming Interface (ISAPI) module such as ISAPI Rewrite. IIS version 7 and later may support rewriting directly.

The Bootstrap Process

Drupal bootstraps itself on every request by going through a series of bootstrap phases. These phases are defined in `bootstrap.inc` and proceed as described in the following sections.

Initialize Configuration

This phase populates Drupal's internal configuration array and establishes the base URL (`$base_url`) of the site. The `settings.php` file is parsed via `include_once()`, and any variable or string overrides established there are applied. See the "Variable Overrides" and "String Overrides" sections of the file `sites/all/default/default.settings.php` for details.

Early Page Cache

In situations requiring a high level of scalability, a caching system may need to be invoked before a database connection is even attempted. The early page cache phase lets you include (with `include()`) a PHP file containing a function called `page_cache_fastpath()`, which takes over and returns content to the browser. The early page cache is enabled by setting the `page_cache_fastpath` variable to `TRUE`, and the file to be included is defined by setting the `cache_inc` variable to the file's path. See the chapter on caching for an example.

Initialize Database

During the database phase, the type of database is determined, and an initial connection is made that will be used for database queries.

Hostname/IP-Based Access Control

Drupal allows the banning of hosts on a per-hostname/IP address basis. In the access control phase, a quick check is made to see if the request is coming from a banned host; if so, access is denied.

Initialize Session Handling

Drupal takes advantage of PHP's built-in session handling but overrides some of the handlers with its own to implement database-backed session handling. Sessions are initialized or reestablished in the session phase. The global `$user` object representing the current user is also initialized here, though for efficiency not all properties are available (they are added by an explicit call to the `user_load()` function when needed).

Late Page Cache

In the late page cache phase, Drupal loads enough supporting code to determine whether or not to serve a page from the page cache. This includes merging settings from the database into the array that was created during the initialize configuration phase and loading or parsing module code. If the session indicates that the request was issued by an anonymous user and page caching is enabled, the page is returned from the cache and execution stops.

Language Determination

At the language determination phase, Drupal's multilingual support is initialized and a decision is made as to which language will be used to serve the current page based on site and user settings. Drupal supports several alternatives for determining language support, such as path prefix and domain-level language negotiation.

Path

At the path phase, code that handles paths and path aliasing is loaded. This phase enables human-readable URLs to be resolved and handles internal Drupal path caching and lookups.

Full

This phase completes the bootstrap process by loading a library of common functions, theme support, and support for callback mapping, file handling, Unicode, PHP image toolkits, form creation and processing, mail handling, automatically sortable tables, and result set paging. Drupal's custom error handler is set, and all enabled modules are loaded. Finally, Drupal fires the init hook, so that modules have an opportunity to be notified before official processing of the request begins.

Once Drupal has completed bootstrapping, all components of the framework are available. It is time to take the browser's request and hand it off to the PHP function that will handle it. The mapping between URLs and functions that handle them is accomplished using a callback registry that takes care of both URL mapping and access control. Modules register their callbacks using the menu hook (for more details, see Chapter 4).

When Drupal has determined that there exists a callback to which the URL of the browser request successfully maps and that the user has permission to access that callback, control is handed to the callback function.

Processing a Request

The callback function does whatever work is required to process and accumulate data needed to fulfill the request. For example, if a request for content such as `http://example.com/q=node/3` is received, the URL is mapped to the function `node_page_view()` in `node.module`. Further processing will retrieve the data for that node from the database and put it into a data structure. Then, it's time for theming.

Theming the Data

Theming involves transforming the data that has been retrieved, manipulated, or created into HTML (or XML or other output format). Drupal will use the theme the administrator has selected to give the web page the correct look and feel. The resulting output is then sent to the web browser (or other HTTP client).

Summary

After reading this chapter, you should understand in general how Drupal works and have an overview of what happens when Drupal serves a request. The components that make up the web page serving process will be covered in detail in later chapters.