



Community Experience Distilled

Migrating to Drupal 7

Learn how to quickly and efficiently migrate content into Drupal 7 from a variety of sources including Drupal 6 using automated migration and import processes

Trevor James

[PACKT] open source*
PUBLISHING community experience distilled

www.it-ebooks.info

Migrating to Drupal 7

Learn how to quickly and efficiently migrate content into Drupal 7 from a variety of sources including Drupal 6 using automated migration and import processes

Trevor James



BIRMINGHAM - MUMBAI

Migrating to Drupal 7

Copyright © 2012 Packt Publishing

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the author, nor Packt Publishing, and its dealers and distributors will be held liable for any damages caused or alleged to be caused directly or indirectly by this book.

Packt Publishing has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, Packt Publishing cannot guarantee the accuracy of this information.

First published: December 2012

Production Reference: 1141212

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham B3 2PB, UK.

ISBN 978-1-78216-054-0

www.packtpub.com

Cover Image by J.Blaminsky (milak6@wp.pl)

Credits

Author

Trevor James

Project Coordinator

Esha Thakker

Reviewers

Anutosh Ghosh

Sammy Spets

Proofreader

Aaron Nash

Acquisition Editor

Andrew Duckworth

Indexer

Hemangini Bari

Commissioning Editor

Harsha Bharwani

Production Coordinator

Prachali Bhiwandkar

Technical Editor

Vrinda Amberkar

Cover Work

Prachali Bhiwandkar

About the Author

Trevor James is a Drupal developer residing in Middletown, MD, USA. Trevor has been using Drupal intensively since 2007 and designing websites for over 15 years using a combination of HTML, CSS, ColdFusion, PHP, jQuery, and JavaScript.

Trevor's focus is on building Drupal-based web applications and portals for education, non-profit, and medical systems, and small business environments. He is interested in the best methods of integrating web services with Drupal sites, optimizing Drupal sites' performance, and using Drupal content types, Views, Panels, and other contributed modules to develop front-end interfaces that support data intensive websites.

He loves teaching people about Drupal and how to use this excellent open source content management framework. Trevor authored the following Packt books:

- *Drupal 7 Business Solutions* (<http://www.packtpub.com/drupal-7-business-solutions-to-build-powerful-web-site/book>), published in early 2012
- *Drupal Web Services* (<http://www.packtpub.com/drupal-web-services/book>), published in November 2010
- *Drupal 6 Performance Tips* (<https://www.packtpub.com/drupal-6-performance-tips-to-maximize-and-optimize-your-framework/book>), published in February 2010

Trevor created a 14-hour video tutorial series titled *Drupal 7* for **Virtual Training Company (VTC)** in 2011. The video is available via the VTC website at <http://www.vtc.com/products/Drupal-7-Tutorials.htm>.

Many thanks, as before, to the Packt's Editorial staff including Vrinda Amberkar, Andrew Duckworth, Shreerang Deshpande, Manali Mehta, and Esha Thakker for suggesting the initial outline and plan for this title and for asking me to write this book. Their guidance throughout the writing process has been excellent, as always.

Many thanks to Gayle Kelch and her team at the U.S. Fire Administration's National Fire Data Center for allowing me to use the USFA's National Fire Department Census (<https://apps.usfa.fema.gov/census>) data throughout the book. This data is in the public domain and available for download at the URL noted above.

I would like to thank my developer colleagues Chris Desautels, Rich Kucera, and Kris Weinhold for keeping me on my toes in terms of new Drupal developments, inspiring me with their development process, and for helping me to spread the Drupal knowledge to the masses.

Thanks to the reviewers of the book for their guidance on the book's development. Reviewers' suggestions and critiques remain a huge part of the process of making Packt books stronger and more accurate especially in an open source software environment that is changing by the minute.

As before, the book could not have been written without the support of my wife Veronica (a Drupal builder herself) and our twin daughters Francesca and Clare.

This one is for my colleagues at Howard Hughes Medical Institute who continue to do complex and out of the box Drupal development.

About the Reviewers

Anutosh Ghosh loves coding, but has worked extensively only in the world of PHP and its associated areas, for over five years now. He has a good knowledge of Magento, and has worked on the integration of Magento Web Services with SAP for more than two and a half years.

He is trying hard to figure out the jargon of Java as well, among other things. However, he likes to venture out into other technologies as and when he gets time.

When bored, he gets some recreation by watching cool movies and singing regional songs. However, he loves to poke around in forums and Stack Overflow, from time to time.

Today, whatever I have become is only because of my family, especially my mother, whose perseverance and experience has always been my base.

Sammy Spets has been making Drupal do wild things since 2004, which has been a real pleasure in his life. So much so that Sammy volunteered to be a core maintainer for Drupal 6 and a maintainer of the e-commerce module, which was the commerce module of choice way back when. For the e-commerce module, Sammy made design changes to the payment system, built a few modules to support payment gateways, and added PostgreSQL support among other things.

In 2008, IDG Australia contracted Sammy to design and lead the development of a hybrid Drupal/legacy platform. The platform allowed IDG developers to gradually migrate their websites and web applications over to Drupal 6, which was still in beta. In addition to the platform, Sammy was tasked with creating a module suite for IDG staff to create surveys and report on them. This module suite was built prior to webform and leveraged the power of the Drupal 6 Form API in all its glory. Sammy also trained IDG developers to develop modules and themes in Drupal 6.

Early in 2009, a short contract with Demonz Media in Sydney, Australia brought about some patches to Ubercart, which Demonz gladly contributed back to the community.

Following that, Sammy traveled to Louisville, Kentucky USA where he contributed code to improve the experience for developers extending Ubercart, using its API. Ryan Szrama introduced Sammy to Chick-fil-A and Lyle Mantooth introduced Sammy to Korean food and some amazing fried chicken.

In 2011, Sammy joined the Magicspark team building Drupal sites and maintaining servers. During this time Sammy built a services platform to feed webform data to Marketo and LoopFuse from client Drupal sites via Magicspark's servers. In addition to this, Sammy redeveloped the UI on the "Where to Buy" page of the Redwood Systems website using OpenLayers mapping.

Aside from the geeky stuff, Sammy loves to cook, fine tune recipes, play pool, carve turns on a snowboard, hit the gym, ride motorcycles, fine dine, and drink champagne. *Drupal 7 Guide to Migration* is the first book Sammy has worked on.

Sammy is willing to assist with migrations and can be contacted by his e-mail: sammys@sammypets.com.

I would like to thank Jason Chinn from Magicspark for the cool projects and for giving me the spare time to review this book. Thank you to my Mum, Anja Spets, for all her support over the years. Last, but not least, I thank my good friends, Martijn Blankers and Job de Graaff, for minimizing the distractions while I reviewed this book.

www.PacktPub.com

Support files, eBooks, discount offers and more

You might want to visit www.PacktPub.com for support files and downloads related to your book.

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at www.PacktPub.com and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at service@packtpub.com for more details.

At www.PacktPub.com, you can also read a collection of free technical articles, sign up for a range of free newsletters and receive exclusive discounts and offers on Packt books and eBooks.



<http://PacktLib.PacktPub.com>

Do you need instant solutions to your IT questions? PacktLib is Packt's online digital book library. Here, you can access, read and search across Packt's entire library of books.

Why Subscribe?

- Fully searchable across every book published by Packt
- Copy and paste, print and bookmark content
- On demand and accessible via web browser

Free Access for Packt account holders

If you have an account with Packt at www.PacktPub.com, you can use this to access PacktLib today and view nine entirely free books. Simply use your login credentials for immediate access.

Table of Contents

Preface	1
Chapter 1: Preparing Drupal for Content Migration	7
Preparing for migration	7
Required core modules	8
Required contributed modules	9
The Administration Menu module	9
Chaos Tool Suite (CTools)	10
Views	11
Job Scheduler	11
Features	12
Installing the Feeds module	12
Installing the Feeds Tamper module	15
Other import module considerations	17
Prepping your existing data for migration	17
Summary	18
Chapter 2: Starting a Migration Path	19
Creating a content type	19
Planning for the content type fields	21
Additional contributed modules needed	23
The Location module	23
The Location Feeds module	24
The Link module	24
Adding fields to the content type	25
The Location and text fields	25
Adding a Location field	25
Integer fields	29
Term references	31
Adding a vocabulary	31
Adding the term reference field	32

Node references	33
Installing the References module	33
Adding the node reference field	33
Testing our content types	35
Testing the node reference field	35
Changing the display formatter on the node reference field	36
Testing the term reference field	37
Content type summary	38
Migrating images and files	39
Summary	40
Chapter 3: Creating a Feeds Importer	41
Creating a feeds importer	42
Exporting, cloning, and deleting importers	42
Importer configuration	43
Basic settings	43
Fetcher	45
Parser	46
Processor	47
Mapping your importer	49
Adding a source and target	51
Running an import process	54
Summary	60
Chapter 4: Feeds Tamperers	61
Using the Feeds Tamper module	62
Adding a tamper plugin	64
Running the import with the tamper plugin	67
Testing the tamper plugin results	68
Summary	69
Chapter 5: Maintaining a Migration Path	71
Updating imports	72
Cloning importers	72
The cloning process	73
Running the import update	76
Summary	79
Chapter 6: Packaging Content Types and Feeds Importers	81
Features	82
Building a content type feature	82
Creating and enabling the feature	83
The custom feature module	86
Overriding your feature	88
The Diff module	88
Reviewing the override	89

Recreating the feature	90
Reverting features	92
The Feed Importer feature	93
Migrating your feature to another Drupal site	95
Summary	96
Chapter 7: Migration Using the Migrate Module	97
Migrate module installation and configuration	98
Installing Migrate	98
Configuring Migrate	100
Viewing a specific migration mapping	102
Running the migration	106
Summary	108
Chapter 8: Migrating Content from Earlier Drupal Versions	109
Upgrading Drupal 6 to 7	110
Upgrade Status and Upgrade Assist	114
Upgrading Drupal core	119
Migrating Drupal 6 CCK fields and content to Drupal 7	122
Upgrading your contributed modules	124
Summary	126
Chapter 9: Migrating from WordPress	127
Migrating content from WordPress to Drupal	128
Installing and configuring WordPress Migrate	130
WordPress Migrate configuration	132
Exporting a WXR file from WordPress	134
Summary	138
Index	139

Preface

Both new and seasoned users of the Drupal content management framework want to migrate content from other websites and sources into the Drupal system. You may have a website built in a different codebase and database system that you want to move into Drupal. As long as that system allows for exporting of data into CSV, RSS and/or XML, you can get this content imported to Drupal easily. With Drupal you can easily import this data into its MySQL-driven database and PHP-driven structure, simply by configuring powerful modules and running an import, all from the Drupal interface. You can set up a website using Drupal, literally in hours, that contains the exact same content as your older legacy site just by using these migration and import processes.

You can also migrate content into Drupal 7 from earlier versions of Drupal as well as other open source content management systems. This book will provide all the steps you need to migrate content into the Drupal framework in order to build a next generation dynamic website using the same content you've been hosting in another website application. You can build a Drupal website without sacrificing any of your existing content.

What this book covers

Chapter 1, Preparing Drupal for Content Migration, details how to prep your Drupal website for imports of content and data from a legacy content management system or other site. Performance considerations, core Drupal modules, required contributed modules, and setup of the **Feeds** module and the **Feeds Tamper** module will all be discussed. Other modules including the **Migrate** module will be detailed. We will also prep our data export and get it ready to import into Drupal.

Chapter 2, Starting a Migration Path, shows how to create your content type and fields in Drupal that will contain your imported content. We'll create a new Drupal content type, build out a map of the types of fields we'll be importing, create the fields and add them to the content type in Drupal, discuss migration of files including images, set up some image handling, and add validation to our fields. We'll also create entity reference fields, term reference fields, and field collections.

Chapter 3, Creating a Feeds Importer, dives into using the **Feeds** module to configure our importers. We'll create a feeds importer, and look at basic settings including exporting, cloning, deleting, and tampering. We'll tweak our feeds importer settings including the fetcher, parser, and processor. We'll attach a feeds importer to a content type and also use the standalone feeds importer form. You'll learn how to create the feed importer mapping in order to map your legacy data into the Drupal fields you've created. Then you'll run the initial import and test it.

Chapter 4, Feeds Tamperers, expands the configuration of our feeds importer by adding tamperers to the mapping. We'll use the **Feeds Tamper** module, create a tamper for a few of our fields, and use the tamper for multivalue piped and/or comma-separated data. We'll create a tamper to handle character sets, and other plugins including HTML, other text, string, and lists. We'll run an import using the tamper plugin and test the import to confirm that the tamper worked.

Chapter 5, Maintaining a Migration Path, explains how to manage and maintain our migration path, using feeds, over time. We'll clone importers and re-run our imports to update and replace content.

Chapter 6, Packaging Content Types and Feeds Importers, shows you how to use the powerful **Features** module to save your feeds configurations to code and store them in a module(s) format. You can then share and implement these modules across other sites you manage.

Chapter 7, Migration Using the Migrate Module, jumps into a discussion and demo of the **Migrate** module. We'll install and use the **Migrate** module to migrate content from other sources into our site.

Chapter 8, Migrating Content from Earlier Drupal Versions, shows you how to export your content from a Drupal 6 site and import it to a Drupal 7 site for migration and upgrade reasons. We'll run through the basic steps of a Drupal 6 to 7 upgrade path for content migration.

Chapter 9, Migrating from WordPress, takes content and data from the popular Wordpress blog application framework and migrates this blog content into your Drupal site. We'll use a module called **Wordpress Migrate** in this chapter.

What you need for this book

To run the examples in the book, you will need the following:

- Drupal 7 website running in a local environment or hosted environment
- Content or data in a CSV file that you will be importing into Drupal
- RSS or XML feeds that you want to import

For this book, it is assumed that you have a working installation of Drupal 7 on either your localhost server or on a hosted server. All the examples will be run on a localhost version of Drupal running in the MAMP environment but you can also work with XAMPP, WAMP, or any of the myriad of Drupal package installers that are available. I won't be describing the Drupal install process in detail as there are many resources out there that can help you install Drupal. So it's assumed that you have core Drupal installed and are ready to go. I will explain all install processes for the modules we're going to use including the **Feeds**, **Feeds Tamper**, **Migrate**, and **Wordpress Migrate** modules.

Who this book is for

This book is for Drupal users, website managers, webmasters, content editors, or developers who have already installed and configured a Drupal site and understand its web-based administration; and who want to import data from other sources and websites into the Drupal framework.


This book will have little in terms of programming or code. Everything we do in the book will be configured easily by using the Drupal administration interface and module admin screens. We will look at some code briefly when we set up our **Feature** modules. You do not need to have any previous MySQL or PHP experience to work through these examples.


Conventions

In this book, you will find a number of styles of text that distinguish between different kinds of information. Here are some examples of these styles, and an explanation of their meaning.

Code words in text are shown as follows: "Install this module, as you would do for any Drupal contributed module, to your `/sites/all/modules/contrib` directory".

New terms and **important words** are shown in bold. Words that you see on the screen, in menus or dialog boxes for example, appear in the text like this: "Go to your core modules admin screen and uncheck the **Toolbar** module to disable it".

[ Warnings or important notes appear in a box like this.]

[ Tips and tricks appear like this.]

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book – what you liked or may have disliked. Reader feedback is important for us to develop titles that you really get the most out of.

To send us general feedback, simply send an e-mail to feedback@packtpub.com, and mention the book title via the subject of your message.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide on www.packtpub.com/authors.

Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

Downloading the example code

You can download the example code files for all Packt books you have purchased from your account at <http://www.PacktPub.com>. If you purchased this book elsewhere, you can visit <http://www.PacktPub.com/support> and register to have the files e-mailed directly to you.

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books – maybe a mistake in the text or the code – we would be grateful if you would report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting <http://www.packtpub.com/support>, selecting your book, clicking on the **errata submission form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded on our website, or added to any list of existing errata, under the Errata section of that title. Any existing errata can be viewed by selecting your title from <http://www.packtpub.com/support>.

Piracy

Piracy of copyright material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works, in any form, on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at copyright@packtpub.com with a link to the suspected pirated material.

We appreciate your help in protecting our authors, and our ability to bring you valuable content.

Questions

You can contact us at questions@packtpub.com if you are having a problem with any aspect of the book, and we will do our best to address it.

1

Preparing Drupal for Content Migration

You want to start using Drupal to power your website but have many existing web pages and need to migrate this content to Drupal. You would like to redirect the existing site URLs to the new URL paths that you create in Drupal so that visitors to your site will be able to easily find content on the new site via their browser's bookmarks. You want to retain the overall "look and feel" of your existing content in your new Drupal website. Is this possible? Can you do this easily?

This chapter will answer these questions for you and show you how to prep your new Drupal installation, so that you can import all of your existing website content into the new Drupal site structure. We'll begin by installing and configuring the modules you will need to get your migration process rolling.

We'll cover the following topics in this chapter:

- Performance considerations and core Drupal modules that will be useful
- Installing and configuring the **Feeds** and **Feeds Tamper** modules
- Prepping your legacy data for migration

Preparing for migration

This chapter assumes that you have installed Drupal using the Standard install on either your localhost development environment or on a hosted web server. We're going to start from a core Drupal install using Drupal version 7.15, the latest Drupal version at the time of this book's writing. Let's get started.

First you should load your Drupal site's status report, and confirm that your core Drupal environment is working correctly and that you have the correct PHP configuration for your migration. In our local development version of Drupal 7.15, we can confirm that the site is running on a PHP 5.3.x application environment (in this case powered by MAMP) and has a PHP memory limit set relatively high at 512 MB.

This memory limit is reasonable for a development environment, though in a production server, you'll most likely want to run a memory limit from 96M to 128M. You can tweak the memory limit using a few methods.

In an application such as MAMP Pro, you can simply tweak your `php.ini` file by editing the loaded PHP template via the MAMP Pro interface. You can also add the following line of code to your Drupal site's `setting.php` file and then flush your Drupal cache:

```
ini_set('memory_limit', '96M');
```

You should also be comfortable with accessing your MySQL database for your Drupal site by using phpMyAdmin or another type of MySQL tool. MAMP or MAMP Pro also provide an easy access to the phpMyAdmin interface by using the MAMP **WebStart** button in the MAMP interface. We'll be looking at tables in the Drupal site database, once we start running our imports. It's recommended that we run our migration process on a staging or development server before running the same migration on a production site.

Required core modules

Most of the Drupal core modules that we need will be installed automatically when we install our core Drupal 7.x site. The most important and crucial core modules are the following:

- Field
- Field SQL Storage
- Field UI
- Node
- Taxonomy

All of these should be enabled as they are required by Drupal core. Drupal 7.x core now includes the content construction kit concept and module as part of core. We do not need to install any additional modules at this point to be able to create content types. There may be other core modules you have or want to enable in your site. Go ahead and enable them now.

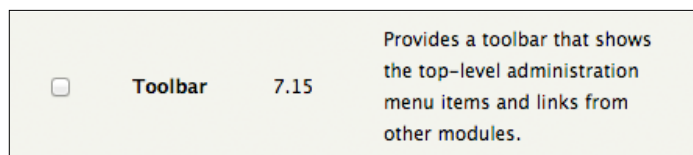
Required contributed modules

There are a number of Drupal contributed modules that we'll be using throughout the book, which we should install and configure now. Some of these modules are requirements and dependencies of the **Feeds** module that we'll be using for our migration processes in the first six chapters. The installation and configuration of each of these modules will be outlined in this section.

The Administration Menu module

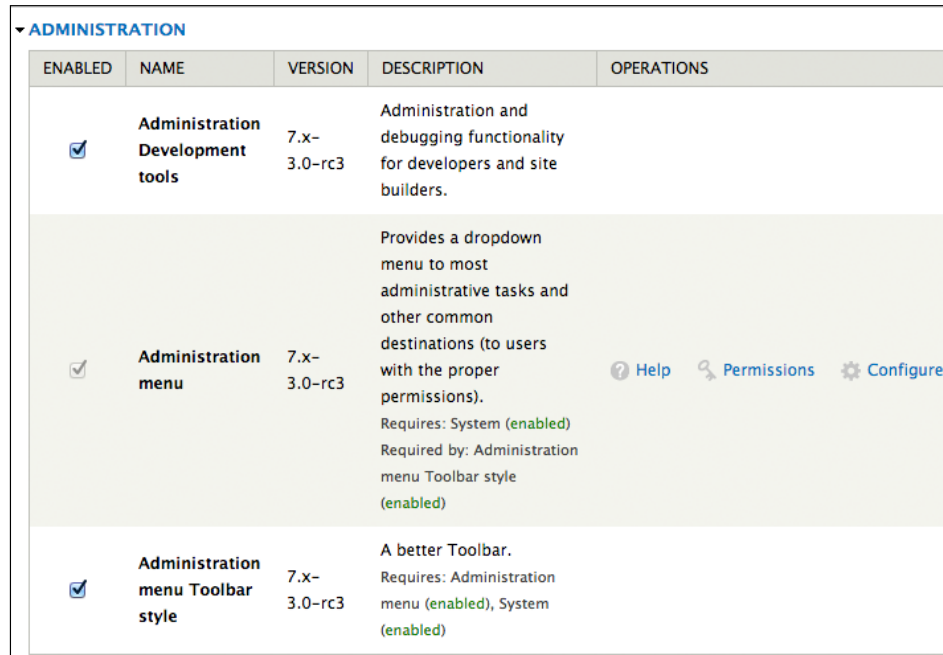
The **Administration Menu** module provides a helpful administration toolbar for the Drupal admin interface. By default core Drupal enables a core module called **Toolbar**. In this section we'll disable **Toolbar**, and then install and configure the **Administration Menu** module:

1. Go to your core modules admin screen and uncheck the **Toolbar** module to disable it. Then save your module configuration. This will remove the default black admin toolbar in the header of your site.



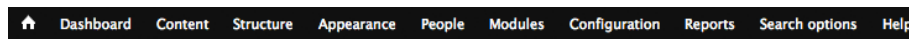
2. Download the latest 7.x stable version of the Administration Menu module from its project page at http://drupal.org/project/admin_menu/. The current version is 7.x-3.0-rc3.
3. Install this module, as you would do for any Drupal contributed module, to your `/sites/all/modules/contrib` directory.

4. Enable the following modules in the Administration fieldset of your module's screen:
 - **Administration Development tools**
 - **Administration menu**
 - **Administration menu Toolbar style**



ADMINISTRATION				
ENABLED	NAME	VERSION	DESCRIPTION	OPERATIONS
<input checked="" type="checkbox"/>	Administration Development tools	7.x-3.0-rc3	Administration and debugging functionality for developers and site builders.	
<input checked="" type="checkbox"/>	Administration menu	7.x-3.0-rc3	Provides a dropdown menu to most administrative tasks and other common destinations (to users with the proper permissions). Requires: System (enabled) Required by: Administration menu Toolbar style (enabled)	Help Permissions Configure
<input checked="" type="checkbox"/>	Administration menu Toolbar style	7.x-3.0-rc3	A better Toolbar. Requires: Administration menu (enabled), System (enabled)	

5. Refresh your module's page and you should now see the black Administration horizontal drop-down menu appear in your site's header area, as shown in the following screenshot:



Chaos Tool Suite (CTools)

The **Chaos Tool Suite (CTools)** module is a requirement of the **Feeds**, **Feeds Tamper**, and **Views** modules (all installed later in this chapter); so let's go ahead and install it now to get this requirement out of the way. To install CTools, follow these steps:

1. CTools can be downloaded from <http://drupal.org/project/ctools>. The latest version is 7.x-1.2.

2. Install it as you would any Drupal contributed module.
3. Once CTools is installed, load your module's admin screen and enable the **Chaos Tools core** module under the **Chaos Tool Suite** module fieldset.
4. The core module is the only one you'll need for now and the default installed configuration is fine. You do not need to make any additional configurations.

Views

We won't be using the **Views** module extensively in this book or for our migration processes, but **Views** is an indispensable Drupal module and you'll most likely want to use it to create lists of content on your site. It's also required by the **Feeds News** module, so let's install it now:

1. Download the latest version of **Views** from <http://drupal.org/project/views/>. The latest version is 7.x-3.5.
2. Install it as you would any Drupal contributed module.
3. Once **Views** is installed, load your module's admin screen and enable the **Views** and **Views UI** modules and save your modules' configuration.
4. To access the **Views** administration interface once you enable the module, you can go to **Structure | Views** from your admin menu or go to `admin/structure/views`.
5. No additional configuration needs to take place now.

Job Scheduler

The **Job Scheduler** module is required by the **Feeds** and **Feeds Tamper** modules, so let's install it now:

1. Download the latest version of **Job Scheduler** from http://drupal.org/project/job_scheduler. This is an alpha version of the module at version 7.x-2.0-alpha3 at the time of this book's writing.
2. Install it as you would any Drupal contributed module.
3. Once installed, load your module's admin screen and enable the **Job Scheduler** and **Job Scheduler Trigger** modules.
4. The **Job Scheduler** module is basically an API that we're loading into our Drupal site that provides many helper functions for Drupal developers. We do not need to actually configure the module. **Feeds** and **Feeds Tamper** will both hook into the module on their own.

Features

The **Features** module is a powerful Drupal module that allows you to package and save your Drupal configurations including content types and **Feed** importers into code. This means you can build a content type and then save your entire content type's configuration as a module. Then you can take this **Features** module and install it on another site. Then enable it and you'll have your entire content type on that other Drupal website. This module is extremely helpful if you have a development or staging site and you want to move a content type from the staging site to a production site without having to rebuild the type. You can just install and enable the module.

The **Features** module is also required by the **Feeds News** module, as **Feeds News** hooks into **Features** to create an example Feature module. So we'll install it for that purpose but we'll also be using the **Features** module extensively in *Chapter 6, Packaging Content Types and Feeds Importers*, when we will create **features** modules for our content type and Feeds importer configurations.

1. Download the latest version of **Features** from <http://drupal.org/project/features>. The latest version of the module is 7.x-1.0.
2. Install it as you would any Drupal contributed module.
3. Once installed, load your module's admin screen and enable the Features module.
4. The configuration screen for **Features** is located at **Structure | Features** via the admin menu or by going to `admin/structure/features`.
5. We will not be configuring or adding any **Features** in this chapter but we will be returning to this module in *Chapter 6, Packaging Content Types and Feeds Importers*.

We've completed installing our required modules. We can now move on to installing and configuring the **Feeds** and **Feeds Tamper** modules. These are the modules we'll be using to import our migrated content.

Installing the Feeds module

We're going to use the **Feeds** module to import our content. This module is extremely powerful and allows you to set up automated imports of content from multiple formats including CSV and OPML files; and RSS, XML, and ATOM feeds. The import process we will use in later chapters creates nodes and taxonomy terms in the target website. This is the method we're going to use to migrate our content. We'll be creating a **Feeds** module based importer that allows us to import data from a CSV file into nodes that are part of a content type in our site. These nodes will hold our individual imported content and act as our new web pages. We'll also import data to taxonomy vocabs on our site so we can use this data as tags.

Feeds support importing from large files that are full of thousands of rows of data and the import process can run in as quickly as two minutes. To get started using the **Feeds** module we need to install it first. Follow these steps to install:

1. Download the latest version of **Feeds** from <http://drupal.org/project/feeds/>. The latest version is 7.x-2.0-alpha5.
2. Install it as you would any Drupal contributed module.
3. Once installed, enable the following modules via your module's admin screen:
 - **Feeds**
 - **Feeds Admin UI**
 - **Feeds Import**
 - **Feeds News**

I've included a screenshot of what your module's admin screen in the **Feeds** fieldset should look like:

FEEDS				
ENABLED	NAME	VERSION	DESCRIPTION	OPERATIONS
<input checked="" type="checkbox"/>	Feeds	7.x-2.0-alpha5	Aggregates RSS/Atom/RDF feeds, imports CSV files and more. Requires: Chaos tools (enabled), Job Scheduler (enabled) Required by: Feeds Import (enabled), Feeds News (enabled), Feeds Tamper (enabled), Feeds Admin UI (enabled), Feeds Tamper Admin UI (enabled), Location Feeds (enabled)	Permissions
<input checked="" type="checkbox"/>	Feeds Admin UI	7.x-2.0-alpha5	Administrative UI for Feeds module. Requires: Feeds (enabled), Chaos tools (enabled), Job Scheduler (enabled) Required by: Feeds Tamper Admin UI (enabled)	Configure
<input checked="" type="checkbox"/>	Feeds Import	7.x-2.0-alpha5	An example of a node importer and a user importer. Requires: Feeds (enabled), Chaos tools (enabled), Job Scheduler (enabled)	
<input checked="" type="checkbox"/>	Feeds News	7.x-2.0-alpha5	A news aggregator built with feeds, creates nodes from imported feed items. With OPML import. Requires: Features (enabled), Feeds (enabled), Chaos tools (enabled), Job Scheduler (enabled), Views (enabled)	

4. Once enabled, click on the **Configure** link under the Operations column next to your **Feeds Admin UI** module. That will load the **Feeds importers** configuration screen from `admin/structure/feeds`. You can also get to this screen by going to **Structure | Feeds importers** from your admin menu.
5. The **Feeds importers** screen is the launch pad for creating, overriding, exporting or cloning an importer. We'll be doing this starting in *Chapter 3, Creating a Feeds Importer*.
6. **Feeds** loads with some default importers including the following:
 - **Node import** for importing data and content into nodes on your site
 - **User import** for importing users into your Drupal's user interface
 - **Feed** for importing from RSS or Atom feeds
 - **OPML import** for importing OPML files
7. The **Feed** importer is the only importer currently attached to a content type called Feed, on the site. In *Chapter 3, Creating a Feeds Importer*, we'll start configuring and using these importers and crafting our own content importer, but for now it's good to just view the defaults that the module ships with. You should see the following at this point:

Node import	Import nodes from CSV file.	[none]	Default	Override Export Clone Tamper	<input checked="" type="checkbox"/>
User import	Import users from CSV file.	[none]	Default	Override Export Clone Tamper	<input checked="" type="checkbox"/>
Feed	Import RSS or Atom feeds, create nodes from feed items.	Feed	Default	Override Export Clone Tamper	<input checked="" type="checkbox"/>
OPML import	Import subscriptions from OPML files. Use together with "Feed" configuration.	[none]	Default	Override Export Clone Tamper	<input checked="" type="checkbox"/>

Save

8. The **Feeds** module has another configuration screen that we'll be using later to run the actual import process of our content. You can get to this screen by clicking on the **Import** link that shows in the page description at the top of the **Feeds importers** screen in the intro text area (see preceding screenshot). Clicking on **Import** will take you to the following screen at the following path, `/import`:

Node import	Import nodes from CSV file.
User import	Import users from CSV file.
Feed	Import RSS or Atom feeds, create nodes from feed items.
OPML import	Import subscriptions from OPML files. Use together with "Feed" configuration.

The **Import** screen allows you to click on an import link and load the import administrative page. For example, loading the **Node** import at this point will load a page that then allows you to upload a CSV file and then click on the **Import** button to run the import. Again we'll be looking at this process in great detail starting in *Chapter 3, Creating a Feeds Importer*.

We've installed and enabled the **Feeds** module and taken a brief look at its default configuration screen. We're now ready to install another Feeds-based module that will hook into the **Feeds** module, called **Feeds Tamper**.

Installing the Feeds Tamper module

The **Feeds Tamper** module is a module that enhances the **Feeds** module by allowing you to add tamper plugins to your importer that you create in **Feeds**. Adding a tamper will allow you to treat and pre-process your imported data before it actually populates the Drupal database and your nodes. Tamper can be used to do things including but not limited to the following:

- Add a required field filter to your import process
- Decode and encode HTML entities
- Strip HTML tags from the import
- Explode and implode lists of data
- Format numbers
- Convert case
- Convert boolean values
- Find and replace data upon import
- Trim and truncate data

This module will be helpful to use when we want to import multiple values into a **Node Reference** or **Term Reference** field in our content type. We can add a pipe in place of a comma for instance across our entire imported data.

To install the **Feeds Tamper** module follow these steps:

1. Download the latest version of **Feeds Tamper** from http://drupal.org/project/feeds_tamper. The latest version is 7.x-1.0-beta3.
2. Install it as you would any Drupal contributed module.
3. Once installed, enable the **Feeds Tamper** and **Feeds Tamper Admin UI** modules on your module's admin screen in the **Feeds** fieldset section.
4. Once enabled, you can access the **Feeds Tamper** configuration by first returning to your **Feeds importers** screen and then clicking on the **Override** link next to an importer.
5. Once the actual importer configuration screen loads, click on the **Mapping** link at the bottom of the navigation block in the left sidebar.
6. When the mapping screen loads, click on the **Configure Feeds Tamper** link:

Mapping for Node processor

Define which elements of a single item of a feed (= Sources) map to which content pieces in Drupal. Make sure that at least one definition has a *Unique target*. A unique target means that a value for only occur once. E. g. only one item with the URL <http://example.com/content/1> can exist.

[Configure Feeds Tamper](#)

SOURCE	TARGET	UNIQUE TARGET
title	Title	<input type="checkbox"/>
body	Body	
published	Published date	
guid	GUID	<input checked="" type="checkbox"/>

Name of source field: Select a target:

7. This will launch the Tamper plugins screen, which will look similar to the following:

title -> Title

DESCRIPTION	PLUGIN	STATUS	OPERATIONS	ENABLED
No plugins defined.				
+ Add plugin				

body -> Body

DESCRIPTION	PLUGIN	STATUS	OPERATIONS	ENABLED
No plugins defined.				
+ Add plugin				

8. From this screen you can add your tamper plugins. We'll be doing this in *Chapter 4, Feeds Tamplers*.

We now have all of our **Feeds** modules installed and enabled and we're ready to start building our content type to hold our imported data. Once we have our content type built, we can return to the **Feeds** module and use it to build our importer for our migration process.

Other import module considerations

In *Chapter 7, Migration Using the Migrate Module*, we'll use another module to run migrations of content, called the Migrate module. We'll look at Migrate in more detail later in the book but for now you can review its project page on Drupal.org at <http://drupal.org/project/migrate/>. The current version of this module is 7.x-2.4.

Like the **Feeds** module, Migrate provides an interface in Drupal to import your content. Migrate also has integration with the Drush module, so you can run migrate commands using Drush in the command-line prompt.

Prepping your existing data for migration

In this book we're going to import data and content to our site using CSV files. The **Feeds** module easily allows us the ability to upload a CSV file and import its content into Drupal nodes. So you're going to need a CSV file to use, for the examples in this book. I'm providing a default CSV example file in the code package that comes with the book but you can also use your own CSV. You will want to follow these requirements for creating and saving your CSV file for use in the examples:

- Make sure you save your CSV file in UTF-8 encoding.
- Make sure you add column headers in your CSV.
- If you are adding multiple values to a cell, confirm that you have a consistent separation character. For example use commas to separate values. Otherwise use pipes.

- Confirm you can open your CSV file in an application such as Microsoft Excel, and also in a text editor such as TextWrangler.
- Confirm that you have named your column header titles using a consistent naming convention; for example, make sure you are using underscores instead of blank spaces in your column header titles. If you are importing a column of data, which contains a Google Map URL for example, name the title of that column as `google_map`. Also it's good practice to confirm that all your column header titles are in lowercase.

We've now prepped our CSV file and we're ready to start building our Drupal content type that will hold this imported content and data.

Summary

We have successfully prepped our Drupal site for our migration. In this chapter, we installed a number of required modules including **Views**, **Features**, **Job Scheduler**, and **CTools**. We also installed and took a look at the base configurations of the **Feeds** and **Feeds Tamper** modules. We discussed an alternative to the **Feeds** module called **Migrate** and we also prepped our CSV file for the migration process.

In the next chapter we will build our content type and add custom fields to it that will eventually contain our migrated content and data. We will also discuss how best to handle migration of images and files.

2

Starting a Migration Path

We have exported all of our existing website content to a CSV file. You have your content nicely organized into columns and rows in your CSV and it's ready to import into Drupal. But before you run the import we need to set up the container in Drupal to hold our imported and migrated content. In Drupal this container is called a **content type**. In this chapter we will build the content type to hold our migrated content and add fields to this content type to hold all of the data that we need to migrate.

We'll cover the following topics in this chapter:

- Creating a content type
- Planning a map of the fields in your content type to the data in your CSV
- Adding fields to your content-type based on the mapping
- Configuring field settings and validation
- Adding the Location fields
- Adding node reference fields
- Adding term reference fields
- Migrating images and files

Creating a content type

Let's jump right into creating a content type in Drupal to contain our migrated content and data. Our CSV file that we'll be using in this example contains about 24 columns of data and thousands of nodes. If you work with your own CSV, it may differ in the number of rows and columns. We're going to import about 1,000 rows of data into our Drupal site. Each row of data will become a Drupal node or page in our Drupal site. Each node in Drupal will contain all of the data that resides in each column per row from our CSV file. This is why it's very helpful to use a CSV file for importing, as you can easily visualize how the rows of data will import into a Drupal content type.



We'll be using data and content that is in the public domain and provided by the United States Fire Administration. Credit for the data goes to U.S. Fire Administration's (USFA's) National Fire Department Census (<http://apps.usfa.fema.gov/census/>). You can find a copy of the CSV file in the code directory provided with this book. You can also download your own copy of the data from <http://apps.usfa.fema.gov/census-download/main/download>.

We can leave the headers in our CSV columns as we'll use the header during the importer creation process using the **Feeds** module. One thing you should check on and confirm is that your column headers are in lowercase and the words are separated with underscores. So if the column header is titled **Dept Type**, you should change this to `dept_type`. Here's an example of the CSV file we'll be using:

B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
fire_dept_name	hq_addr_1	hq_addr2	hq_city	hq_state	hq_zip	mail_addr1	mail_addr2	mail_po_box	mail_city	mail_state	mail_zip	hq_phone	hq_fax	county
#9 Area Volunteer Fire Department	Route 1 Box 63 FD		Eufaula	OK	74432	Route 1 Box 63 FD			Eufaula	OK	74432	(918) 452-3763		MCINTOSH
101 Gamaliel Fire Protection District	5012 101 HWY	PO Box 30	Gamaliel	AR	72537			PO Box 30	Gamaliel	AR	72537-0030	870-467-545	870-467-557	BAXTER
103CES Fire Department CTANG	100 Nicholson RD		East Granby	CT	6026	100 Nicholson RD			East Granby	CT	06026-9309	860-292-252	860-292-247	HARTFORD
104th FW Barnes ANG Fire/Rescue	175 Falcon DR	104th FW Barn	Westfield	MA	1085	175 Falcon D 104th FW Barnes ANGB			Westfield	MA	1085	(413) 588-91	(413) 572-15	HAMPDEN
11 Point Rural Fire Association	908 E Business RT 60-63		Willow Sprin	MO	65793-3432	908 E Business RT 60-63			Willow Sprin	MO	65793-3432	(417) 469-91	(417) 469-91	HOWELL
111th Fighter Wing PaANG	2164 McGuire ST	111th CES/CEF	Willow Grow PA	PA	19090-5232	2164 McGuire 111th CES/CEF			Willow Grow PA	PA	19090-5232	215-443-149	215-443-187	MONTGOME
119th CES/CEF	1400 28th AVE		FARGO	ND	58102-1052	1400 28th AVE			FARGO	ND	58102-1052	(701) 451-22	(701) 451-21	CASS
131st Fighter Wing Fire Protection	10800 Lambert Interr Stop 28		Bridgeton	MO	63044-2371	10800 Lambi Stop 28			Bridgeton	MO	63044-2371	(314) 263-63	(314) 263-63	SAINT LOUIS
134th CES/CEF Tennessee Air National	124 Briscoe DR		McGhee Tys TN	TN	37777	124 Briscoe DR			McGhee Tys TN	TN	37777	865-985-331	865-985-338	BLOUNT
143rd Airlift Wing Fire Emergency Ser	11 Flightline DR		North Kingst RI	RI	2852	11 Flightline DR			North Kingst RI	RI	2852	401-886-130	401-886-005	WASHINGTON
148th CES Fire Department	4630 Mustang DR		Duluth	MN	55811-6036	4630 Mustang DR			Duluth	MN	55811-6036	(218) 788-74	(218) 788-74	SAINT LOUIS
151 CES/CEF Utah Air National Guard	765 N 2200 W		Salt Lake City UT	UT	84116-2926	765 N 2200 W			Salt Lake City UT	UT	84116-2926	801-245-218	801-245-218	SALT LAKE
156th CES/Fire Dept	200 AVE Jose A. Santana		Carolina	PR	00979-1502	200 AVE Jose A. Santana			Carolina	PR	00979-1502	(787) 253-52	(787) 253-52	CAROLINA
16 Springs Canyon Volunteer Fire Dep	764 16 Springs Canyon RD		Cloudcroft	NM	88317	764 16 Springs Canyon RD			Cloudcroft	NM	88317-9403	575-687-409	575-687-409	OTERO
161st Air National Guard	3200 E Old Tower RD		Phoenix	AZ	85034	3200 E Old Tower RD			Phoenix	AZ	85034	(602) 302-9232		MARICOPA
162 Fighter Wing Fire Department	1400 E Super Sabre D 162 CES/CEF		Tucson	AZ	85706-6052	1400 E Super 162 CES/CEF			Tucson	AZ	85706-6052	(520) 295-61	(520) 295-67	PIMA
165th Airlift Wing Fire Department	1401 Robert B. Miller Jr. DR		Garden City	GA	31408-9001	1401 Robert B. Miller Jr. DR			Garden City	GA	31408-9001	(912) 966-82	(912) 966-86	CHATHAM
167 CES Fire Department	222 Sabre Jet BLVD		Martinsburg WV	WV	25401-7704	222 Sabre Jet BLVD			Martinsburg WV	WV	25401-7704	(304) 262-5277		BERKELEY
174 Fighter Wing Fire Department	6001 E Molloy RD	Bldg. 644	Syracuse	NY	13211-7099	6001 E Molle Bldg. 644			Syracuse	NY	13211-7099	(315) 454-66	(315) 454-65	ONONDAGA
180th Ohio Air National Guard Fire De	2660 S Eber RD		Swanton	OH	43558-8752	2660 S Eber RD			Swanton	OH	43558-8752	419-868-411	419-868-431	LUCAS
182nd AW Fire Department Illinois IL	2401 S Falcon BLVD		Peoria	IL	61607-5013	2401 S Falcon BLVD			Peoria	IL	61607-5013	(309) 633-51	(309) 633-55	PEORIA
183d Fighter Wing Fire Department	3101 SW J. David Joni Capital Airport		Springfield	IL	62707-5000	3101 SW J. D Capital Airport			Springfield	IL	62707-5000	(217) 757-13	(217) 757-13	SANGAMON
185th Air Refueling Wing Fire Departn	2920 Headquarters AVE		Sioux City IA	IA	51111	2920 Headquarters AVE			Sioux City IA	IA	51111-1300	712-233-077	712-233-084	WOODBURY
186 Air Refueling Wing Fire & Emerge	6225 M ST	BLDG 155	Meridian	MS	39307-7222	6225 M ST BLDG 155			Meridian	MS	39307-7222	601-484-974	601-484-937	LAUDERDALE
188th Fighter Wing Fire Department	4850 Leigh AVE	BLDG 201	Fort Smith	AR	72903	4850 Leigh A BLDG 201			Fort Smith	AR	72903-6018	479-573-521	479-573-514	SEBASTIAN
1st District VFD/ Colman Park Fire Dep	38 County Rd 308		Iuka	MS	38859	38 County Rd 308			Iuka	MS	38859	(662) 423-9767		PRENTISS
231st Civil Engineering Flight - MO Air	10800 Lambert International BLVD		St Louis	MO	63044	10800 Lambert International BLVD			St Louis	MO	63044	(314) 527-64	(314) 527-64	SAINT LOUIS
233rd Firefighting Team CA Army Nat	850 All America City BLVD		Roseville	CA	95678-1594	850 All America City BLVD			Roseville	CA	95678-1594	916-782-808	916-782-808	PLACER
250 Volunteer Fire Department	HWY 250		Waldron	AR	72958			PO Box 1153	Waldron	AR	72958	(479) 637-3443		SCOTT
2604 Volunteer Fire Department	319 FM 2604		Whitney	TX	76692			PO Box 1784	Whitney	TX	76692-1784	254-694-4511		HILL
287 R/C Fire and Rescue	8616 S Highway 287		Corsicana	TX	75109-0655	8616 S Highway 287			Corsicana	TX	75109-0655	(903) 874-50	(903) 872-51	NAVARRO
3 Mile Corner VFD/ Scooba Dist # 1	Willie Gillespi C/O 3 A 78-A		Scooba	MS	39358	Willie Gillespi 78-A			Scooba	MS	39358	662-476-8264		KEMPER
3-G Volunteer Fire Company Inc.	Blue RD	PO Box 112	Glenfield	NY	13345	Blue RD			Glenfield	NY	13345	(315) 376-2135		LEWIS
3-N-1 Volunteer Fire Department	142 High ST		Rosanky	TX	78953			PO Box 51	Rosanky	TX	78953-0051	512-237-2893		BASTROP
4-Communities Fire Department	10901 Highway 98	PO Box 1625	Magnolia	AR	71754			PO Box 1625	Magnolia	AR	71754-1625	870-596-2517		COLUMBIA
421 Area Emergency Services Volunte	1758 Bristol Caverns HWY		Bristol	TN	37620	1758 Bristol Caverns HWY			Bristol	TN	37620	423-878-0054		SULLIVAN
482 SPTG/CEF Fire Department	29050 Coral Sea BLVD		Homestead / FL	FL	33039-1299	29050 Coral Sea BLVD			Homestead / FL	FL	33039-1299	(305) 224-74	(305) 224-77	MIAMI-DADE
4th District Volunteer Fire Departmen	30 Boyd Harvey RD		Jayess	MS	39641-8116	30 Boyd Harvey RD			Jayess	MS	39641-8116	601-876-288	601-876-288	WALTHAM
7 Hickory-Morgan Fire Protection Dist	17508 E County Road 1400N		Charleston	IL	61920-8472	17508 E County Road 1400N			Charleston	IL	61920-8472	(217) 348-6881		COLES
7 Twp and Westmoreland Fire Dept	510 Campbell ST	Jim Smith	Westmoreland	KS	66549	510 Campbell Jim Smith			Westmoreland	KS	66549	(785) 457-3736		POTTAWATTAMIE
79 East Volunteer Fire Department	5624 N US 79		Palestine	TX	75801	5624 N US 79			Palestine	TX	75801	903-729-5645		ANDERSON
7th Ward Volunteer Fire Department	20206 N LA Highway 82		Abbeville	LA	70510	20206 N LA Highway 82			Abbeville	LA	70510-0373	337-893-802	337-893-498	VERMILION
812 Volunteer Fire Department/EMS	1525 FM 812	PO Box 448	Cedar Creek	TX	78612	1525 FM 812			Cedar Creek	TX	78612	(512) 284-07	(512) 601-17	BASTROP
84 East Volunteer Fire Department	160 ACR 385		Palestine	TX	75801	160 ACR 385			Palestine	TX	75801	908-729-6627		ANDERSON
9 Mile Ridge Volunteer Fire Departme	6735 9 Mile Ridge RD		Hardy	AR	72542			PO Box 1165	Hardy	AR	72542-1165	000-000-0000		FULTON
90 CES/CEF F.E. Warren AFB	6205 10th Cavalry AV F.E. Warren AFB		Cheyenne	WY	82005	6205 10th C.F.E. Warren AFB			Cheyenne	WY	82005	307-773-293	307-773-446	LARAMIE
A & A Township Volunteer Fire Depart	6494 State Road 42		Eminence	IN	46125			PO Box 225	Eminence	IN	46125-0225	765-528-243	765-528-216	MORGAN

Follow these steps to create a content type:

1. To create the content type you can go to Structure | Content Types and click on the **Add content type** link. This will load the **Add content type** form.
2. Give your content type a name. In this example we'll call our content type **Fire Department**.

3. Type a description for the content type. For this example let's use this description: The Fire Department content type contains data and images about each fire department in the U.S. gathered from the USFA National Fire Department Census.
4. Change the **Title** field label to Fire Department Name. You can always leave this label set to the default **Title** but I like to rename my **Title** field labels to be specific for each content type that I create.
5. You can leave all of the other type settings including the **Submission form settings**, **Publishing options**, **Display settings**, **Comment settings**, and **Menu settings** set to their defaults for now.
6. Click on the **Save** content type button.

Once you save, your new content type will be created and you'll see it appear in the Content types table. You'll also see that your content type has an automatically generated machine name, in our case `fire_department`.

This is what you should see in your main content types' administration screen:

Fire Department (Machine name: fire_department)				
The Fire Department content type contains data and images about each fire department in the U.S. gathered from the USFA National Fire Department Census.	edit	manage fields	manage display	delete

Planning for the content type fields

So now that we have created our content type we are ready to add custom fields to the content type that will hold the data from our CSV file. Before we start adding the fields let's take a quick glance over our CSV data to see what types of fields and data we need to collect. We're going to illustrate this in a table for reference purposes and this will help us when we start adding our fields. On the left is the name of the CSV column and on the right is the type of the field we need to create in Drupal:

Name of the CSV column	Type of the field
fire_dept_name	Title (This field has already been created in the previous section as the default Title field in our content type. We will import the <code>fire_dept_name</code> column into the Title field.)
hq_addr1, hq_addr2, hq_city, hq_state, hq_zip, mail_addr1, mail_addr2, mail_po_box, mail_city, mail_state, and mail_zip	Location and Text fields. (We'll import some of the address data into fields provided by the Location module.)
hq_phone and hq_fax	Location fields

Name of the CSV column	Type of the field
county and dept_type	Term reference
organization_type	Node reference
website	Link
number_of_stations, active_firefighters_career, active_firefighters_volunteer, active_firefighters_paid_per_call, nonfirefighting_civilian, nonfirefighting_volunteer, and primary_agency	Integer

As you can see here we're going to be importing content into various types of Drupal fields. Some of the fields will be handled by contributed modules that we will install, including **Node Reference**, **Location**, and **Link**. The term reference fields will be clickable links to the corresponding tag in Drupal once we run the import. We'll import those tags into a vocabulary in the Drupal taxonomy first. The Node reference field similarly will be a link to a corresponding organization type node on the site.

We'll also be adding other types of fields to hold additional content that we will not necessarily be importing but that we'd like to include in our content type for the future. This shows you the power and flexibility of moving content to Drupal. You can move existing content and also build in new fields to hold new content that you'll add in the future development of the site.

We also need to add a column to our CSV to hold a unique identifier. When we import the CSV content, each Drupal node created will have a unique node ID or **nid** assigned to it by Drupal. But we also want to add a GUID column to our CSV, so that when we run the importer we're mapping the Feeds Importer GUID to the CSV GUID. This will give a unique ID to all elements that we import. If you decide to run an update or replace your data, each node will get the correct data from the CSV by referencing the unique identifier.

Ultimately this unique identifier will help prevent duplicate data and corrupted imports in the future as we maintain our import process. In your CSV file go ahead and add a new column and give the column a header name of id. The **id** column will map to the GUID when you create your feed importer. You will not need to add a custom field to your content type for the ID data as this data is automatically imported into the feeds_item table in your database.

Now you'll need to add sequential numbers to the id column in your CSV so each row has a unique ID. You can do this easily in Excel or another spreadsheet program.

Additional contributed modules needed

We're going to use three additional contributed modules, **Location**, **Location Feeds**, and **Link**, which we did not mention earlier in *Chapter 1, Preparing Drupal for Content Migration*. The **Location** module will enable us to add a location-based field group to our content type that we can use to import address data into. The **Location Feeds** module will enable us to map to the Location fields with our Feeds importer. The **Link** module we'll use to import the Fire Department's URL. Let's install these modules.

The Location module

The **Location** module is available for download from its project page at <http://drupal.org/project/location>. The latest version is 7.x-3.0-alpha1. Go ahead and install this module like any contributed Drupal module. Once installed you can enable the following modules on your modules administration screen. Make sure to leave the **Location Add Another** and the **Node Locations** modules disabled. You must also enable the **Location CCK** module under the CCK group. Be aware that your location fields will not save data correctly if you enable the **Node Locations** module. Use **Location CCK** for our examples instead of the **Node Locations** module.

- **Location**
- **Location Fax**
- **Location Phone**
- **Location Search**
- **Location Taxonomy**
- **User Locations**
- **Location CCK**



Downloading the example code

You can download the example code files for all Packt books you have purchased from your account at <http://www.packtpub.com>. If you purchased this book elsewhere, you can visit <http://www.packtpub.com/support> and register to have the files e-mailed directly to you.

You should see the following on your modules admin screen once enabled:

LOCATION				
ENABLED	NAME	VERSION	DESCRIPTION	OPERATIONS
<input checked="" type="checkbox"/>	Location	7.x-3.0-alpha1	The location module allows you to associate a geographic location with content and users. Users can do proximity searches by postal code. This is useful for organizing communities that have a geographic presence. Required by: Node Locations (disabled), Location Add Another (disabled), Location CCK (enabled), Location Fax (enabled), Location Feeds (enabled), Location Generate (disabled), Location Phone (enabled), Location Search (enabled), Location Taxonomy (enabled), User Locations (enabled)	Help Permissions Configure
<input type="checkbox"/>	Location Add Another	7.x-3.0-alpha1	Allows you to quickly add locations directly from a node without having to click 'edit' first. Requires: Location (enabled), Node Locations (disabled)	
<input checked="" type="checkbox"/>	Location Fax	7.x-3.0-alpha1	Allows you to add a fax number to a location. Requires: Location (enabled)	
<input checked="" type="checkbox"/>	Location Phone	7.x-3.0-alpha1	Allows you to add a phone number to a location. Requires: Location (enabled)	
<input checked="" type="checkbox"/>	Location Search	7.x-3.0-alpha1	Advanced search page for locations. Requires: Search (enabled), Location (enabled)	
<input checked="" type="checkbox"/>	Location Taxonomy	7.x-3.0-alpha1	Associate locations with taxonomy terms. Requires: Location (enabled), Taxonomy (enabled), Options (enabled), Field (enabled), Field SQL storage (enabled)	
<input type="checkbox"/>	Node Locations	7.x-3.0-alpha1	Associate locations with nodes. Requires: Location (enabled) Required by: Location Add Another (disabled)	
<input checked="" type="checkbox"/>	User Locations	7.x-3.0-alpha1	Associate locations with users. Requires: Location (enabled)	Permissions Configure

The Location Feeds module

In order to import data into the Location fields we'll need to install the **Location Feeds** module. This module is available via its project page at http://drupal.org/project/location_feeds. The current version is 7.x-1.6. Install the module like any contributed module and then enable it on your modules administration screen. The module will be in the Feeds grouping of modules on your modules administration page.

The Link module

We need to install the **Link** module to handle importing any type of URL or website address into a field on our site. The **Link** module is available from its project page at <http://drupal.org/project/link>. Install the module as you would any contributed module and enable it on your modules administration page. The current module version is 7.x-1.0.

Adding fields to the content type

We're now ready to add custom fields to our content type using the migration plan we will create later. Go back to your Fire Department content type and click on the Manage Fields tab. This will open up the field's configuration screen.

The Location and text fields

Let's go ahead and add a group of Location fields to our content type that we'll use to map the imported address data from our CSV file. I'm going to advise adding all fields of one type in order of their appearance in the CSV. Your migration plan will help you to do this. You can easily rearrange your fields once you've created them all if you want the order to be exact to your CSV's order.

Bear in mind that the `fire_department_name` column will map to the content type title field, so you do not need to add a custom field for that column.

The location fields will be inserted using the **Location** module that we added to our site earlier. Now that we've enabled that module we can add a Location field to our content type. The Location field is actually made up of an entire set of fields that will hold our address information and data.

Adding a Location field

Let's go ahead and add our location field. To do this, add a new field called `Fire Department HQ Address` and choose `Location` as the type of field. The widget will default to the Location field. Click on the **Save** button.

The field settings screen will be loaded. The Location field is made up of multiple fields that you can collect data for, and you can also set display settings. On the screen you should see both collection settings and display settings for the following fields:

- **Location name**
- **Street Location**
- **Additional**
- **City**
- **State/Province**
- **Postal Code**
- **Country**
- **Coordinate Chooser**
- **Phone Number**
- **Fax number**

Starting a Migration Path

We will be collecting all of this data so make sure to set all collection settings to **Allow**.

We will also be displaying all of our location settings. You should see the following screen at this point:

The screenshot shows the 'FIELD SETTINGS' interface for the 'Fire Department HQ Address' field. It includes a 'LOCATIVE INFORMATION' section with 'COLLECTION SETTINGS' and 'DISPLAY SETTINGS'.

COLLECTION SETTINGS

NAME	COLLECT	WIDGET	DEFAULT
Location name	Allow		<input type="text"/> e.g. a place of business, venue, meeting point
Street location	Allow		<input type="text"/>
Additional	Allow		<input type="text"/>
City	Allow		<input type="text"/>
State/Province	Allow	Autocomplete	<input type="text"/>
Postal code	Allow		<input type="text"/>
Country	Allow		United States
Coordinate Chooser	Allow		
Phone number	Allow		<input type="text"/>
Fax number	Allow		<input type="text"/>

DISPLAY SETTINGS

Hide fields from display

- ☐ Location name
- ☐ Street location
- ☐ Additional
- ☐ City
- ☐ State/Province
- ☐ Postal code
- ☐ Country
- ☐ Coordinate Chooser
- ☐ Fax number

Click on the **Save settings** button. The next screen will give you a preview of what the locative form will look like. You can set default values for the fields here if you need to.

Click on **Save settings** again.

If you now go to add a new Fire Department node to your site you should see the group of Fire Department HQ Address location fields on the content type form. Notice that the **State/Province** field is an autocomplete. If you start typing in the state, it will auto complete. You should see something similar to the following on your content type form:

FIRE DEPARTMENT HQ ADDRESS

Location name
e.g. a place of business, venue, meeting point

Street

Additional

City

State/Province

Postal code

Country

Latitude

Longitude

If you wish to supply your own latitude and longitude, you may enter them above your location from the address, for example if you change the address, delete the

Phone number

Fax number

Also notice that the group contains Latitude and Longitude fields. You can enter this data if you have it for a Fire Department and then if you utilize mapping modules and mapping integration later in your Drupal site you can do some interesting mapped displays of your data.

For our import example I'm interested in capturing the Fire Department Headquarters data via these location fields so I can map the headquarters. We also have mailing address data in our CSV. I'm going to add text fields for the following data:

- **Mailing Address 1**
- **Mailing Address 2**
- **Mail PO Box**

- **Mail City**
- **Mail State**
- **Mail Zip**

This will give you some practice in adding text fields to your site. Let's add one now.

I'm going to demonstrate adding one text field and then you should add the remaining text fields based on our map or your CSV to practice. To add your text field for Mailing Address 1, do the following:

1. Give your field a label name. This name should be human readable; so, we'll enter `Mailing Address 1`. When you do this, you will notice that Drupal will automatically add a machine name in the machine name column of our content type screen.
2. If you are planning on having multiple custom content types on your site, it's a good practice to prepend the machine name of the field with the name of your type, in our case with `fire_`. So, for example, we can add `fire_` to the machine name on the mailing address field and our machine name will be `field_fire_mail_addr1`. This helps organize your fields and makes it easier to know what content type a field belongs to if you're looking in the database for the instance of this data.
3. Click the **Edit** link next to the machine name to do this. There's a character limit to this machine name so we'll shorten the whole thing to `fire_mail_addr_1`. You don't need to add `field_`. Drupal will add `field_` to the field name.
4. In the **Select a field type** drop-down menu, select **Text**.
5. Click on the **Save** button. This will launch the next field settings screen.
6. Set a maximum length on the **Field Settings** screen. Leave it at the default of 255 if that is sufficient. Click on the **Save settings** button.
7. On the next screen, you can add Help text, set the size of the text field shown when editing content, add text processing, a default value, and set whether the field should allow for multiple values. We can safely leave all these set to their default for this field.

8. Save your settings. You'll be returned to the content type where you'll see your new field.
9. Add the remaining text fields to the content type based on your migration map and CSV.

Integer fields

Now let's add our integer fields to the content type. We're going to add integer fields per our content migration plan. Let's demonstrate one of these for the `number_of_stations` content. As you did for the text fields, go to the **Manage Fields** screen of your content type:

1. Add a field with a Number of Stations label.
2. Edit the machine name and prepend your field name with `fire_` so that your entire field machine readable name is `fire_num_stations`.
3. Select a field type of **Integer**.
4. Click on the **Save** button.
5. On the field settings screen click on **Save field** settings.
6. With integer fields you can add a minimum and/or maximum value as a control. This is helpful if you want to force the user to enter specific values.
7. You can also add a prefix and/or suffix.
8. You can add a default value, and also configure whether the user can add multiple values for this field.

9. You should have an integer field with settings that look similar to the following:

Label *

☐ Required field

Help text

Instructions to present to the user below this field on the editing form.
Allowed HTML tags: <a> <big> <code> <i> <ins> <pre> <q> <small> <sub> <sup> <tt> <p>

Minimum

The minimum value that should be allowed in this field. Leave blank for no minimum.

Maximum

The maximum value that should be allowed in this field. Leave blank for no maximum.

Prefix

Define a string that should be prefixed to the value, like '\$ ' or '€ '. Leave blank for none. Separate singular and plural values with a pipe ('pound|pounds').

Suffix

Define a string that should be suffixed to the value, like ' m', ' kb/s'. Leave blank for none. Separate singular and plural values with a pipe ('pound|pounds').

DEFAULT VALUE
The default value for this field, used when creating new content.
Number of Stations

NUMBER OF STATIONS FIELD SETTINGS
These settings apply to the *Number of Stations* field everywhere it is used. Because the field already has data, some settings can no longer be changed.
Number of values

We'll leave the field settings at their defaults here and just click on the **Save settings** button. Add the remaining integer fields.

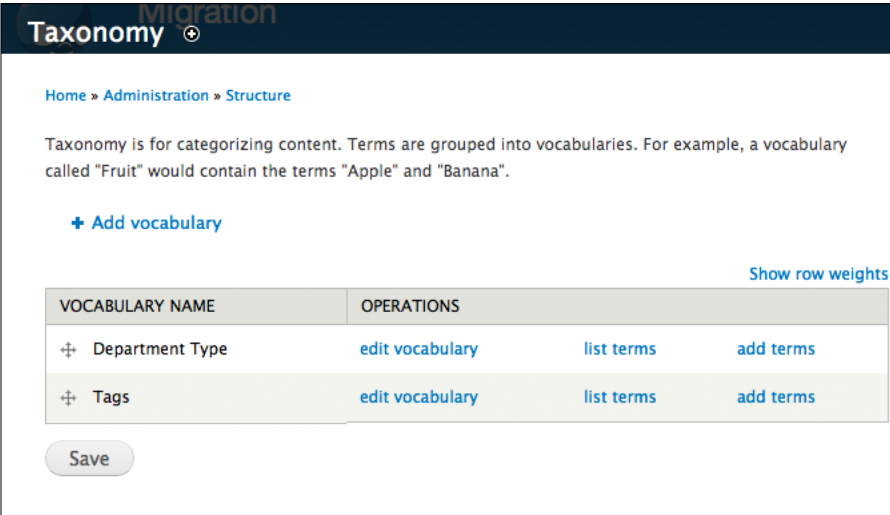
Term references

We determined in our migration map that some of our content would be best used on the site as a Drupal term. Drupal terms or tags allow the content (in this case the tag) to be linked to a page on the site that shows all other content that is tagged similarly. Tags can also be integrated easily with the **Views** module. To import our columns of terms we're going to add term reference fields to our content type. The term reference field comes as part of Drupal's core content type structure. The first thing we need to do is add a vocabulary to our taxonomy to contain the tags that we'll be importing.

Adding a vocabulary

Our column in the CSV file is called **dept_type** and we need to add the vocabulary container to hold the imported tags.

1. Go to **Structure | Taxonomy**.
2. Click on the **Add vocabulary** link.
3. Give the vocab a name as `Department Type`.
4. Edit the machine name and prepend `fire_`.
5. Click on the **Save** button. You now have a new vocabulary to hold your imported `dept_type` tags.



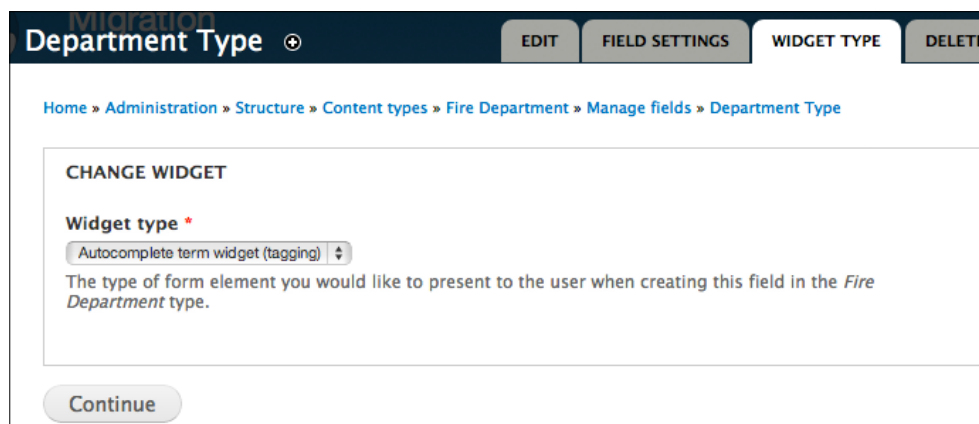
The screenshot shows the 'Taxonomy' administration page in Drupal. At the top, there's a breadcrumb trail: 'Home » Administration » Structure'. Below this, a brief explanation of taxonomy is provided. A blue '+ Add vocabulary' link is visible. Below the link is a table with two columns: 'VOCABULARY NAME' and 'OPERATIONS'. The table lists two vocabularies: 'Department Type' and 'Tags'. Each row has three links: 'edit vocabulary', 'list terms', and 'add terms'. A 'Save' button is located at the bottom left of the table area.

VOCABULARY NAME	OPERATIONS
+ Department Type	edit vocabulary list terms add terms
+ Tags	edit vocabulary list terms add terms

Adding the term reference field

Now let's add our corresponding term reference field to the content type. Use the same process to add a field as we did with the text and integer fields:

1. Add a new field labeled `Department Type`.
2. Prepend the field's machine name with `fire_` so that the field is called **`fire_department_type`**.
3. Select a field type of Term reference.
4. Select **Autocomplete term widget (tagging)** as the widget type.



5. Click on **Save**.
6. On the **Field settings** screen choose the vocabulary that you want to map to on the import. In our case it is `Department Type`.
7. Click on the **Save field settings**.
8. Leave the next screen of field settings at their defaults and click on **Save settings**. Save your content type.
9. The autocomplete widget will allow us to import comma or piped separated terms into the field. At this point you can add more vocabularies to your site to hold the other tags that you need to import, if you have other columns of this type of data in your CSV file.

Node references

Node references allow us to add a field that references and links to another node on the site. So this can be helpful to us if we want to relate or connect one node to another. When one node shows on the site, there will be a corresponding reference back to the node that was the referring node. In Drupal 6 (and still in Drupal 7) this is referred to as a Node reference. To add node reference fields to our content type we need to first install the References module.

Installing the References module

First we need to install the **References** module. Let's do that now.

1. Download the latest version of the **References** module from <http://drupal.org/project/references>.
2. Install the module as you would any Drupal contributed module.
3. Once installed load your modules admin screen and enable the **Node Reference**, **API**, **References**, and **User Reference** modules.

<input checked="" type="checkbox"/>	Node Reference	7.x-2.0+8-dev	Requires: Field (enabled), Field SQL storage (enabled), References (enabled), Options (enabled) Required by: Drupal (Field type(s) in use - see Field list), Fire Department Content Type (enabled)
<input checked="" type="checkbox"/>	References	7.x-2.0+8-dev	Defines common base features for the various reference field types. Requires: Field (enabled), Field SQL storage (enabled), Options (enabled) Required by: Node Reference (enabled), Fire Department Content Type (enabled), User Reference (enabled)
<input checked="" type="checkbox"/>	User Reference	7.x-2.0+8-dev	Defines a field type for referencing a user from a node. Requires: Field (enabled), Field SQL storage (enabled), References (enabled), Options (enabled)

Save your module configuration.

Adding the node reference field

Before adding our node reference field we should first create another content type that is going to hold our imported content that we will be referencing to. This additional content type will contain nodes for each piece of referenced content. Then our node reference field will map the imported data to those nodes on our site. This is very similar to how the term references work.

1. Add a content type to hold your node reference content. Name it Organization Type since that's the data we'll be importing. We do not need to give it any fields other than the default Title field for now. Save the content type.
2. Now click **Manage Fields** on your Fire Department content type.

3. Add a new field with a label of Organization Type.
4. Give the field a machine readable name of fire_organization_type.
5. Select a field type of **Node** Reference.
6. Choose the **Autocomplete text field** widget.
7. Save your node reference field.
8. On the Field settings screen select the **Organization Type** under the **Content types that can be referenced** section as we want to reference nodes of that type.
9. You should see a screen similar to the following:

The screenshot shows the 'Autocomplete matching' section with a dropdown set to 'Contains'. Below it is a 'Size of textfield' input set to 60. The 'DEFAULT VALUE' section shows 'Organization Type' as the default value. The 'ORGANIZATION TYPE FIELD SETTINGS' section includes a 'Number of values' dropdown set to 1. The 'Content types that can be referenced' section has a list of checkboxes, with 'Organization Type' checked and others unchecked.

Autocomplete matching
 Select the method used to collect autocomplete suggestions. Note that *Contains* can cause performance issues on sites with thousands of nodes.

Size of textfield *

DEFAULT VALUE
The default value for this field, used when creating new content.
Organization Type

ORGANIZATION TYPE FIELD SETTINGS
These settings apply to the *Organization Type* field everywhere it is used. Because the field already has data, some settings can no longer be changed.

Number of values
 Maximum number of values users can enter for this field. 'Unlimited' will provide an 'Add more' button so the users can add as many values as they like.

Content types that can be referenced
☐ Article
☐ Basic page
☐ Beer
☐ Blog entry
☐ Feed
☐ Feed item
☐ Fire Department
☒ Organization Type
☐ Panel
☐ Wine
☐ Wine Producer

10. Leave the rest of the settings at their defaults and click the **Save settings** button.

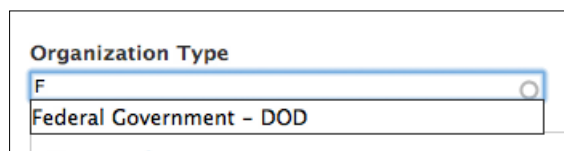
Testing our content types

Let's now test both the Fire Department and the Organization Type content types and confirm that we can add nodes to our site using both types. As we already tested our location fields earlier in this chapter, let's do specific tests of both our node reference field and term reference fields.

Testing the node reference field

We can easily test our node reference field by adding a node to our site using the new Organization Type content type. If we look at our CSV file we can see that one of the nodes will be called Federal Government – DOD. Let's use that one for our first organization type node. To add the node:

1. Go to **Content | Add Content**.
2. Click on the **Organization Type** link.
3. Add the **Federal Government – DOD** value as the title.
4. Click on **Save** to save your new node to the site.
5. Now go to **Content | Add Content** and click on the **Fire Department** link.
6. Add a node for testing purposes and make sure to enter the **Federal Government – DOD** into the autocomplete field for your node reference. You'll see that it's autocomplete so Drupal will suggest the new node you added above:



The screenshot shows a web form titled "Organization Type". Below the title is a text input field containing the letter "F". A dropdown menu is open below the input field, displaying a list of suggestions. The first suggestion, "Federal Government - DOD", is highlighted in blue. Below the dropdown, there is a link that says "More settings".

7. Save your **Fire Department** node.
8. On the resulting node view page you'll see the result. Your organization type will be shown. The following is the resulting value of the reference back to the **Federal Government – DOD** node you created earlier:

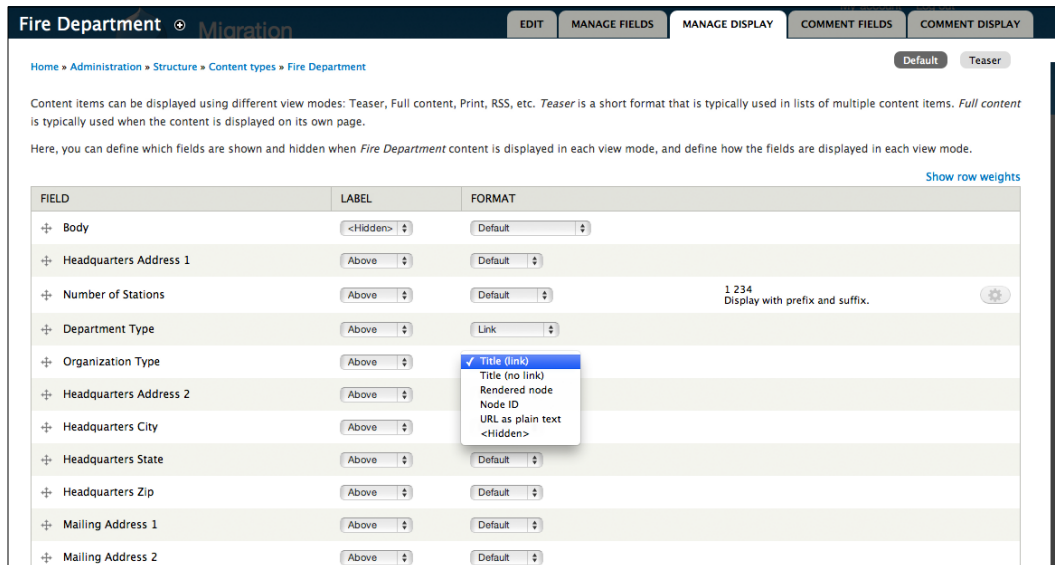


The screenshot shows a box containing the text "Organization Type:" followed by a blue hyperlink that reads "Federal Government - DOD".

Changing the display formatter on the node reference field

We can go even further to test out our new node reference field. By default the node reference linkage by field will get a link display but we can change the display format of the field at the **Fire Department** content type level.

1. Load your content type configuration screen and click on the **Manage Display** tab. This will load a display screen that looks like the following:



Fire Department Migration

EDIT MANAGE FIELDS MANAGE DISPLAY COMMENT FIELDS COMMENT DISPLAY

Home » Administration » Structure » Content types » Fire Department

Default Teaser

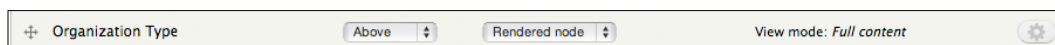
Content items can be displayed using different view modes: Teaser, Full content, Print, RSS, etc. *Teaser* is a short format that is typically used in lists of multiple content items. *Full content* is typically used when the content is displayed on its own page.

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

Show row weights

FIELD	LABEL	FORMAT
+ Body	<Hidden>	Default
+ Headquarters Address 1	Above	Default
+ Number of Stations	Above	Default 1,234 Display with prefix and suffix.
+ Department Type	Above	Link
+ Organization Type	Above	Title (link)
+ Headquarters Address 2	Above	Title (no link)
+ Headquarters City	Above	Rendered node
+ Headquarters State	Above	Node ID
+ Headquarters Zip	Above	URL as plain text
+ Mailing Address 1	Above	<Hidden>
+ Mailing Address 2	Above	<Hidden>

2. Now click on the **Title (link)** formatter to tweak its display settings.
3. If you click on the **Title (link)** box you will see other options load. Check the box next to **Link** label to the referenced entity including **Title (no link)**; **Rendered node**, **Node ID**, and **URL** as plain text.



+ Organization Type

Above

Rendered node

View mode: Full content

4. To change the format, select one of the other display formatters. Let us select the **Rendered Node** for example purposes. When you select **Rendered node** the default **View** mode that will show will be **View mode: Full content**.
5. Click on the **Update Save** button and the formatter will change. Now save your content type's display.

Now go back to your content listing and view the Fire Department node you created. You'll now notice that the Organization type node content is showing up on your fire department node. In this case it still appears as a link as all the Organization type node contains is the title of the specific entity that you are referencing.

131st Fighter Wing Fire Protection

[View](#)[Edit](#)[Log](#)

Submitted by Anonymous (not verified) on Sun, 08/19/2012 - 14:42

Headquarters Address 1:
10800 Lambert International BLVD

Number of Stations:
1

Department Type:
[Mostly Volunteer](#)

Organization Type:
[Federal Government - DOD](#)

Submitted by [admin](#) on Mon, 08/06/2012 - 16:25

You can embed the referenced node, or simply link to it using either of these two formatter methods. Let's change the display back to **Title (link)** as that's the formatter we'll use going forward.

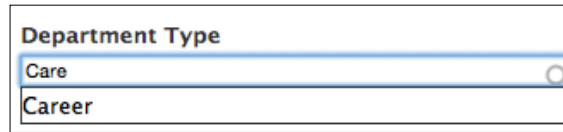
We'll also explore doing this linkage with the **Views** module later in the book.

Testing the term reference field

We can also test the term reference field we added to make sure it creates a referenced linkage using a similar process to the one we used with the entity reference. First we need to add a tag to the **Department Type** vocab.

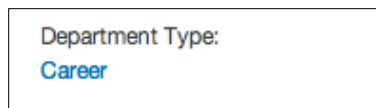
1. Go to your vocabulary screen and click on the add terms link next to Department Type.
2. Add a tag called Career.
3. Click on the **Save** button.

4. Now go to the Fire Department content you added previously and enter Career as the term reference field autocomplete value.



A screenshot of a web form with a label 'Department Type'. Below the label is a dropdown menu. The dropdown is open, showing two options: 'Care' and 'Career'. The 'Care' option is currently selected and highlighted with a blue border.

5. Save the node.
6. Now when you view the node you should see a linked **Department Type**. If you click on the **Career** link it will take you to that tag's corresponding tag display page.



A screenshot of a node view. It shows the text 'Department Type:' followed by a blue, clickable link labeled 'Career'.

7. Click on the **Career** tag to see its tag display page.



A screenshot of the 'Career' tag display page. The page has a title 'Career' at the top. Below the title are two buttons: 'View' and 'Edit'. The main content area displays the text 'Aberdeen Proving Ground Fire & Emergency Services' in a bold font. Below this, it says 'Submitted by admin on Mon, 08/06/2012 - 16:30'. At the bottom left of the page, there is an RSS feed icon.

Content type summary

We have successfully added all of our content type fields that we will need for the importer and migration process. When you view your content type's fields you should see a screen similar to the following:

LABEL	MACHINE NAME	FIELD TYPE	WIDGET
+ Fire Department Name	title	Node module element	
+ Body	body	Long text and summary	Text area with a summary
+ Fire Department HQ Address	field_fire_department_hq_address	Location	Location Field
+ Number of Stations	field_fire_number_of_stations	Integer	Text field
+ Department Type	field_fire_department_type	Term reference	Autocomplete term widget (tagging)
+ Organization Type	field_fire_organization_type	Node reference	Autocomplete text field
+ Mailing Address 1	field_fire_mail_addr1	Text	Text field
+ Mailing Address 2	field_fire_mail_addr2	Text	Text field
+ Mail PO Box	field_fire_mail_po_box	Text	Text field
+ Mail City	field_fire_mail_city	Text	Text field
+ Mail State	field_fire_mail_state	Text	Text field
+ Mail Zip	field_fire_mail_zip	Text	Text field
+ County	field_fire_county	Term reference	Autocomplete term widget (tagging)
+ Website	field_fire_website	Link	Link
+ Active Firefighters Career	field_fire_active_c	Integer	Text field
+ Active Firefighters Volunteer	field_fire_active_volunt	Integer	Text field
+ Active Firefighters Paid Per Call	field_fire_active_fire_paid_p	Integer	Text field
+ Non-Firefighting Civilian	field_fire_non_fire_civilian	Integer	Text field
+ Non-Firefighting Volunteer	field_fire_non_fire_volunteer	Integer	Text field
+ Primary Agency for Employment	field_fire_primary_agency	Integer	Text field
+ Fire Department Image	field_fire_department_image	Image	Image

Now that we have built our content types and fields we're ready to move on to the import process. Before we begin this discussion, let's briefly take a look at how you can migrate content having images including JPEG, GIF, and PNG, and files including PDF, DOC, and other multimedia files.

Migrating images and files

Let's talk briefly about the best methods for migrating images and files to your new site. If you have a lot of images and/or files in your existing web content it's easy to migrate these using the following guidelines. You can do the following to handle migration of images when the images are embedded in your web page with HTML:

1. In your **Fire Department** content type, you have a **Body** field that comes as part of the core content type. This is a large text area field that you can import your main page content into, especially if that page content contains hyperlinks, images, or references to other files as HTML tags. This will work even more smoothly if the HTML contains absolute links to the image content.

2. When we build the importer in the next chapter we'll specify that we want to import this body content with a full HTML format, so that we can preserve the HTML as it was written in your old website.
3. So for now just confirm that your **Body** field has **Filtered Text** as its text format.
4. Now generally on a website, images, or files tend to be stored or uploaded to a specific directory. You should confirm what directory your current site stores these files in and note the path to the directory.
5. Now you can rebuild that folder or directory on your new Drupal server so that the paths will remain the same to your files.
6. The last thing you'll need to do is copy all your images and files from your old site to the new folder on your new server. Now all the files and images will be exactly at the same path and URL and so your imported code will not break or cause any issues and the images and files will match and map correctly.

This method is not fool proof. You may still find that you have to manually migrate over some images and files and re-upload them into a new file field or image field that you add to your Drupal content type. One of the benefits to migrating your images is that it will be easier to manage them through the Drupal user interface.

Summary

In this chapter we successfully built our content types for our import process and added custom fields to the types to collect the migrated content. We added text fields, integer fields, term references, and node references to our content type. We also discussed about migrating images and files using a streamlined approach. In the next chapter we will begin working on constructing our import process.

3

Creating a Feeds Importer

We have built our content type and added custom fields to hold our imported content and data. We're now ready to build our importer that will allow us to import our content from a CSV file into our content type. At this point we'll assume that you have finished adding all of your custom fields to your content type that you'll need based on your migration plan and on your CSV file. We'll also assume that you have named your CSV column headers so that they are all lower case and words are separated by underscores.

Additionally I have added a column to my CSV that will hold the GUID or unique identifier numeric ID for each content row, and subsequently each feed item that gets imported into a Drupal node. In Excel, I simply added a GUID column to my CSV file and then filled that column with sequential numbers, in this case from 1-999 as I'll be importing 999 rows of data.

We'll cover the following topics in this chapter:

- Creating a feeds importer
- Exporting, cloning, and deleting feeds importers
- Importer basic settings
- Importer fetcher
- Parsers
- Processors
- Mapping your importer
- Adding a source and target and using GUID for mapping
- Running the import process and testing it

Creating a feeds importer

Let's jump right into creating a feeds importer to handle our import and migration. To do this you will need to go to **Structure | Feeds importers**. This will launch the main importer screen showing the default importers that the module gives us:

- **Node import:** This is used to import data into nodes
- **User import:** This is used to import user accounts from CSV files
- **Feed:** This is used to import data from RSS or Atom feeds
- **OPML import:** This is used to import from OPML files

We're going to use the Node import mechanism as we want to import our data into nodes on our site. Next to each importer is a table showing what the importer is attached to, the status, various linked operations, and whether the importer is enabled.

Exporting, cloning, and deleting importers

Rather than editing the default node importer the **Feeds** module provides it is better practice to use the Clone operation to clone an existing importer. This way you are not manipulating the original importer and you always have the default node importer in its original state. There are other operations that you can do to an importer including exporting it, deleting it, and adding a tamper to it. We'll come back to the process for adding a tamper later in *Chapter 4, Feeds Tamper*.

We're going to clone the node import. So click on the **Clone** link. The screen that appears next will give you a form where you can add a name and description for your importer. Let's call it the `Fire Department Importer` and give it a simple description. You should now be directed to a screen showing you the settings for your new Fire Department importer. Before we continue with this, go back to the main **Structure | Feeds importers** screen and you should now see something similar to the following:

[+ Add importer](#)

NAME	DESCRIPTION	ATTACHED TO	STATUS	OPERATIONS	ENABLED
Fire Department Importer	This is used to import firehouse content from CSV into nodes.	[none]	Normal	Edit Export Clone Delete Tamper	<input checked="" type="checkbox"/>
Node import	Import nodes from CSV file.	[none]	Default	Override Export Clone Tamper	<input checked="" type="checkbox"/>
User import	Import users from CSV file.	[none]	Default	Override Export Clone Tamper	<input checked="" type="checkbox"/>
Feed	Import RSS or Atom feeds, create nodes from feed items.	Feed	Default	Override Export Clone Tamper	<input checked="" type="checkbox"/>
OPML import	Import subscriptions from OPML files. Use together with "Feed" configuration.	[none]	Default	Override Export Clone Tamper	<input checked="" type="checkbox"/>

[Save](#)

Importer configuration

Now that we have cloned and created our Fire Department importer, let's start configuring it. Make sure to have your CSV file ready as we'll be referring to it and using it during this process.

Basic settings

Make sure you have clicked on the **Edit** link to edit your Fire Department importer. The whole importer configuration is stacked in a tabbed menu in the left of the screen. We're going to look at the basic settings first.

Click on the **Settings** link to open up the basic settings form. The form on the right shows the importer name and description you gave it in the last section. You can attach the importer to a specific content type if you want or you can use the standalone import form. If you have a site where you are running a lot of imports and you want to keep things organized and consistent it may be helpful to attach an importer to a content type that's used solely for running the import. However in our case we're going to use the standalone form as it provides a simpler process and mechanism for our use case.

Periodic import allows you to set up imports to run automatically based on a cron task you've configured for your site. We'll leave this set to **Off** for now.

Leave the **Import on submission** checkbox checked as we want to import our content as soon as we run the import mechanism.

Process in background can be helpful if you have a large import file with thousands of records and you want to control the import via a cron task, to allow for importing a number of records at a time in batches. So for example you could set **Periodic import** to **Every 3 hours**. Then you can configure your Drupal cron task to run every 3 hours at `/admin/config/system/cron`, or by going to **Configuration | System | Cron**. When cron runs, your import will also run automatically.

You should now see the following screen:

The screenshot shows the 'Basic settings' configuration page for a Feeds Importer. The page is divided into two main sections: a left sidebar with navigation links and a main content area. The sidebar includes sections for 'Basic settings', 'Fetcher', 'Parser', and 'Processor', each with a 'Settings' link. The main content area is titled 'Basic settings' and contains several configuration options. The 'Name' field is set to 'Fire Department Importer'. The 'Description' field contains the text 'This is used to import firehouse content from CSV into nodes.' The 'Attach to content type' dropdown is set to 'Use standalone form'. The 'Periodic import' dropdown is set to 'Off'. The 'Import on submission' checkbox is checked, and the 'Process in background' checkbox is unchecked. A 'Save' button is located at the bottom of the page.

Basic settings	
Attached to: [none]	Settings
Periodic import: off	
Import on submission	
Fetcher Change	
File upload	Settings
Upload content from a local file.	
Parser Change	
CSV parser	Settings
Parse data in Comma Separated Value format.	
Processor Change	
Node processor	Settings
Create and update nodes.	Mapping

Basic settings [Help](#)

Name *

A human readable name of this importer.

Description

A description of this importer.

Attach to content type
 [+](#)
If "Use standalone form" is selected a source is imported by using a form under <http://migration.localhost/import>. If a content type is selected a source is imported by creating a node of that content type.

Periodic import
 [+](#)
Choose how often a source should be imported periodically. [Requires cron to be configured.](#)

☒ **Import on submission**
Check if import should be started at the moment a standalone form or node form is submitted.

☐ **Process in background**
For very large imports. If checked, import and delete tasks started from the web UI will be handled by a cron task in the background rather than by the browser. This does not affect periodic imports, they are handled by a cron task in any case. [Requires cron to be configured.](#)

We've completed our basic settings configuration. Let's move on to the **Fetcher** tab.

Fetcher

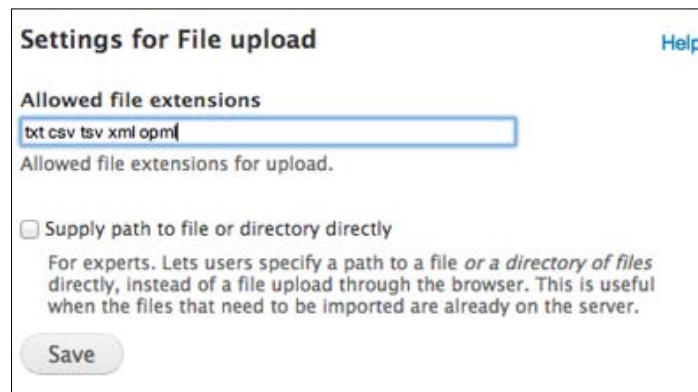
For Fetcher settings, follow these steps:

1. To access the Fetcher settings click on the **Change** link. The following screen will display. You want to leave this set to the default **File upload** configuration, as we will be uploading a CSV file for the import.



The screenshot shows a dialog box titled "Select a fetcher" with a "Help" link in the top right corner. There are two options: "File upload" with a description "Upload content from a local file." and a radio button labeled "Select" (which is selected), and "HTTP Fetcher" with a description "Download content from a URL." and a radio button labeled "Select" (which is not selected). A "Save" button is at the bottom.

2. Now click on the **Settings** link next to **File upload**. This will show a form that allows you to define the extensions (or types) of files you will want to upload during the import process. Confirm that `csv` is one of the file extensions. You should see the following:

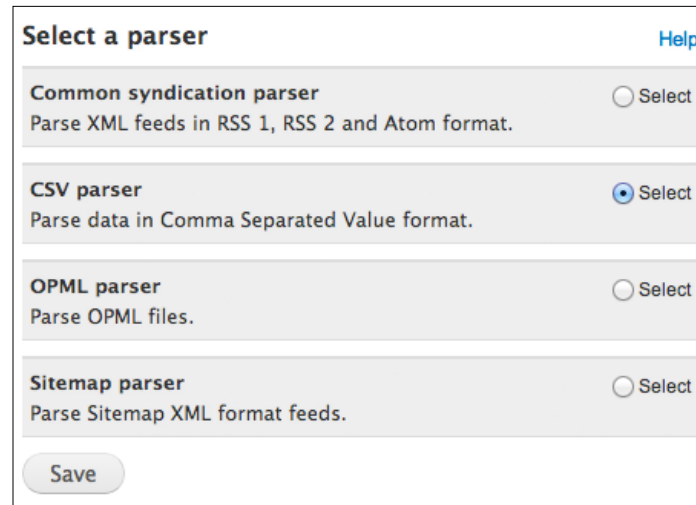


The screenshot shows a form titled "Settings for File upload" with a "Help" link in the top right corner. It has a section "Allowed file extensions" with a text input field containing "txt csv tsv xml opml". Below the input field is the text "Allowed file extensions for upload." There is a checkbox labeled "Supply path to file or directory directly" which is unchecked. Below the checkbox is a paragraph: "For experts. Lets users specify a path to a file or a directory of files directly, instead of a file upload through the browser. This is useful when the files that need to be imported are already on the server." A "Save" button is at the bottom.

Parser

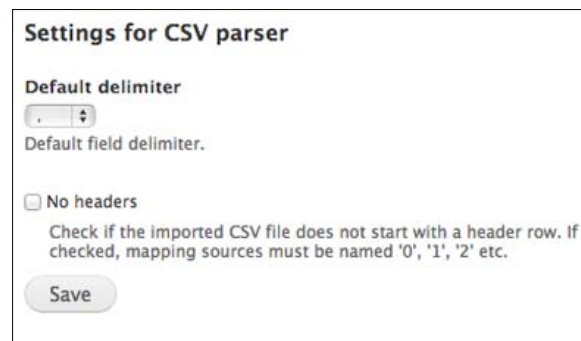
For the Parser settings, follow these steps:

1. Click on the **Change** link next to the **Parser** tab. This form allows you to define the parser that you want to use for the import. We're going to leave the default **CSV parser** selected. You should see the following screen:



The screenshot shows a form titled "Select a parser" with a "Help" link in the top right corner. It contains four radio button options, each with a description: "Common syndication parser" (Parse XML feeds in RSS 1, RSS 2 and Atom format.), "CSV parser" (Parse data in Comma Separated Value format.), "OPML parser" (Parse OPML files.), and "Sitemap parser" (Parse Sitemap XML format feeds.). The "CSV parser" option is selected. A "Save" button is at the bottom left.

2. Now click on the **Settings** link next to the **CSV parser** tab. This form allows you to tweak the delimiter for your CSV import. We will leave this set to the default comma separated delimiter but you could change this to either a semicolon or a tab. You'll also be able to override this delimiter later using the **Feeds Tamper** module. As we are importing a CSV that contains headers, leave the **No headers** checkbox unchecked. You should see the following screen:



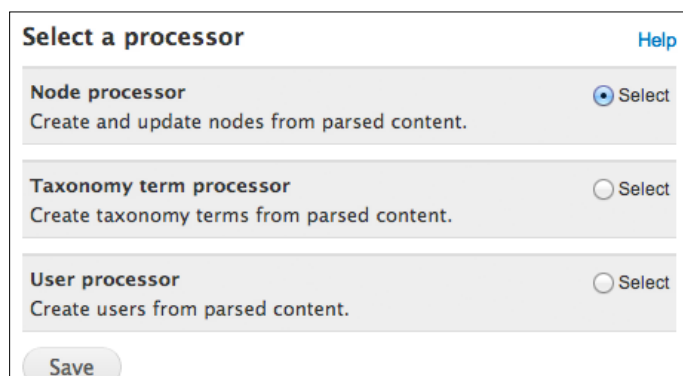
The screenshot shows a form titled "Settings for CSV parser". It has a "Default delimiter" dropdown menu set to a comma, with a description "Default field delimiter." Below it is a "No headers" checkbox, which is unchecked, with a description "Check if the imported CSV file does not start with a header row. If checked, mapping sources must be named '0', '1', '2' etc." A "Save" button is at the bottom.

3. If you do make changes make sure to click on the **Save** button.

Processor

For the Processor settings, follow these steps:

1. Click on the **Change** link next to the **Processor** tab. You'll see a processor form load with the default **Node processor** selected. As we're going to be importing into nodes on our site, we can leave this selected. You should see the following screen:



Select a processor [Help](#)

Node processor ☒ Select
Create and update nodes from parsed content.

Taxonomy term processor ☐ Select
Create taxonomy terms from parsed content.

User processor ☐ Select
Create users from parsed content.

[Save](#)

2. Now click on the **Settings** link next to the **Node processor** tab. This will launch a form that allows you to tweak the specific node processor configuration.
3. On the **Settings for Node processor** configuration screen you have to choose either of the three options – **Update existing nodes**, **Replace existing nodes**, or **Do not update existing nodes** (leave existing nodes untouched). This is required as you are importing a GUID or unique identifier for each node. So when you re-run an import process with the updated or new content, you can update or replace the nodes that you've already imported. By default your Node processor will be set to **Replace existing nodes**. Let's change this to **Update existing nodes**.
4. The next settings determine the text format for the import. If you are importing a large amount of body text into the body field of your content type and if you have HTML code to import, then you should make sure that it is set to **Full HTML** here.
5. Next, we want to choose the content type for the import process. Nodes of the selected type will be created during the import process, so choose the content type you built and added custom fields to earlier. In this case it is **Fire Department**.

6. The **Author** field is set by default to **anonymous** but you can change this to be another user account on the site, if you want all imported nodes to be associated with this user account.
7. **Expire nodes** allows you to delete imported nodes after a specific time period based on a node's publication date. This may be helpful for some sites if you want to delete content after a specified period of time.
8. You should see the following screen at this point:

The screenshot displays the Feeds Importer configuration interface. On the left is a sidebar with a list of settings categories: Basic settings, Fetcher, Parser, and Processor. The 'Node processor' category is currently selected. The main area on the right is titled 'Settings for Node processor' and contains several sections: 'Update existing nodes' with radio buttons for 'Do not update existing nodes', 'Replace existing nodes', and 'Update existing nodes (slower than replacing them)' (which is selected); 'Text format' with a dropdown menu set to 'Full HTML'; 'Content type' with a dropdown menu set to 'Fire Department'; 'Author' with a text input field containing 'anonymous'; and 'Expire nodes' with a dropdown menu set to 'Never'. A 'Save' button is located at the bottom of the main area. A 'Help' link is visible in the top right corner of the main area.

9. Make sure to save your **Node processor** configuration.

Mapping your importer

Now we're ready to create our mapping. The mapping will associate our CSV file's columns of content and data with the custom fields we've added to our content type. Here we'll tell the **Feeds** module and our importer which fields to add our imported content to during the import process.

To get started with your mapping click on the **Mapping** link in the **Node processor** tab on your **Importer configuration** screen. The mapping screen will load. This screen shows a table that has the following columns:

- **SOURCE:** This is the name of your source column in the CSV – use the header label here.
- **TARGET:** This is the custom field you want to map the source to in your content type.
- **UNIQUE TARGET:** This checkbox allows you to set a unique target. Generally this will be available for your **GUID** and **Title** fields.
- **Remove checkboxes:** This allows you to remove any of the added mapped rows if you need to.

If you scroll down to the bottom of the mapping screen, you'll see a fieldset called **Legend**. If you expand this you'll see detailed descriptions of all your available target fields. This table will look similar to the following, depending on how much you've added to your content type:

Targets	
NAME	DESCRIPTION
URL	The external URL of the item. E. g. the feed item URL in the case of a syndication feed. May be unique.
GUID	The globally unique identifier of the item. E. g. the feed item GUID in the case of a syndication feed. May be unique.
Title	The title of the node.
Node ID	The nid of the node. NOTE: use this feature with care, node ids are usually assigned by Drupal.
User ID	The Drupal user ID of the node author.
Published status	Whether a node is published or not. 1 stands for published, 0 for not published.
Published date	The UNIX time when a node has been published.
Promoted to front page	Boolean value, whether or not node is promoted to front page. (1 = promoted, 0 = not promoted)
Sticky	Boolean value, whether or not node is sticky at top of lists. (1 = sticky, 0 = not sticky)
Comments	Whether comments are allowed on this node: 0 = no, 1 = read only, 2 = read/write.
Temporary target 1	A field that stores the source data temporarily so that it can be used with the Feeds Tamper rewrite plugin.
Body	The Body field of the node.
Headquarters Address 1	The Headquarters Address 1 field of the node.
Number of Stations	The Number of Stations field of the node.
Headquarters Address 2	The Headquarters Address 2 field of the node.
Headquarters City	The Headquarters City field of the node.
Headquarters State	The Headquarters State field of the node.
Headquarters Zip	The Headquarters Zip field of the node.

Adding a source and target

To add a source and target, first enter the name of your source into the field that contains the name of source field default text. So for example in our CSV we have the **fire_dept_name** column. Add that to the source field and then choose the **Title** as your target from the drop-down box. Make sure to remove the existing default mapping of the Title source to the Title target (if it exists), as we're going to specify our **fire_dept_name** as the source to use.

The screenshot shows a configuration interface with a source field and a target selection dropdown.

Source Field: fire_dept_name

LEGEND

Targets

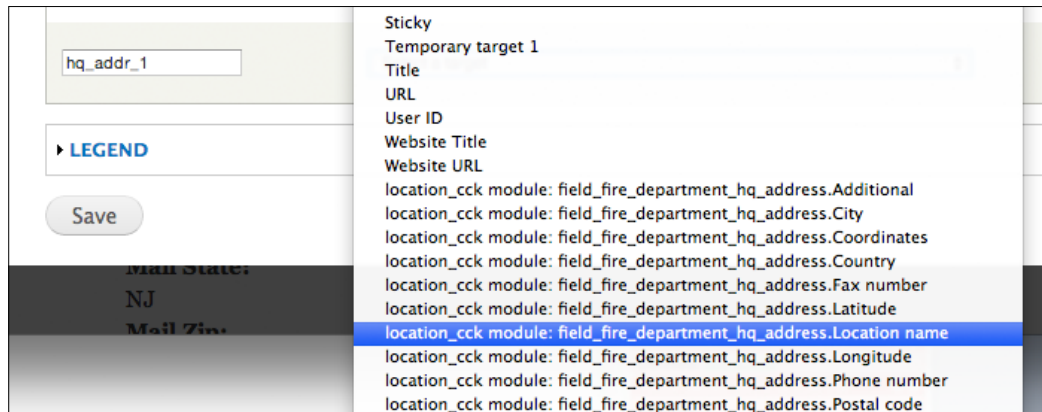
NAME
URL
GUID
Title
Node ID
User ID
Published status
Published date
Promoted to front page
Sticky
Comments
Temporary target 1

Select a target

- Active Firefighters Career
- Active Firefighters Paid Per Call
- Active Firefighters Volunteer
- Body
- Comments
- County
- Department Type
- Fire Department Image
- GUID
- Headquarters Address 1
- Headquarters Address 2
- Headquarters City
- Headquarters Fax
- Headquarters Phone
- Headquarters State
- Headquarters Zip
- Mail City
- Mail PO Box
- Mail State
- Mail Zip
- Mailing Address 1
- Mailing Address 2
- Node ID
- Non-Firefighting Civilian
- Non-Firefighting Volunteer
- Number of Stations
- Organization Type (Node reference by Feeds GUID)
- Organization Type (Node reference by Feeds GUID) -- allow duplicate nodes
- Organization Type (Node reference by Feeds URL)
- Organization Type (Node reference by Feeds URL) -- allow duplicate nodes
- Organization Type (Node reference by node ID)
- Organization Type (Node reference by node ID) -- allow duplicate nodes
- Organization Type (Node reference by node title)
- Organization Type (Node reference by node title) -- allow duplicate nodes
- Primary Agency for Employment
- Promoted to front page
- Published date
- Published status
- Sticky
- Temporary target 1
- Title**
- URL
- User ID

Make sure to click on the **Add** button first before clicking on the **Save** button on your mapping. Otherwise your source and target will not be added correctly.

To add the address fields you can map these to the specific **Location CCK** module fields as your target. For example map the **hq_addr_1** column of data to the **location_cck module: field_fire_department_hq_address.Street** location target, as shown in the following screenshot:



Make sure to check the Unique Target box next to the **GUID** field since this field will be used as our unique identifier.



Go ahead and add the remainder of your source and target mapped fields. Once you add all your mapped fields you should see an importer mapping that resembles the following :

SOURCE	TARGET	UNIQUE TARGET	
body	Body		<input type="checkbox"/> Remove
published	Published date		<input type="checkbox"/> Remove
guid	GUID	<input checked="" type="checkbox"/>	<input type="checkbox"/> Remove
fire_dept_name	Title	<input checked="" type="checkbox"/>	<input type="checkbox"/> Remove
hq_addr_2	Headquarters Address 2		<input type="checkbox"/> Remove
hq_addr_1	Headquarters Address 1		<input type="checkbox"/> Remove
hq_city	Headquarters City		<input type="checkbox"/> Remove
hq_state	Headquarters State		<input type="checkbox"/> Remove
hq_zip	Headquarters Zip		<input type="checkbox"/> Remove
mail_addr1	Mailing Address 1		<input type="checkbox"/> Remove
mail_addr2	Mailing Address 2		<input type="checkbox"/> Remove
mail_po_box	Mail PO Box		<input type="checkbox"/> Remove
mail_city	Mail City		<input type="checkbox"/> Remove
mail_state	Mail State		<input type="checkbox"/> Remove
mail_zip	Mail Zip		<input type="checkbox"/> Remove
hq_phone	Headquarters Phone		<input type="checkbox"/> Remove
hq_fax	Headquarters Fax		<input type="checkbox"/> Remove
county	County		<input type="checkbox"/> Remove
dept_type	Department Type		<input type="checkbox"/> Remove
website	Website URL		<input type="checkbox"/> Remove
number_of_stations	Number of Stations		<input type="checkbox"/> Remove
active_firefighters_career	Active Firefighters Career		<input type="checkbox"/> Remove
active_firefighters_volunteer	Active Firefighters Volunteer		<input type="checkbox"/> Remove

Now that you have completed your mapping make sure to save your mapping configuration once again. We're now ready to run our import.

Running an import process

To run the import you need to go to the import URL of your site. This will load the following screen:

Import	
Import	Description
Fire Department Importer	This is used to import firehouse content from CSV into nodes.
Node Import	Import nodes from CSV file.
User Import	Import users from CSV file.
Feed	Import RSS or Atom feeds, create nodes from feed items.
OPML Import	Import subscriptions from OPML files. Use together with "Feed" configuration.

Click on the **Fire Department Importer** link to open up the specific import screen for our importer. The **Importer** screen contains three tabs at the top: **Import**, **Delete items**, and **Log**. The **Import** tab is the default. This tab shows you the status of your import, and then gives you the import area where you can upload the CSV file that you want to import. You will see a CSV file listed (if you've already uploaded one), and a **Choose File** button to upload a brand new file. When you upload a new file it replaces the existing file. You can also override the delimiter here that's set for your import process.

The status will show you when you last ran your import and how many total items were imported. Items here refer to the **Feed Items** that are imported, which are also the nodes you are creating. In the following screenshot you'll see that an import that was executed four days ago and 888 items were imported. The name of the file uploaded is `fire-department.csv`:

Fire Department Importer

[Import](#)[Delete items](#)[Log](#)

Status

- Last import: 4 days ago.
- 888 imported items total.

Import

Import [CSV files](#) with one or more of these columns: body, published, guid, fire_dept_name, hq_addr_2, hq_addr_1, hq_city, hq_state, hq_zip, mail_addr1, mail_addr2, mail_po_box, mail_city, mail_state, mail_zip, hq_phone, hq_fax, county, dept_type, organization_type, website, number_of_stations, active_firefighters_career, active_firefighters_volunteer, active_firefighters_paid_per_call, non_firefighting_civilian, non_firefighting_volunteer, primary_agency_for_em.

- Columns **guid**, **fire_dept_name** are mandatory and values in these columns are considered unique: only one entry per value in one of these column will be created.
- [Download a template](#)

Delimiter



The character that delimits fields in the CSV file.

☐ No Headers

Check if the imported CSV file does not start with a header row. If checked, mapping sources must be named 'o', '1', '2' etc.

File

[fire-department.csv](#)

242.57 KB

text/csv

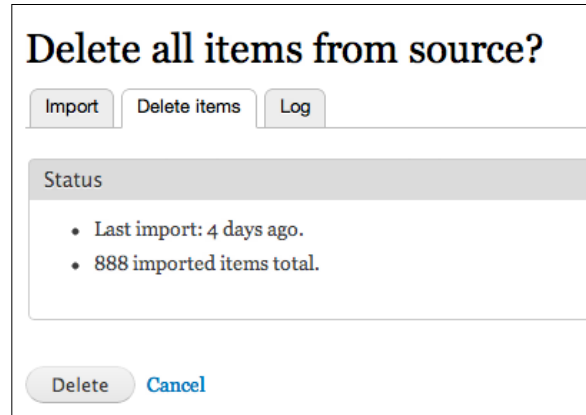
[Choose File](#)

No file chosen

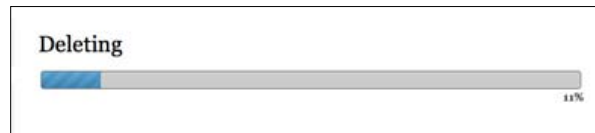
Select a different file from your local system.

[Import](#)

If you click on the **Delete items** tab, you will see a **Delete** button. Click on this button to delete all the items or nodes that you've imported into the site. This is a quick and easy method of clearing out all of your imported nodes but bear in mind that they really will be deleted. For the sake of this example, we're going to delete all the previously imported nodes so that we can run a clean import again. The delete screen looks as follows:



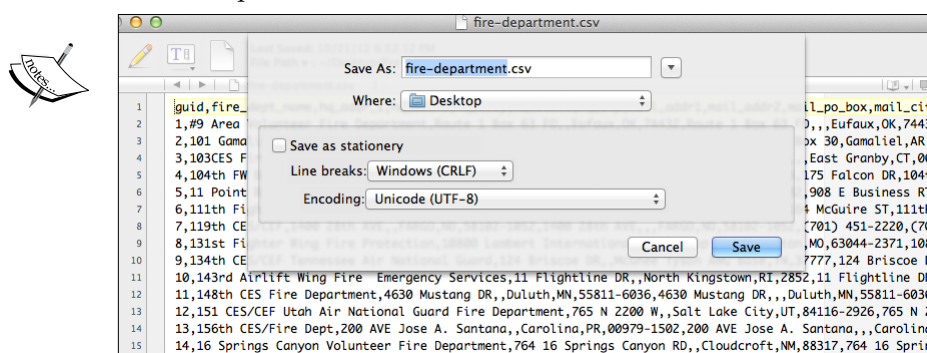
When you click on **Delete** you'll see a progress bar, as shown in the following screenshot, showing the deletion process taking place:



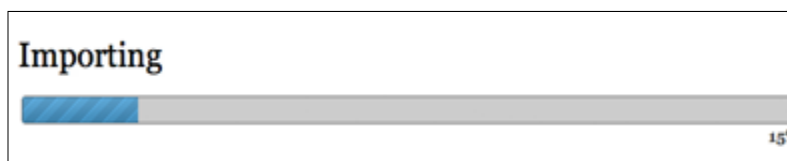
At the end of the delete process you should get a message informing that the nodes were deleted.

Now if you check your content on the site, you should not have any of these nodes, as they were just deleted. Additionally your main **Import** screen will report that the status shows **No imported items**.

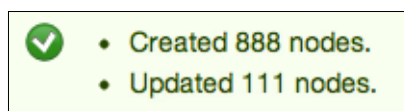
If you are uploading your CSV file from a Mac computer, you need to confirm that the file is using Windows formatted UTF-8 encoding. You can confirm this by using a program such as TextWrangler (<http://www.barebones.com/products/textwrangler/>). You can re-open the file in TextWrangler using UTF-8 encoding. Then you can re-save your CSV file with Windows (CRLF) formatting. The following screenshot shows this process:



Let's run the import. First upload your CSV file using the file uploader. Once you upload the CSV file, go ahead and click on the **Import** button. Again you'll see a progress bar showing you the import status in real time:



You should see a status message at the end of the import notifying about the success of your import and how many nodes were created and updated depending on whether you are updating the nodes.



You can load your main content administration screen to see all of your imported nodes in the content table, as shown in the following screenshot:

<input type="checkbox"/>	Audubon Fire Department	Fire Department	Anonymous (not verified)	published	10/08/2012 - 14:19	edit	delete
<input type="checkbox"/>	Test Department	Fire Department	admin	published	10/08/2012 - 14:14	edit	delete
<input type="checkbox"/>	Atwater Fire Department	Fire Department	Anonymous (not verified)	published	08/27/2012 - 22:50	edit	delete
<input type="checkbox"/>	Attica Fire Department	Fire Department	Anonymous (not verified)	published	08/27/2012 - 22:50	edit	delete
<input type="checkbox"/>	Auburn Fire Department	Fire Department	Anonymous (not verified)	published	08/27/2012 - 22:50	edit	delete
<input type="checkbox"/>	Auburn Volunteer Fire Department	Fire Department	Anonymous (not verified)	published	08/27/2012 - 22:50	edit	delete
<input type="checkbox"/>	Atlanta Volunteer Fire Department	Fire Department	Anonymous (not verified)	published	08/27/2012 - 22:50	edit	delete
<input type="checkbox"/>	Atkins Volunteer Fire Department	Fire Department	Anonymous (not verified)	published	08/27/2012 - 22:50	edit	delete
<input type="checkbox"/>	Atkinson Fire Department	Fire Department	Anonymous (not verified)	published	08/27/2012 - 22:50	edit	delete
<input type="checkbox"/>	Athens Volunteer Fire Department	Fire Department	Anonymous (not verified)	published	08/27/2012 - 22:50	edit	delete
<input type="checkbox"/>	Atoka Fire Department	Fire Department	Anonymous (not verified)	published	08/27/2012 - 22:50	edit	delete
<input type="checkbox"/>	Athens Fire Department	Fire Department	Anonymous (not verified)	published	08/27/2012 - 22:50	edit	delete
<input type="checkbox"/>	Ashland Fire Department	Fire Department	Anonymous (not verified)	published	08/27/2012 - 22:50	edit	delete
<input type="checkbox"/>	Ashton Fire Department	Fire Department	Anonymous (not verified)	published	08/27/2012 - 22:50	edit	delete
<input type="checkbox"/>	Arlington Volunteer Fire Department	Fire Department	Anonymous (not verified)	published	08/27/2012 - 22:50	edit	delete

If you view the details of one of your imported nodes, you should see all of the imported content that was mapped into your content type's custom fields. The **Department Type** and **County** should be links to the term URL pages, and the **Organization Type** should be a link to the Organization Type node reference. You should see a node that looks similar to the following:

Auburn Volunteer Fire Department

[View](#)[Edit](#)[Log](#)

Submitted by Anonymous (not verified) on Sun, 10/21/2012 - 14:52

Fire Department HQ Address:
1517 Seventeenth ST
Auburn, NE 68305
United States
Phone: 402-274-3517
See map: [Google Maps](#)

Number of Stations:
1
Department Type:
[Volunteer](#)

Organization Type:

[Local \(includes career combination and volunteer\)](#)

Submitted by admin on Sun, 08/19/2012 - 17:25

Mailing Address 1:
Mailing Address 2:
Mail PO Box:
PO Box 165
Mail City:
Auburn
Mail State:
NE
Mail Zip:
68305
County:
[NEMAHA](#)

Active Firefighters Career:
☐

Active Firefighters Volunteer:
☐

Active Firefighters Paid Per Call:
☐

Non-Firefighting Civilian:
☐

Notice here that the Location field data mapped correctly and the **Location** module has automatically added a link to the Google Maps view for the specific location. This is one benefit of mapping to the Location fields, as you can leverage Google Maps automatically via the module.

Summary

In this chapter we successfully created an importer and walked through the entire importer configuration process in detail. We also created a mapping for our imported content. We then executed the import process showing how we can quickly import thousands of nodes of content into our Drupal site. We also discussed updating, replacing, and deleting our imported nodes.

In the next chapter we're going to continue to tweak our import process by adding tampers to our importer using the **Feeds Tamper** module. This will allow us to pre-process our data during the actual import process and make tweaks to the formatting of the data and content as we import it.

4

Feeds Tamperers

We have successfully imported roughly one thousand rows of content from our CSV file into corresponding nodes on our Drupal site. We now have the bulk of content and data imported and sitting in custom fields within our content types in Drupal. We have imported terms that are now term references or clickable tag links on our nodes, and we have also imported data that links to other nodes on our site in the form of Node References. In the previous chapter, we built and configured our importer and mapped our CSV columns to their corresponding content type field. We then ran our import.

All the content in our imported CSV and in the previous chapter's importer mapping were of single value data. In other words, for each cell in our columns in the CSV file, we only have one value that we're importing into our content type field. What happens if you want to import multiple values of data into one field? Additionally what if we want to strip out specific HTML tags or other special characters from our content during the import process? Can we import content and format it to be uppercase or all lowercase values? Can we change words in our content on import, for example replacing all instances of "Govt." with "Government"?

In this chapter we're going to explore how to tweak our importer to allow for special handling during the import process by using a module called **Feeds Tamper**.

We'll be covering the following topics in this chapter:

- Using the **Feeds Tamper** module
- Adding a tamper plugin to imported content
- Running a feed import update with our tampers applied
- Testing the import to confirm tamper worked

Using the Feeds Tamper module

In *Chapter 1, Preparing Drupal for Content Migration*, we installed the **Feeds Tamper** module. Recall that we enabled the **Feeds Tamper** and **Feeds Tamper Admin UI** modules on our modules administration screen. If you have not installed the **Feeds Tamper** module yet you can download its latest version from the module project page at http://drupal.org/project/feeds_tamper. The version we're using in this book is version 7.x-1.0-beta3 but at the time of publication the module has been updated to its beta4 version.

There are two methods of getting to the Feeds Tamper configuration for your importer. First, you can load your **Feeds Importers** configuration screen and click on the **Tamper** link next to the importer that you want to add the tamper to. This is shown in the following screenshot:

NAME	DESCRIPTION	ATTACHED TO	STATUS	OPERATIONS	ENABLED
Fire Department Importer	This is used to import firehouse content from CSV into nodes.	[none]	Normal	Edit Export Clone Delete Tamper	<input checked="" type="checkbox"/>

You can also click on the **Edit** link to edit your importer, and then click on the **Mapping** link in your Node processor settings tab. When the settings screen loads for your mapping, you will see a **Configure Feeds Tamper** link at the top-left corner of your mapping table:

Mapping for Node processor Help			
Define which elements of a single item of a feed (= <i>Sources</i>) map to which content pieces in Drupal (= <i>Targets</i>). Make sure that at least one definition has a <i>Unique target</i> . A unique target means that a value for a target can only occur once. E. g. only one item with the URL http://example.com/content/1 can exist.			
Configure Feeds Tamper			
SOURCE	TARGET	UNIQUE TARGET	
body	Body	<input type="checkbox"/>	Remove
published	Published date	<input type="checkbox"/>	Remove
guid	GUID	<input checked="" type="checkbox"/>	Remove
fire_dept_name	Title	<input checked="" type="checkbox"/>	Remove

Either of the previous methods will load the same Feeds Tamper configuration screen, which will look as follows when it is launched:

body -> Body

DESCRIPTION	PLUGIN	STATUS	OPERATIONS	ENABLED
No plugins defined.				
+ Add plugin				

published -> Published date

DESCRIPTION	PLUGIN	STATUS	OPERATIONS	ENABLED
No plugins defined.				
+ Add plugin				

guid -> GUID

DESCRIPTION	PLUGIN	STATUS	OPERATIONS	ENABLED
No plugins defined.				
+ Add plugin				

fire_dept_name -> Title

DESCRIPTION	PLUGIN	STATUS	OPERATIONS	ENABLED
No plugins defined.				
+ Add plugin				

The **Tamper plugins** screen for your importer shows each of the fields that you are mapping, any existing tamper plugins you have added, and a link to add a plugin. The **Feeds Tamper** module allows you to add a specific type of plugin to each field that will modify the data that's being imported into that field during the actual import process.

Currently all of our fields should show a **No plugins defined** text description, as we have not added any tamper plugins yet. Let's go ahead and add a tamper plugin to one of our fields.

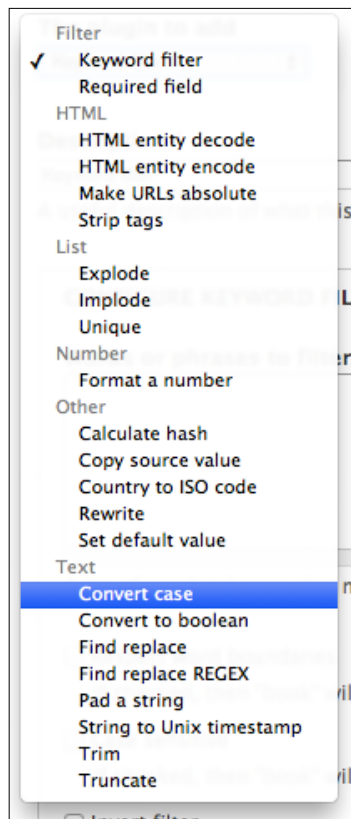
Adding a tamper plugin

You will most likely be adding a tamper plugin to your importer to correct issues in the content you want to import. For example, in the CSV file we are using the `hq_city` column that contains the majority of the cities in initial capital and then lowercase. However there are instances of city names in uppercase. As we're striving to import consistent data and present our content consistently, we may want to convert the all-caps words to initial caps on those specific values. We can do this by applying a tamper plugin to our `hq_city` field. Let's go ahead and do this:

1. Launch your **Fire Department importer** and go to its Feed tamper configuration screen.
2. Look for the `hq_city` field. You'll notice that it currently says **No plugins defined.**
3. Click on the **Add plugin** link:

hq_city -> Headquarters City				
DESCRIPTION	PLUGIN	STATUS	OPERATIONS	ENABLED
No plugins defined.				
+ Add plugin				

4. Select the **Convert case** plugin from the plugin drop-down list:



5. Leave the **CONFIGURE CONVERT CASE** selected to the default **Title case** value, as we want to set the value of our `hq_city` in initial capital.
6. You can also add a custom description to your plugin so that you can have custom plugins for each field. I'll tweak the description to read `Convert case of HQ City`. I'll also tweak the machine name of the specific plugin to match this description. My machine name is now `convert_case_hq_city`.



If you plan to reuse plugins across many fields, you may want to leave the description and machine name set to their default values.

You should now see something similar to the following:

The plugin to add

Convert case

Description *

Convert case of HQ City

Machine name: convert_case [Edit]

A useful description of what this plugin is doing.

CONFIGURE CONVERT CASE

How to convert case

☒ Title case

☐ Lower Case

☐ Upper case

Add

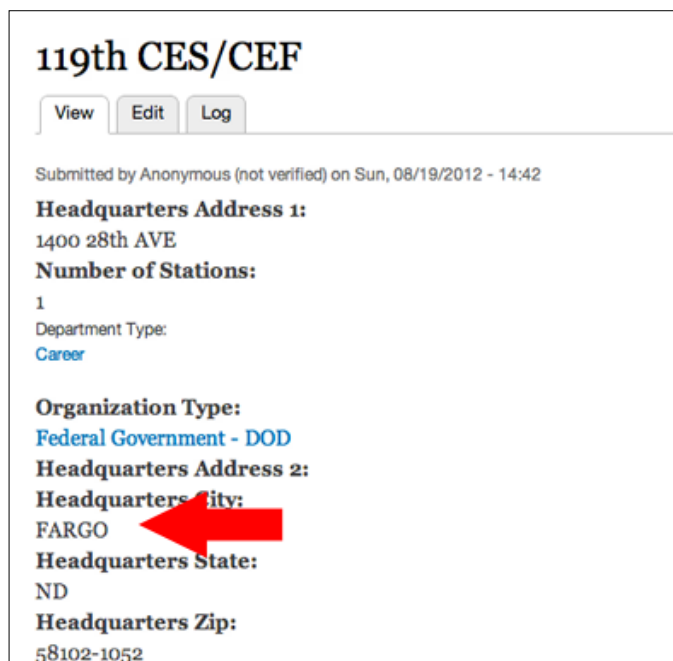
- Click on the **Add** button.
- Now in your `hq_city` table on the Feeds Tamper configuration screen you should see your **Convert case of HQ City** plugin. Your table will also show the plugin as enabled and you can now edit or delete the plugin.

hq_city -> Headquarters City				Show row weights
DESCRIPTION	PLUGIN	STATUS	OPERATIONS	ENABLED
+ Convert case of HQ City	Convert case	Normal	Edit Delete	<input checked="" type="checkbox"/>
+ Add plugin				

- Scroll to the bottom of your Feeds Tamper configuration screen and click on the **Save** button to confirm all of your changes have been saved.

Running the import with the tamper plugin

We're now ready to run our import again, this time using our tamper plugin on the `hq_city` field. We're going to simply update all of our previously imported nodes by re-running the import process. We'll then check our imported nodes to confirm that all of the `hq_city` values are composed of words in initial capital. We can confirm this by comparing our CSV data to the imported data specifically on any values that are all capitals. They should change to be initial capital on the import process and show the same changed value on our node. For example, on this node for **119th CES/CEF**, notice that the `hq_city` value is set to **FARGO**. Once we re-run the import to update this node, the tamper plugin should make this value initial capital and then lowercase.



119th CES/CEF

View Edit Log

Submitted by Anonymous (not verified) on Sun, 08/19/2012 - 14:42

Headquarters Address 1:
1400 28th AVE

Number of Stations:
1

Department Type:
[Career](#)

Organization Type:
[Federal Government - DOD](#)

Headquarters Address 2:
Headquarters City:
FARGO

Headquarters State:
ND

Headquarters Zip:
58102-1052

To run our import again, go to your import URL and click on the **Fire Department Importer** link. Now scroll down and click on the **Import** button. The import process will run an update. Once the process finishes you should see a notice telling you that a specific number of nodes, in our case 192, have been updated. These are the 192 nodes that had all `hq_city` values in uppercase.

Testing the tamper plugin results

Now that we have run our import update, let's go back to load the **119th CES/CEF** node and confirm that the **Headquarters City** is now reading **Fargo** instead of **FARGO**. Loading that node does indeed show that our `hq_city` value is now **Fargo**. Our tamper plugin worked successfully.

119th CES/CEF

ViewEditLog

Submitted by Anonymous (not verified) on Sun, 08/19/2012 - 18:12

Headquarters Address 1:
1400 28th AVE

Number of Stations:
1

Department Type:
[Career](#)

Organization Type:
[Federal Government - DOD](#)

Headquarters Address 2:

Headquarters City:
Fargo

Headquarters State:
ND

Notice that the **Mail City** is also in uppercase. You could now go back to your tamper plugin configuration screen and add this same plugin to the **Mail City** field so that it would affect those values as well.



There are many more examples of Feeds Tamper plugins and other related Feeds Tamper documentation on its project documentation page via Drupal.org, at <http://drupal.org/node/1246562>.

Summary

In this chapter we successfully added the Feeds Tamper plugins to fields within our importer. We added a **Convert case** plugin that takes values and adds an initial cap to the value and leaves the remainder of the value in lowercase. This fixes some values in our imported data that are in uppercase and leaves that data in an inconsistent imported format. Using the tamper plugins helps us to import consistent data and keep our content consistent in its presentation in Drupal. This helps us to refactor and clean up our content as we import it.

In the next chapter, we're going to look at how best to maintain our migration path by using the **Feeds** module's built-in abilities to clone, edit, and update importers.

5

Maintaining a Migration Path

Now that we have our content type and importer configured and we have successfully imported all of our content and data, we should start thinking about how best to maintain our workflow and migration process over time. This will be especially important if we are going to be running additional imports in the future to update our data and content. This may occur if we're actively developing a website with some of the imported content, but we need to append new content later to our site from an existing legacy website. In this case we'll need to continue to use our **Feeds** importer or build a new importer that can add and append data to the content we've already imported into our site.

We also may want to quickly build another content type based on the original type we set up in the earlier chapter and also re-use an existing importer with our new content type. We might also want to add a new field to our content type to collect more imported data. This would involve making a clone of an existing importer and tweaking its mapping configuration.

In this chapter we're going to explore how best to manage and maintain our content type and importers, and keep our process and workflow flexible and consistent.

We'll be covering the following topics in this chapter:

- Re-running imports to replace and update existing nodes
- Cloning importers
- Running an import update
- Confirming our update worked

Updating imports

At this point you have a fully functional import with associated mapping that imports data and content from a CSV file into your content type nodes in your Drupal site. This import is working perfectly so far. You'll most likely come across an instance where you'll need to either update the imported content with new data or add new data to existing nodes. You can easily do this by running an update importer that only targets the fields that need to be updated and also adds the new content to existing nodes. The update process will work without changing or updating all of the content. The updates will only tweak or add the new content without manipulating any of the other fields in your existing content type. So, for example, if a fire department gets updated regularly in the CSV with a new number of stations, you can easily run this update importer and tweak the number of stations data in your existing node content without messing with any of your other existing node data. The update process is then seamless and you can continue to leverage **Feeds** to run update imports.

There are some nuances to the import process that we'll outline by showing the entire update process and how we'll maintain and manage it. Let's get started.

Cloning importers

The scenario we have is an existing field with data in our content type that needs to be updated with new content via the import process. So, for example, your manager walks into your office and hands you a brand new updated CSV file that has new data in the `number_of_stations` column of the CSV. Many of the fire departments have built new stations this year so the data in the website needs to reflect this update.

To do this the first thing we need to do is build a new importer to handle just the updated data. We don't want to replace all of our existing node data, or even update it for that matter, as there may have been changes to data by content editors within the website since we last executed the import process. All we need to do is update one field in our nodes with the updated data.

So let's first be clear on what we need to update. Our existing importer contains a mapping for the **number_of_stations** source that maps to the **Number of Stations** field in our content type. This is the field we need to update. So let's first make a clone of our existing importer and use this cloned importer to handle this updated field and data only.

The cloning process

Go ahead and launch your importer administration screen and click on the **Clone** link next to the **Fire Department Importer**, as shown in the following screenshot:

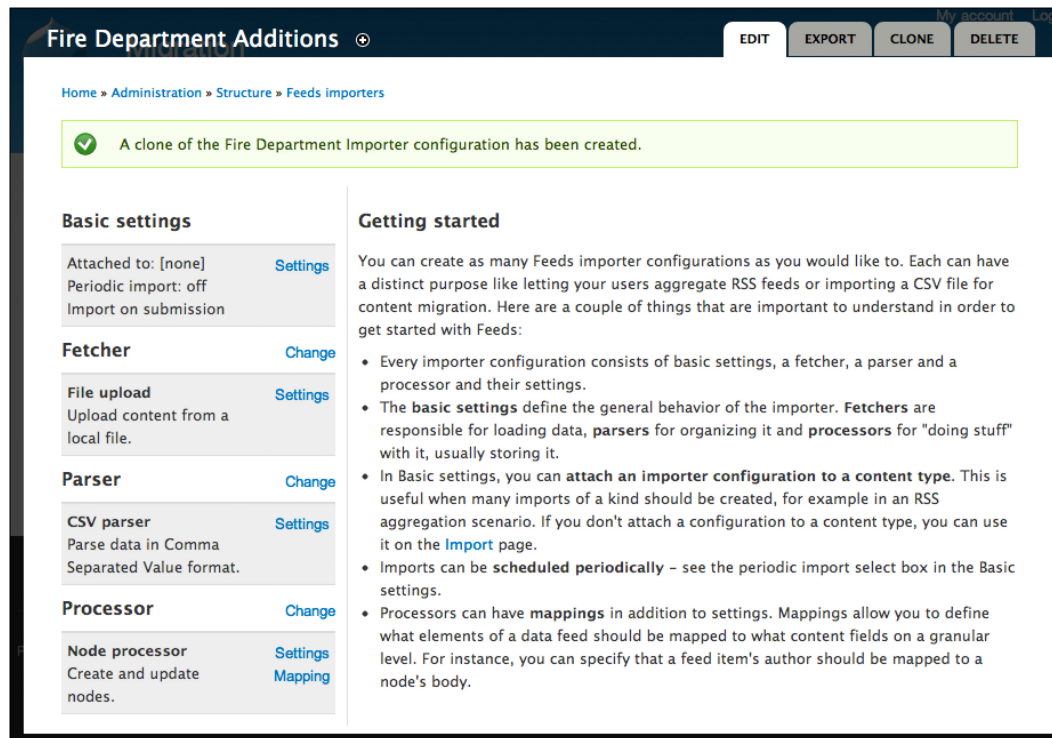


This will launch the importer configuration where you can add a new name, **Fire Department Additions** and the description, **This importer will update the number_of_stations field**. Be very specific with your description so you'll know what this importer does in future if you keep it on your site.

 A screenshot of the 'Fire Department Importer' configuration page. The page has a dark header with the title 'Fire Department Importer' and buttons for 'EDIT' and 'E'. Below the header is a breadcrumb trail: 'Home » Administration » Structure » Feeds importers » Fire Department Importer'. The main form has two sections: 'Name' and 'Description'. The 'Name' section has a text input field containing 'Fire Department Additions' and a 'Machine name' field containing 'fire_department_additions'. Below the name field is a note: 'A natural name for this configuration. Example: RSS Feed. You can always change this name later.' The 'Description' section has a text input field containing 'This importer will update the number_of_stations field.' and a note: 'A description of this configuration.' At the bottom of the form is a 'Create' button.

Click on the **Create** button. The next screen that launches will contain your new cloned importer configuration. There will be a message telling you about the status of the new configuration – **A clone of the Fire Department Importer configuration has been created.**

At this point your configuration will look exactly like the importer you cloned in terms of configuration settings.



Now what we need to do here is just change the Mapping configuration. All of our other importer settings can stay in place on this specific cloned importer. We need to update the mapping to remove all the fields except the one that we want to update.

1. To do this, click on the **Mapping** link.
2. When the mapping configuration loads you'll notice a **Remove** checkbox in each row of mapped data. We want to remove all the mapped fields leaving only the `number_of_stations` mapping and the `guid` mapping. `guid` is the only required field in the mapping as the GUID acts as our unique identifier field and this way we can be sure that we're updating the correct node when we run our new importer since it will be referencing the same GUID.
3. So check the **Remove** checkbox next to all fields except for the **number_of_stations** and the **guid**, as shown in the following screenshot; you should have a screen with most of the **Remove** checkboxes checked:

SOURCE	TARGET	UNIQUE TARGET	
body	Body		<input checked="" type="checkbox"/> Remove
published	Published date		<input checked="" type="checkbox"/> Remove
guid	GUID	<input checked="" type="checkbox"/>	<input type="checkbox"/> Remove
fire_dept_name	Title	<input type="checkbox"/>	<input checked="" type="checkbox"/> Remove
mail_addr1	Mailing Address 1		<input checked="" type="checkbox"/> Remove
mail_addr2	Mailing Address 2		<input checked="" type="checkbox"/> Remove
mail_po_box	Mail PO Box		<input checked="" type="checkbox"/> Remove
mail_city	Mail City		<input checked="" type="checkbox"/> Remove
mail_state	Mail State		<input checked="" type="checkbox"/> Remove
mail_zip	Mail Zip		<input checked="" type="checkbox"/> Remove
county	County		<input checked="" type="checkbox"/> Remove
dept_type	Department Type		<input checked="" type="checkbox"/> Remove
website	Website URL		<input checked="" type="checkbox"/> Remove
number_of_stations	Number of Stations		<input type="checkbox"/> Remove
active_firefighters_career	Active Firefighters Career		<input checked="" type="checkbox"/> Remove
active_firefighters_volunteer	Active Firefighters Volunteer		<input checked="" type="checkbox"/> Remove
active_firefighters_paid_per_call	Active Firefighters Paid Per Call		<input checked="" type="checkbox"/> Remove
non_firefighting_civilian	Non-Firefighting Civilian		<input checked="" type="checkbox"/> Remove
non_firefighting_volunteer	Active Firefighters Volunteer		<input checked="" type="checkbox"/> Remove
primary_agency_for_em	Primary Agency for Employment		<input checked="" type="checkbox"/> Remove

- Once you select the required **Remove** checkboxes, go ahead, and click on the **Save** button. This will immediately refresh your screen and you'll see your three remaining saved fields. The other mapped fields have been removed from this importer.

Configure Feeds Tamper

SOURCE	TARGET	UNIQUE TARGET	
guid	GUID	<input checked="" type="checkbox"/>	<input type="checkbox"/> Remove
number_of_stations	Number of Stations		<input type="checkbox"/> Remove

▶ **LEGEND**

Now that we have our new importer with the correct mapping we should double-check that we're going to be only updating the nodes when we import. To do this, click on the **Settings** link of the Node processor, which will load the following screen where you should have the **Update existing nodes (slower than replacing them)** option selected:

Settings for Node processor [Help](#)

Update existing nodes
☐ Do not update existing nodes
☐ Replace existing nodes
☒ Update existing nodes (slower than replacing them)
Existing nodes will be determined using mappings that are a "unique target".

Text format *

Full HTML

Select the input format for the body field of the nodes to be created.

Content type

Fire Department

Select the content type for the nodes to be created. **Note:** Users with "import fire_department_additions feeds" permissions will be able to **import** nodes of the content type selected here regardless of the node level permissions. Further, users with "clear fire_department_additions permissions" will be able to **delete** imported nodes regardless of their node level permissions.

Author

anonymous

Select the author of the nodes to be created – leave empty to assign "anonymous".

Expire nodes

Never

Select after how much time nodes should be deleted. The node's published date will be used for determining the node's age, see Mapping settings.

Save

Running the import update

Now we have completed tweaking our cloned importer, we are ready to run the import process and update our nodes. For the purposes of this example, I'm going to update one node in my existing content using one row of data for the `number_of_stations` field from a new CSV file. This will show you how the data in that field is updated using our new importer, and it'll also be easy to check and confirm since it is just one node that we will need to check. The node I'm going to update is for the **Audubon Fire Department** and it currently has **1** as its value for `number_of_stations`.

In the new CSV file all the columns for the fields that we're not importing have been removed. The only columns of data I have are for the `guid`, `fire_dept_name`, and the column we want to update, which is the `number_of_stations` field. You should have something similar to the following screenshot. Make sure that you have the correct `guid` and `fire_dept_name` values to match the record you're updating so that it does update the correct record.

	A	B	C	D
1	id	fire_dept_name	number_of_stations	
2	992	Audubon Fire Department	3	
3				

Now that we have the correct CSV format and data, we're ready to run the import. The import process will be the same as our previous process that we executed in the earlier chapter.

1. First launch your/import page.

Import

Import	Description
Fire Department Additions	This importer will update the number_of_stations field.
Fire Department Importer	This is used to import firehouse content from CSV into nodes.
Node Import	Import nodes from CSV file.
User Import	Import users from CSV file.
Feed	Import RSS or Atom feeds, create nodes from feed items.
OPML Import	Import subscriptions from OPML files. Use together with "Feed" configuration.

2. Now click on the **Fire Department Additions** link. This will launch the import screen.

3. Choose your new CSV file from your desktop after clicking on the **Choose File** button.

Fire Department Additions

ImportDelete itemsLog

Status

- Last import: 4 min ago.
- 1 imported items total.

Import

Import [CSV files](#) with one or more of these columns: `guid`, `fire_dept_name`, `number_of_stations`.

- Columns **guid**, **fire_dept_name** are mandatory and values in these columns are considered unique: only one entry per value in one of these column will be created.
- [Download a template](#)

Delimiter

,

↕

The character that delimits fields in the CSV file.

☐ No Headers

Check if the imported CSV file does not start with a header row. If checked, mapping sources must be named 'o', '1', '2' etc.

File

[fire-department-additions-new.csv](#)

67 bytes

text/csv

Choose File

No file chosen

Select a different file from your local system.

Import

4. Now click on the **Import** button.
5. When your import process runs you should get a message that tells you that the import has updated a node and the import executed successfully.

- To test that the new data has been inserted and **Number of Stations** has been updated, load the node in question and review that field's data. The **Number of Stations** field in this case should now be **3**. Notice that no other data was updated.

Audubon Fire Department

[View](#) [Edit](#) [Log](#)

Submitted by Anonymous (not verified) on Mon, 08/27/2012 - 18:50

Headquarters Address 1:
221 W Merchant ST

Number of Stations: 3

Department Type:
[Volunteer](#)

You have successfully completed the import update process.

Summary

In this chapter we successfully updated our imported content with new data and content by running a new import process. First, we cloned our existing importer and made tweaks to its configuration so we would be updating specific fields but not others. We removed the mapped fields that we did not want to update or touch with our new import. We then executed the import process again and our content was updated with the specific field's new data.

In the next chapter we're going to look at exporting and migrating our content type and importers to other sites by using the **Features** module, a very powerful tool for migrating Drupal core and contributed module configurations.

6

Packaging Content Types and Feeds Importers

It would be helpful if we could easily take the content types and importers we built in the previous chapters and somehow package up their entire configurations and settings into something like a module. This way we could easily move these content types and importers to another version of our site or to another Drupal site altogether, as their configurations would be stored in code. One use case for this is when you have multiple versions of your site – a development site (your local environment in MAMP), a staging or sandbox site where you test your site, and a production site. Many projects you work on in the Drupal environment will fit into this deployment process of developing a site locally, pushing your code to a site in a staging or test server, and then deploying your code to production.

You may also want to install a separate Drupal site and test your content types and importers on that separate site. How do we do this without having to rebuild the content type manually and create the importers on each site from scratch? If we have to rebuild our types and importers based on our original site's configurations then we're opening ourselves up for error – we may miss a specific configuration while rebuilding that could be crucial. For example, you may add a field to your type on your local development site but then forget to add the same field to your staging site.

This is where the **Features** module steps in. It is a powerful module that will allow us to package up our content types and feeds importers' configurations into code.

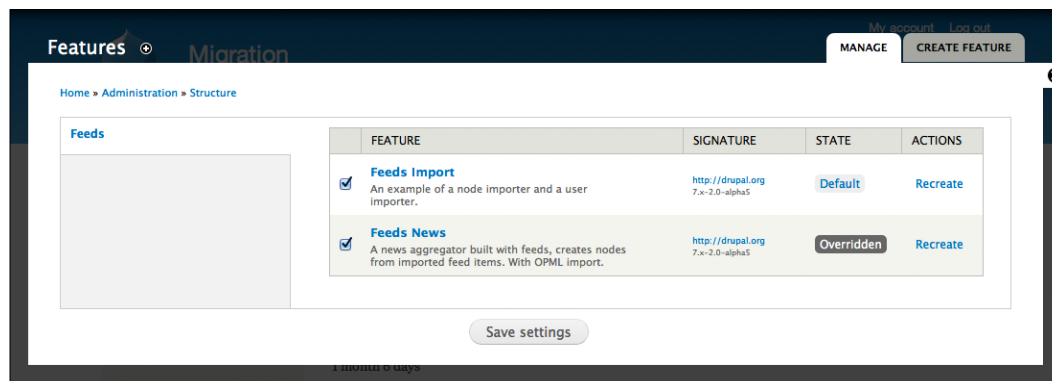
We'll cover the following topics in this chapter:

- Installing features
- Creating a feature for your content type and for your feeds importer
- Overriding features
- Reviewing feature overrides and reverting the features code
- Sharing a **Features** module and implementing it on a separate Drupal site

Features

Let's get started. First, we will look at some background information on what **Features** does. The code that the **Features** module will give us is in the form of module files sitting in a module folder that we can save to our `/sites/all/modules` directory, as we would do for any other contributed module. Using this method, we will have the entire configuration that we spent hours on building, saved into a module file and in code. The **Features** module will keep track of the tweaks we make to our content type configuration or importer for us. If we make changes to our type or importer we simply save a new version of our **Features** module.

The **Features** module configuration and the setup screen is at **Structure | Features** or you can go to this path: `admin/structure/features`. There is no generic configuration for **Features** that you need to worry about setting up. If you have the **Feeds** module installed as we do, you'll see two example features that the **Feeds** module provides—**Feeds Import** and **Feeds News**. You can use these provided features or create your own. We're going to create our own in the next section. You should see the following screen at this point:



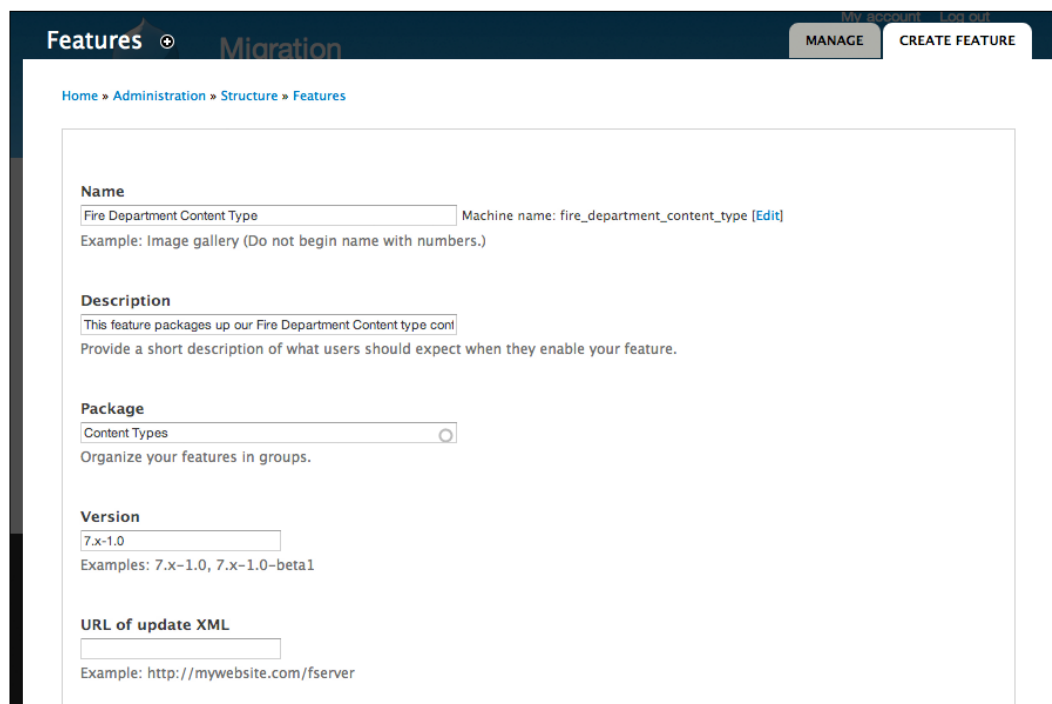
Building a content type feature

We have two custom content types so far on our site, **Fire Department** and **Organization Type**. Let's package up the Fire Department content type as a feature so that the **Features** module can start to keep track of each content type configuration and any changes we make going forward.

Creating and enabling the feature

First click on the **Create Feature** tab on your **Features** administration screen. The screen will load a new create feature form. Now follow these steps to create your first feature. We're going to package up our Fire Department content type:

1. Enter a name for your feature. This should be something specific such as Fire Department Content Type.
2. Add a description for the feature. This should be something like This feature packages up our Fire Department Content type configuration.
3. You can create a specific package for your feature. This will help to organize and group your features on the main Features admin screen. Let's call this package Content Types.
4. Version your feature. This is very important as your feature is going to be a module. It's a good idea to version number your feature each time you make a change to it. Our first version will be 7.x-1.0.
5. Leave the URL of update XML blank for now. By this point you should see the following:



Features Migration MANAGE CREATE FEATURE

Home » Administration » Structure » Features

Name
 Machine name: fire_department_content_type [\[Edit\]](#)
 Example: Image gallery (Do not begin name with numbers.)

Description

 Provide a short description of what users should expect when they enable your feature.

Package
 ○
 Organize your features in groups.

Version

 Examples: 7.x-1.0, 7.x-1.0-beta1

URL of update XML

 Example: http://mywebsite.com/fserver

6. Now we're going to add our components to the feature. As this feature will be our Fire Department content type configuration, we need to choose this content type as our component. In the drop-down box select **Content types: node**.
7. Now check the **Fire Department** checkbox. When you do this you'll see a timer icon appear for a second and then magically all of your content type fields, associated taxonomy, and dependencies will appear in the table to the right. This means that your feature is adding the entire content type configuration.
8. **Features** is a smart module. It will automatically associate any fields, taxonomy or other dependencies and requirements to your specific feature configuration. As our content type has taxonomy vocabulary associated with it (in the form of the term reference fields that we created in an earlier chapter) you'll notice that both **country** and **fire_department_type** are in the **Taxonomy** row of the feature table. You should now see the following:

Edit components

Content types: node

☐ Article
☐ Basic page
☒ Fire Department
☐ Organization Type

FEATURES_API

1

FIELDS

node-fire_department-body node-fire_department-field_fire_active_c
 node-fire_department-field_fire_active_fire_paid_p
 node-fire_department-field_fire_active_volunt
 node-fire_department-field_fire_county
 node-fire_department-field_fire_department_type
 node-fire_department-field_fire_hq_addr2
 node-fire_department-field_fire_hq_address_1
 node-fire_department-field_fire_hq_city
 node-fire_department-field_fire_hq_fax
 node-fire_department-field_fire_hq_phone
 node-fire_department-field_fire_hq_state
 node-fire_department-field_fire_hq_zip
 node-fire_department-field_fire_mail_addr1
 node-fire_department-field_fire_mail_addr2
 node-fire_department-field_fire_mail_city
 node-fire_department-field_fire_mail_po_box
 node-fire_department-field_fire_mail_state
 node-fire_department-field_fire_mail_zip
 node-fire_department-field_fire_non_fire_civilian
 node-fire_department-field_fire_non_fire_volunteer
 node-fire_department-field_fire_number_of_stations
 node-fire_department-field_fire_organization_type
 node-fire_department-field_fire_primary_agency
 node-fire_department-field_fire_website

CONTENT TYPES

fire_department

TAXONOMY

country fire_department_type

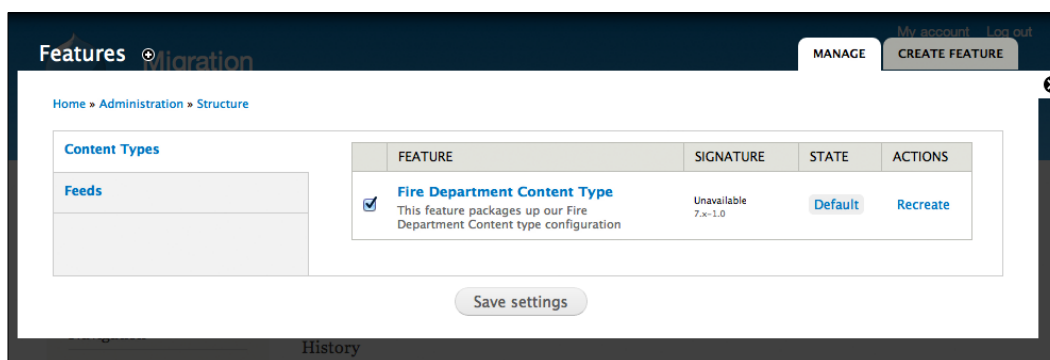
DEPENDENCIES

features field_sql_storage link node node_reference number
 taxonomy text

Normal Auto-detected Provided by dependency

Download feature

9. Now click on the **Download feature** button at the bottom of the screen to download the actual module code for our **Fire Department** feature module. Clicking on **Download feature** will download the .tar file of the module to your local computer.
10. Find the .tar file and then extract it into your /sites/all/modules/ directory. For organizational best practices, I recommend placing it into a / custom directory within your /sites/all/modules as this is really a custom module. So now you should see a folder called fire_department_content_type in your /sites/all/modules/custom folder. This folder contains the feature module files that you just downloaded.
11. Now if you go back to your main **Features** administration screen you will see a new tab titled **Content Types** that contains your new feature module called **Fire Department Content Type**. Currently this feature is disabled. You can notice the version number in the same row.
12. Go ahead and check the checkbox next to your feature and then click on the **Save settings** button. What you are doing here is enabling your feature as a module on your site and now your content type's configuration will always be running from this codebase.
13. When you click on **Save settings**, your feature should now be enabled and showing **Default** status.



When a feature is in **Default** state this means that your configuration (in this case the Fire Department content type) matches your feature modules codebase. This specific feature is now set up to keep track of any changes that may occur to the content type. So for example if you added a new field to your content type or tweaked any of its existing fields, display formatters or any other part of its configuration, that feature module would have a status of **Overridden**. We'll demonstrate this in the next section.

The custom feature module

Before we show the overridden status however, let's take a look at the actual custom feature module code that we've saved. You'll recall that we added a new folder for our Fire Department content type feature to our `/sites/all/modules/custom` folder. If you look inside the feature module's folder you'll see the following files that are the same as the constructs of a Drupal module:

- `fire_department_content_type.features.field.inc`
- `fire_department_content_type.features.inc`
- `fire_department_content_type.features.taxonomy.inc`
- `fire_department_content_type.info`
- `fire_department_content_type.module`

Anyone familiar with Drupal modules should see here that this is indeed a Drupal module with `info`, `.module`, and `.inc` files. If you inspect the `.info` file in an editor you'll see the following code (this is an excerpt):

```
name = Fire Department Content Type
description = This feature packages up our Fire Department Content
type configuration
core = 7.x
package = Content Types
version = 7.x-1.0
project = fire_department_content_type
dependencies[] = features
```

The brunt of our module is in the `fire_department_content_type.features.field.inc` file. This file contains all of our content type's fields defined as a series of the `$fields` array (see the following excerpt of code):

```
/**
 * @file
 * fire_department_content_type.features.field.inc
 */

/**
 * Implements hook_field_default_fields().
 */
function fire_department_content_type_field_default_fields() {
  $fields = array();

  // Exported field: 'node-fire_department-body'.
  $fields['node-fire_department-body'] = array(
    'field_config' => array(
```

```

'active'=>'1',
'cardinality'=>'1',
'deleted'=>'0',
'entity_types'=>array(
  0 =>'node',
),
'field_name'=>'body',
'foreign_keys'=>array(
  'format'=>array(
    'columns'=>array(
      'format'=>'format',
    ),
  ),
'table'=>'filter_format'

```

If you view the `taxonomy.inc` file you'll see two arrays that return the vocabs which we're referencing via the term references of our content type. As you can see, this module has packaged up our entire content type configuration. It's beyond the scope of this book to get into more detail about the actual module files, but you can see how powerful this can be. If you are a module developer you could actually add code to the specific feature module's files to extend and expand your content type directly from the code. This would then be synced to your feature module codebase. Generally, you do not use this method for tweaking your feature but you do have access to the code and can make tweaks to the feature code. What we'll be doing is overriding our feature from the content type configuration level.

Additionally, if you load your module's admin screen on your site and scroll down until you see the new package called **Content Types**, you'll see your feature module enabled here on the modules admin screen:

▼ CONTENT TYPES				
ENABLED	NAME	VERSION	DESCRIPTION	OPERATIONS
<input checked="" type="checkbox"/>	Fire Department Content Type	7.x-1.0	This feature packages up our Fire Department Content type configuration Requires: Features (enabled), Field SQL storage (enabled), Field (enabled), Link (enabled), Node (enabled), Node Reference (enabled), References (enabled), Options (enabled), Number (enabled), Taxonomy (enabled), Text (enabled)	

If you disable the **Features** module here, it will also disable the feature from your **Features** admin screen. The best practice dictates that you should first disable a **Features** module via the **Features** admin screen. This will then disable the module from the modules admin screen.

Overriding your feature

Our feature is now in **Default** status. This means it matches the module codebase. But what will happen if we need to add a field or tweak a field in our content type? This is the whole point of using **Features**. The **Features** module will detect this change for us and let us know that our module is overridden. Let's try it.

First let's make a change to our Fire Department content type. Load and edit your content type by going to **Structure | Content Types | Fire Department**. We're going to make a few changes, listed as follows:

1. First change the **Preview before Submitting** setting to **Disabled**.
2. Under **Publishing options** uncheck the **Promoted to Front Page** checkbox.
3. Save your content type.
4. Now go to **Manage Fields**. Add a brand new field called **Fire Department Image** with a field type of **Image**. On the image field settings leave all settings at default and save settings.
5. Now load your **Manage Display** and tweak the **Website field format** to be **URL**, as a link. We're changing the display formatter for this field here.
6. Save your content type.
7. You have now tweaked some content type general configuration and you've added a new field to the type.
8. Now go to your **Features** admin screen at **Structure | Features**. You should see that your feature is now showing as **Overridden**.

The screenshot shows the 'Structure | Features' page in Drupal. On the left, there's a sidebar with 'Content Types' and 'Feeds' sections. The main area displays a table of features. The first feature, 'Fire Department Content Type', is marked with a checkmark and has a description: 'This feature packages up our Fire Department Content type configuration'. Its signature is '7.x-1.0'. The 'STATE' column shows 'Unavailable' and 'Overridden' (in a grey box). The 'ACTIONS' column has a 'Recreate' link. At the bottom of the table is a 'Save settings' button.

FEATURE	SIGNATURE	STATE	ACTIONS
<input checked="" type="checkbox"/> Fire Department Content Type This feature packages up our Fire Department Content type configuration	Unavailable 7.x-1.0	Overridden	Recreate

Save settings

The Diff module

Before we inspect our override let's install the **Diff** module. The **Diff** module will be helpful to us as we inspect our feature override. This module will help to show the differences between our original feature codebase and the new tweaks we've made to it via our content type changes. This module is a huge helper module, especially when you have multiple developers working on a site at the same time and making configuration changes to content types and other items that are being synced with features.

To install **Diff** get the latest module version from <http://drupal.org/project/diff>.

Install it as you would any contributed module. Go ahead and enable the **Diff** module and save your module configuration:

OTHER			
ENABLED	NAME	VERSION	DESCRIPTION
<input checked="" type="checkbox"/>	Diff	7.x-2.0	Show difference between node revisions.

Reviewing the override

Now load your **Features** admin screen again and this time click on the **Overridden** link. The Fire Department feature configuration screen will load and you should see a **Overridden** status next to your fields. This is because we've made display formatter changes to our content type. We've also added a new field.

Fire Department Content Type

VIEW REVIEW OVERRIDES RECREATE

Home » Features

Fire Department Content Type

This feature packages up our Fire Department Content type configuration

DEPENDENCY	STATUS
Features	Enabled
Field SQL storage	Enabled
Link	Enabled
Node	Enabled
Node Reference	Enabled
Number	Enabled
Taxonomy	Enabled
Text	Enabled

FEATURES_API

DEFAULT

api:1

☐

FIELDS

OVERIDDEN

node-fire_department-body
node-fire_department-field_fire_active_c
node-fire_department-field_fire_active_fire_paid_p
node-fire_department-field_fire_active_volunt
node-fire_department-field_fire_county
node-fire_department-field_fire_department_type
node-fire_department-field_fire_hq_addr2
node-fire_department-field_fire_hq_address_1
node-fire_department-field_fire_hq_city
node-fire_department-field_fire_hq_fax
node-fire_department-field_fire_hq_phone
node-fire_department-field_fire_hq_state
node-fire_department-field_fire_hq_zip
node-fire_department-field_fire_mail_addr1
node-fire_department-field_fire_mail_addr2
node-fire_department-field_fire_mail_city
node-fire_department-field_fire_mail_po_box
node-fire_department-field_fire_mail_state
node-fire_department-field_fire_mail_zip
node-fire_department-field_fire_non_fire_civilian
node-fire_department-field_fire_non_fire_volunteer
node-fire_department-field_fire_number_of_stations
node-fire_department-field_fire_organization_type
node-fire_department-field_fire_primary_agency
node-fire_department-field_fire_website

CONTENT TYPES

DEFAULT

fire_department

TAXONOMY

DEFAULT

county fire_department_type

Revert components

Now we can review our overrides with our newly installed **Diff** module by clicking on either the **Overridden** status link next to the fields, or clicking on the **Review Overrides** tab. Click on the **Review Overrides** tab.

Here you will see the two changes we made outlined in the diff. The first is the new image field and the second is the format type of `link_url`. If you make a content type configuration change such as promoting the content to the front page or disabling preview settings, these specific configurations will not show up in the diff. The diff will only show you changes to the fields you've added in our specific example here. The changes are in the right column and our existing code is on the left.

DEFAULT	OVERRIDES
INFO	
Line 2 dependencies[] = features dependencies[] = field_sql_storage	Line 2 dependencies[] = features dependencies[] = field_sql_storage + dependencies[] = image
dependencies[] = link dependencies[] = node	dependencies[] = link dependencies[] = node
Line 15 features[field][] = node-fire_department-field_fire_active_volunt features[field][] = node-fire_department-field_fire_county	Line 16 features[field][] = node-fire_department-field_fire_active_volunt features[field][] = node-fire_department-field_fire_county + features[field][] = node-fire_department-field_fire_department_image
features[field][] = node-fire_department-field_fire_department_type features[field][] = node-fire_department-field_fire_hq_addr2	features[field][] = node-fire_department-field_fire_department_type features[field][] = node-fire_department-field_fire_hq_addr2
FIELD	
Line 1211 'label' => 'above', 'module' => 'link', - 'type' => 'link_default', 'weight' => '18',)	Line 1211 'label' => 'above', 'module' => 'link', + 'type' => 'link_url', 'weight' => '18',)

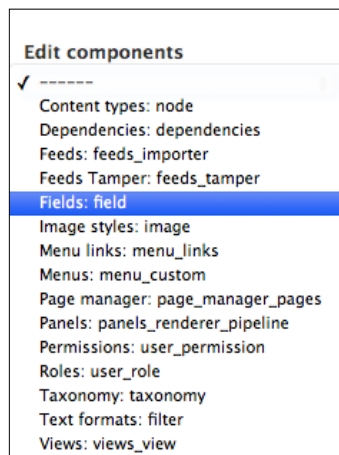
Now we need to recreate our feature module so we can capture these changes and get back to our **Default** synced status.

Recreating the feature

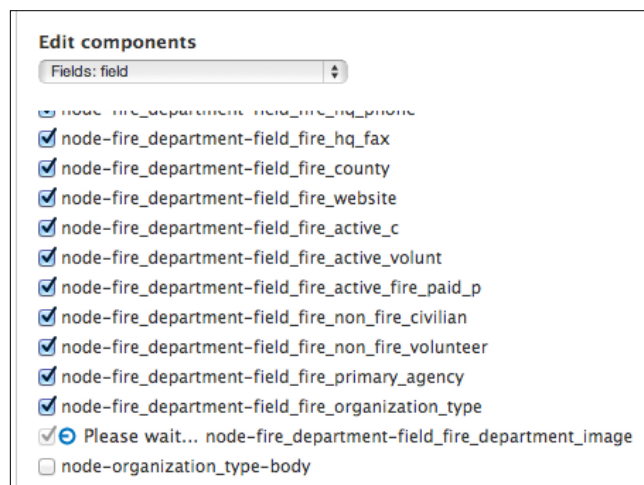
Now that we have reviewed our overrides we can recreate our feature, add the new field to it, and increment its version number. The format type tweak making the website link a `link_url` will be captured automatically when we recreate our feature. To do this, carry out the following steps:

1. Click on the **Recreate** tab.

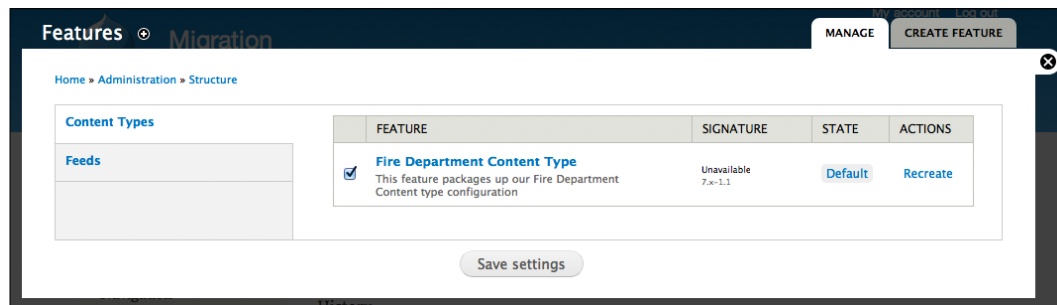
2. Your feature module's config screen will load.
3. It's very important to change the version number of your module here. You can always view the version of a feature module by looking at its `.info` file. We're going to change the version number to `7.x-1.1` here.
4. Now in the **Edit components** drop-down box you need to locate the new Image field you've added. First select **Fields: field** as the type of component you want to search for and add it.



5. Now scroll down in the fields list until you see all of your checked and selected fire department content type fields. Notice that the last field in the list is unchecked. This is the new image field. Check it to add it.



6. Once you have added the field by checking its box, now you can click on the **Download feature** button to download your latest version of the module files.
7. Extract the .tar file and replace all of your existing module's files with this new version.
8. Once you replace the files reload your main **Features** admin page via **Structure | Features**. Your feature should have **Default** status and it should show the **7.x-1.1** version.



That is the Features process in a nutshell. Basically once you start using Features you'll be able to keep a very close eye on any content type, importer, or other site configuration change. Features will catch these changes for you and inform you that your **Features** module is in an overridden state. You'll then review the overrides, add them to your feature, and then recreate your feature.

Reverting features

There will be times when you get code replaced in your local version of your site by another developer's versions of the files. This can happen if you're using a repository to store code and update from. So if the other developer makes a configuration change on his/her version of the site and then commits this new feature module code, you'll then update your site's version with this new code. The process will be similar to recreating a feature.

When you get new code files with a change, your feature will show **Needs Review** status. You can click on the **Needs Review** link and review the changes. Now, your feature will have **Overridden** status just like it was before when you made your own local site changes.

In this case you may want to revert your code to the latest code in the codebase, as it has been updated from SVN or Git for example. This way you can update your local site with the other developer's code. So to do this, just click on the **Revert** button at the bottom of your Feature module's configuration screen. This will revert your code to the latest code files that you've pulled in your repository update and to your local. This can be one of the confusing parts of using the **Features** module as a whole. But just remember that this process will revert your local site's configuration to match the configuration defined in the new version of the **Features** module.

The Feed Importer feature

Now that we have created a feature for our content type and have practiced using the Features module let's go ahead and create a new Feature to package up our Feed importer configuration for our Fire Department content.

To do this follow these steps:

1. Go to **Structure | Features** to load the main **Features** administration screen.
2. Click on the **Create Feature** tab.
3. Give your feature a name: `Fire Department Importer`.
4. In the **Package** field give it a package name: `Importers`.

5. From the **Edit components** select box, choose **Feeds: feeds_importer**:

The screenshot shows the 'Edit components' form for the 'Feeds: feeds_importer' feature. The form is divided into several sections: 'Name', 'Description', 'Package', 'Version', and 'URL of update XML'. The 'Name' section has a text input field containing 'Fire Department Importer' and a 'Machine name' field containing 'fire_department_importer' with an '[Edit]' link. The 'Description' section has a text input field containing 'This is the feature for our feeds importer for the Fire Dept conten' and a prompt to 'Provide a short description of what users should expect when they enable your feature.' The 'Package' section has a dropdown menu set to 'Importers' and a prompt to 'Organize your features in groups.' The 'Version' section has a text input field containing '7.x-1.0' and examples '7.x-1.0, 7.x-1.0-beta1'. The 'URL of update XML' section has a text input field and an example 'http://mywebsite.com/fserver'. At the bottom, there is an 'Edit components' select box with a list of components: 'Content types: node', 'Dependencies: dependencies', 'Feeds: feeds_importer' (highlighted), 'Feeds Tamper: feeds_tamper', 'Fields: field', 'Image styles: image', and 'Menu links: menu_links'. To the right of the select box is a 'Download feature' button.

Name
Fire Department Importer Machine name: fire_department_importer [Edit]
Example: Image gallery (Do not begin name with numbers.)

Description
This is the feature for our feeds importer for the Fire Dept conten
Provide a short description of what users should expect when they enable your feature.

Package
Importers
Organize your features in groups.

Version
7.x-1.0
Examples: 7.x-1.0, 7.x-1.0-beta1

URL of update XML

Example: http://mywebsite.com/fserver

Edit components
✓ -----
Content types: node
Dependencies: dependencies
Feeds: feeds_importer
Feeds Tamper: feeds_tamper
Fields: field
Image styles: image
Menu links: menu_links

Download feature

6. Check the checkbox next to the importer you want to package up; in this case it's `fire_department_importer`:

7. Click on the **Download feature** button to download.
8. Extract the feature module to your `/sites/all/custom` directory.
9. Refresh your **Features** admin screen and you should see your new feature module in the new **Importers** tabbed package. Enable it and it should be set to the **Default** status:

FEATURE	SIGNATURE	STATE	ACTIONS
<input checked="" type="checkbox"/> Fire Department Importer This is the feature for our feeds importer for the Fire Dept content type	Unavailable 7.x-1.0	Default	Recreate

Migrating your feature to another Drupal site

So far in this chapter we've shown how features can help you to manage changes and tweaks to your site's configuration. We've focused on how to do this using a content type and a feed importer. Another powerful feature of the **Features** module is that, as the module packages your configuration changes as code in the form of a custom module, you can easily move your module to another Drupal site that is also running the **Features** module. Then on this new site you can enable your module. When you enable your module, the module will create the content type and its associated components including taxonomy vocabs, associated **Views** and **Panels**, and more.

One note here — you need to make sure that the site you will be implementing the feature module on contains all the dependent modules that your feature module requires. In our case, the majority of the modules are part of Drupal core so we should be able to move our module and implement our content type without too much difficulty. This is a very easy method of migrating content types and configuration from one Drupal site to another. This is the perfect option for when you need to move your configuration from your local development site to a staging site, and then to a production site. You can develop completely locally, and rarely have to tweak any configuration on your staging or production site as you can simply enable your feature module.

To move your module simply copy the specific feature module folder and move it to the site you want to enable it on. You can do this via SFTP or via Git or SVN if you are using a repository to store your code. Deploy the module to the site you want to enable it on. Once the module is on the site you can load the Features administration screen on the site in question and enable your feature module. That's all there is to it.

Summary

In this chapter we successfully installed the **Features** module and built a custom feature that packages our entire Fire Department content type configuration. We showed how to download and extract our module code, and then enable our custom feature and place it into **Default** status.

We added a new field and tweaked some of our content type's existing configuration to test our new feature module and place it into an overridden state. Then we recreated our feature module with the new components and updated configuration and replaced our existing module files with the new ones.

Finally we took our feature module and deployed it to another Drupal website to show that we can easily take our content type and its configuration from our local development site to another Drupal site.

In the next chapter we're going to look at another powerful module, **Migrate**, which can be used to migrate content from earlier versions of a Drupal site (Drupal 6 to a Drupal 7 website); or to migrate content from another content management system.

7

Migration Using the Migrate Module

So far we've focused on using the **Feeds** module to run our migration process. This module and its suite of helper and associate modules, including **Features**, work well and give us a lot of flexibility in our migration approach. There are other contributed modules for Drupal 7 that we should acknowledge so that we're aware that there are other tools we can add to our migration toolbox.

In this chapter we're going to use the **Migrate** module. The **Migrate** module can be used to migrate content into the Drupal framework from other content management systems, similarly to our usage of the **Feeds** module. Migrate can also be used to migrate content from a Drupal 6 to Drupal 7 site during an upgrade process (we'll look at this process in more detail in *Chapter 8, Migrating Content from Earlier Drupal Versions*), or to migrate content from another Drupal 7 site to a new site. This module provides mechanisms for creating nodes, users, terms, comments, and other Drupal entities from some external source.

The module is a large scale API that provides a framework for you to build your own migration process using its functionality. In order to use the module effectively you'll need to learn how to build a Drupal module and also be familiar with PHP and object-oriented programming. The **Migrate** module is basically a tool for Drupal developers to use to build their own migration process to get source data into Drupal.

We're not going to delve into the development side of using Migrate. In this chapter we're going to use the **Migrate Example** module to show how the process runs from the Drupal perspective once you've built your own migration module to handle your own source content. If you want to use the module for your own migration processes you can look closely at the module example files that Migrate provides and these will help you to write your own migration module.

Migrate can be used via its administration user interface in Drupal and is also nicely integrated with the Drush module so you can use Migrate via the command line if that's your method of development.

These are the topics we'll be covering in this chapter:

- Installing the **Migrate** module
- Configuring the **Migrate Example** module
- Running a migration process using Migrate to create nodes
- Creating terms and users using Migrate

Migrate module installation and configuration

Let's get started. The first thing we need to do is download the **Migrate** module. The **Migrate** module's project page is located at <http://drupal.org/project/migrate>. The current version is 7.x-2.4. The module's project page provides a detailed description of the module and links to various resources that will help you to use the module.

Installing Migrate

Download the latest 7.x-2.4. version and install it into your `/sites/all/modules` directory as you would any contributed module. Migrate will work best and to your advantage on a site that has some content types and taxonomies implemented already. It also comes with example source data and content types, terms, and other Drupal entities that you can create on your site by way of example. We're going to use the example module to migrate content into our Fire Department site.

Once installed, browse to your module's administration page and you should see the **Migrate** module under the **Development** package. Migrate comes with a **Migrate Example** module that provides sample data and content for migration practice. Go ahead and enable the example module as well. Enable the Migrate modules and save your module's configuration screen.

DEVELOPMENT			
ENABLED	NAME	VERSION	DESCRIPTION
<input checked="" type="checkbox"/>	Migrate	7.x-2.4	Import content from external sources Required by: Migrate Example (disabled), migrate_example_baseball (disabled), Migrate example - Oracle (disabled), Migrate UI (disabled)
<input checked="" type="checkbox"/>	Migrate Example	7.x-2.4	Example migration data. Requires: Taxonomy (enabled), Options (enabled), Field (enabled), Field SQL storage (enabled), Image (enabled), File (enabled), Comment (enabled), Text (enabled), Migrate (disabled), List (enabled), Number (enabled)
<input checked="" type="checkbox"/>	Migrate UI	7.x-2.4	UI for managing migration processes Requires: Migrate (disabled)

Once you install the module you'll notice a few new example content types provided by the module including Beer, Wine, and Wine Producer. If you go to your **Add Content** screen, you can actually use these content types out of the box to add content. Additionally if you view your taxonomy admin screen you'll see new taxonomies provided by the module, as shown in the following screenshot:

VOCABULARY NAME	OPERATIONS
County	edit vocabulary list terms add terms
Department Type	edit vocabulary list terms add terms
Migrate Example Beer Styles	edit vocabulary list terms add terms
Migrate Example Wine Best With	edit vocabulary list terms add terms
Migrate Example Wine Regions	edit vocabulary list terms add terms
Migrate Example Wine Varieties	edit vocabulary list terms add terms
Tags	edit vocabulary list terms add terms
Save	

You can add terms to these vocabs. The **Migrate Example** module creates these content types and vocabs automatically in order to contain the imported source content that we'll be importing. So a **Migrate** module package basically will create the content types and vocabs for you, much like a custom **Feature** module.

Configuring Migrate

To configure the **Migrate** module, follow these steps:

1. Go to **Content | Migrate**. This will load the main Migrate administration screen. The screen will display a table showing the status of each migration process, the migration item, total rows of data for the item, number of imported and un-imported rows, messages, throughput, and last import time. It's important to realize here that this current table is only showing the example migration process that the **Migrate** module provides us. If you wrote your own migration module, your data would be shown here.
2. Notice that there are migrations for terms including **BeerTerm**, users including **BeerUser**, comments including **WineComment**, and more. Each of these migrations is a link to the specific mapping for the data. All statuses are set to Idle currently, since we have not run a migration yet. The **TOTAL ROWS** field shows how many nodes, users, comments, or terms are going to be imported. Here's what the table should look like:

Migrate

Migration

CONTENT

COMMENTS

MIGRATE

Home » Administration » Content

Migrate

Configure

<input type="checkbox"/>	STATUS	MIGRATION	TOTAL ROWS	IMPORTED	UNIMPORTED	MESSAGES	THROUGHPUT	LAST IMPORTED
<input type="checkbox"/>	Idle	BeerTerm	3	0	3	0	Unknown	
<input type="checkbox"/>	Idle	BeerUser	4	0	4	0	Unknown	
<input type="checkbox"/>	Idle	BeerNode	3	0	3	0	Unknown	
<input type="checkbox"/>	Idle	BeerComment	5	0	5	0	Unknown	
<input type="checkbox"/>	Idle	WinePrep	N/A	N/A	N/A	N/A	Unknown	
<input type="checkbox"/>	Idle	WineBestWith	3	0	3	0	Unknown	
<input type="checkbox"/>	Idle	WineFileBlob	1	0	1	0	Unknown	
<input type="checkbox"/>	Idle	WineFileCopy	1	0	1	0	Unknown	
<input type="checkbox"/>	Idle	WineRegion	12	0	12	0	Unknown	
<input type="checkbox"/>	Idle	WineRole	2	0	2	0	Unknown	
<input type="checkbox"/>	Idle	WineUser	3	0	3	0	Unknown	
<input type="checkbox"/>	Idle	WineVariety	8	0	8	0	Unknown	
<input type="checkbox"/>	Idle	WineProducer	2	0	2	0	Unknown	
<input type="checkbox"/>	Idle	WineProducerMultiXML	2	0	2	0	Unknown	
<input type="checkbox"/>	Idle	WineProducerXML	1	0	1	0	Unknown	
<input type="checkbox"/>	Idle	WineProducerXMLPull	2	0	2	0	Unknown	
<input type="checkbox"/>	Idle	WineWine	2	0	2	0	Unknown	
<input type="checkbox"/>	Idle	WineComment	5	0	5	0	Unknown	
<input type="checkbox"/>	Idle	WineFinish	N/A	N/A	N/A	N/A	Unknown	
<input type="checkbox"/>	Idle	WineTable	3	0	3	0	Unknown	
<input type="checkbox"/>	Idle	WineUpdates	2	0	2	0	Unknown	
<input type="checkbox"/>	Idle	WineUserUpdates	3	0	3	0	Unknown	

3. Scroll down to the bottom of the table and you'll see an **OPERATIONS** section with a drop-down select box and a **Execute** button. Here after you select the import jobs from the Migrate table that you want to run, you can choose the operation to run, whether you want to import, rollback, rollback and import, stop, or reset a migration process. There are detailed descriptions of each process under the **Execute** button.
4. Additionally and similarly to the **Feeds** module you can expand the **OPTIONS** fieldset to see checkboxes for updating previously imported content and for ignoring dependencies during a migration. You can also set a limit on how many items you want to migrate. You should see this at the bottom of the screen:

OPERATIONS

Import

Execute

Choose an operation to run on all migrations selected above:

- Import – Imports all previously unimported records from the source, plus any records marked for update, into destination Drupal objects.
- Rollback – Deletes all Drupal objects created by the migration.
- Rollback and import – Performs the Rollback operation, immediately followed by the Import operation.
- Stop – Cleanly interrupts any import or rollback processes that may currently be running.
- Reset – Sometimes a migration process may fail to stop cleanly, and be left stuck in an Importing or Rolling Back status. Choose Reset to clear the status and permit other operations to proceed.

OPTIONS

☐ Update
Check this box to update all previously-migrated content in addition to importing new content. Leave unchecked to only import new content

☐ Ignore dependencies
Check this box to ignore dependencies when running imports – all migrations will run whether or not their dependent migrations have completed.

Limit to: items Set a limit of how many items to process for each migration, or how long each should run.

5. Now click on the **Configure** tab at the top of the Migration administration screen. This configuration screen will show you all of the custom Migrate classes that your module has implemented (if you wrote a custom migrate module). In our case it shows us the object classes that implement migration handlers provided by the example module. You can use these classes as a model for your own migration. Here you'll see handlers for migrating comments, entity fields, paths, polls, and statistics. These are the custom handlers that the **Migrate** module is providing us for our example. You'll also see the destination entity type (node or entity) that the handler is ultimately going to help map data to.

- Under this **DESTINATION HANDLERS** table, there is a **FIELD HANDLERS** table for handling mapping our data into actual content type fields including **file**, **image**, **node_reference**, **taxonomy_term_reference**, **text**, **user_reference**, and other types of fields. Each has a handler that helps to map data into the field. Here's what the two handler tables look like:

The screenshot shows the 'Migrate configuration' page with tabs for 'CONTENT', 'COMMENTS', and 'MIGRATE'. The 'MIGRATE' tab is active. Below the tabs, there's a breadcrumb trail: 'Home » Administration » Content » Migrate'. A note states: 'In some cases, such as when a handler for a contributed module is implemented in both migrate_extras and the module itself, you may need to disable a particular handler. In this case, you may uncheck the undesired handler below.'

DESTINATION HANDLERS

<input type="checkbox"/>	MODULE	CLASS	DESTINATION TYPES HANDLED
<input checked="" type="checkbox"/>	migrate	MigrateCommentNodeHandler	node
<input checked="" type="checkbox"/>	migrate	MigrateFieldsEntityHandler	entity
<input checked="" type="checkbox"/>	migrate	MigratePathEntityHandler	entity
<input checked="" type="checkbox"/>	migrate	MigratePollEntityHandler	node
<input checked="" type="checkbox"/>	migrate	MigrateStatisticsEntityHandler	node

FIELD HANDLERS

<input type="checkbox"/>	MODULE	CLASS	FIELD TYPES HANDLED
<input checked="" type="checkbox"/>	migrate	MigrateFileFieldHandler	file
<input checked="" type="checkbox"/>	migrate	MigrateImageFieldHandler	image
<input checked="" type="checkbox"/>	migrate	MigrateNodeReferenceFieldHandler	node_reference
<input checked="" type="checkbox"/>	migrate	MigrateTaxonomyTermReferenceFieldHandler	taxonomy_term_reference
<input checked="" type="checkbox"/>	migrate	MigrateTextFieldHandler	text, text_long, text_with_summary
<input checked="" type="checkbox"/>	migrate	MigrateUserReferenceFieldHandler	user_reference
<input checked="" type="checkbox"/>	migrate	MigrateValueFieldHandler	value, list, list_boolean, list_integer, list_float, list_text, number_integer, number_decimal, number_float

At the bottom of the configuration page is a 'Save handler statuses' button.

You should begin to see how powerful this module can be. You can also choose to disable specific handlers here so that those specific mappings and fields will not be handled during the migration process. This provides much flexibility in how you want to configure your migration process.

Viewing a specific migration mapping

Let's now view the details of a specific migration mapping. To do this you can click on any of the Migration item links in the main Migration module table at **Content | Migration**. For example, let's first look at the BeerTerm migration by clicking on its link.

When you view the details for a migration you'll see a tabbed table load showing you the metadata about the specific migration. Here we see the following overview data specified: product owner, implementor, system of record for the migration, and a description of what's being migrated. In this case it's Beer styles from a source database into taxonomy terms in the Drupal site. This is the overview of the migration:

BeerTerm

Home » Administration » Content » Migrate

Overview

Destination

8 mapped.

Source

5 mapped.

Mapping: Done

By priority: 3 OK.

Mapping: DNM

By priority: 5 OK.

Mapping: Client Issues

By priority: 1 Medium.

Product Owner

Liz Taster <ltaster@example.com>

Implementor

Larry Brewer <lbrewer@example.com>

System of record:

Source data

Description:

Migrate styles from the source database to taxonomy terms

Now click on the **Destination** tab. Destination shows that eight data items have been mapped. What this shows is the machine name of the destination field and a description of the field. So, for example, the term ID has a machine name **tid**. This corresponds to the primary key or unique identifier in the source. The machine name corresponds to the term name.

Overview

Destination

8 mapped.

Source

5 mapped.

Mapping: Done

By priority: 3 OK.

Mapping: DNM

By priority: 5 OK.

Mapping: Client Issues

By priority: 1 Medium.

These are the fields available in the destination of this migration. The machine names listed here are those available to be used as the first parameter to `$this->addFieldMapping()` in your Migration class constructor. **Unmapped fields are red.**

Type

taxonomy_term (migrate_example_beer_styles)

MACHINE NAME	DESCRIPTION
tid (PK)	Term: Existing term ID
name	Term: Name
description	Term: Description
parent	Term: Parent (by Drupal term ID)
parent_name	Term: Parent (by name)
format	Term: Format
weight	Term: Weight
path	Node: Path alias

Now click on the **Source** tab. The **Source** tab shows all the fields and their machine names that are coming from the source import file. So here is our original data from our old website. In this case the machine name for our term ID is style (PK).

Home » Administration » Content » Migrate

Overview

Destination
8 mapped.

Source
5 mapped.

Mapping: Done
By priority: 3 OK.

Mapping: DNM
By priority: 5 OK.

Mapping: Client Issues
By priority: 1 Medium.

These are the fields available from the source of this migration. The machine names listed here are those available to be used as the second parameter to `$this->addFieldMapping()` in your Migration class constructor. **Unmapped fields are red.**

Query

```
SELECT met.style AS style, met.details AS details, met.style_parent AS style_parent, met.region AS region, met.hoppiness AS hoppiness
FROM
{migrate_example_beer_topic} met
ORDER BY style_parent ASC
```

MACHINE NAME	DESCRIPTION
style (PK)	met.style
details	met.details
style_parent	met.style_parent
region	met.region
hoppiness	met.hoppiness

The next three tabs show various stages of mapping:

- The **Mapping: Done** tab shows all the field mappings currently implemented in the handler. Here we'll see the destination field in Drupal, the **Source** field from the old database, a description, and the priority. For example, here we'll be mapping the original style of the beer to the term name in Drupal.

Home » Administration » Content » Migrate

Overview

Destination
8 mapped.

Source
5 mapped.

Mapping: Done
By priority: 3 OK.

Mapping: DNM
By priority: 5 OK.

Mapping: Client Issues
By priority: 1 Medium.

DESTINATION	SOURCE	DEFAULT	DESCRIPTION	PRIORITY
name	style			OK
description	details			OK
parent_name	style_parent		The incoming style_parent field is the name of the term parent	OK

- The **Mapping: DNM** (Do not map) tab shows any source data that we're not going to map to Drupal fields or associated entities. In this case Drupal will auto assign a format, weight, parent, and path as we're not mapping those:

Home » Administration » Content » Migrate

Overview

Destination
8 mapped.

Source
5 mapped.

Mapping: Done
By priority: 3 OK.

Mapping: DNM
By priority: 5 OK.

Mapping: Client Issues
By priority: 1 Medium.

DESTINATION	SOURCE	DEFAULT	DESCRIPTION	PRIORITY
	hoppiness		This info will not be maintained in Drupal	OK
format				OK
weight				OK
parent				OK
path				OK

- **Mapping: Client Issues** shows any source data that we're not sure how to handle during the mapping yet or that may have other issues.

Home » Administration » Content » Migrate

Overview

Destination
8 mapped.

Source
5 mapped.

Mapping: Done
By priority: 3 OK.

Mapping: DNM
By priority: 5 OK.

Mapping: Client Issues
By priority: 1 Medium.

DESTINATION	SOURCE	DEFAULT	DESCRIPTION	PRIORITY
	region		Will a field be added to the vocabulary for this?	Medium (#770064)

Running the migration

We're now ready to run our migration using the **Migrate** module. We've inspected our mappings via our configuration and can navigate back to the main Migrate administration screen. Select one or more entities to migrate using the checkboxes. For the purpose of this demo I'll select all boxes. Now scroll down to the bottom of the admin screen and confirm that the **Import** option is selected.

<input checked="" type="checkbox"/>	Idle	WineVariety	8	0	8	0	Unknown
<input checked="" type="checkbox"/>	Idle	WineProducer	2	0	2	0	Unknown
<input checked="" type="checkbox"/>	Idle	WineProducerMultiXML	2	0	2	0	Unknown
<input checked="" type="checkbox"/>	Idle	WineProducerXML	1	0	1	0	Unknown
<input checked="" type="checkbox"/>	Idle	WineProducerXMLPull	2	0	2	0	Unknown
<input checked="" type="checkbox"/>	Idle	WineWine	2	0	2	0	Unknown
<input checked="" type="checkbox"/>	Idle	WineComment	5	0	5	0	Unknown
<input checked="" type="checkbox"/>	Idle	WineFinish	N/A	N/A	N/A	N/A	Unknown
<input checked="" type="checkbox"/>	Idle	WineTable	3	0	3	0	Unknown
<input checked="" type="checkbox"/>	Idle	WineUpdates	2	0	2	0	Unknown
<input checked="" type="checkbox"/>	Idle	WineUserUpdates	3	0	3	0	Unknown
<input checked="" type="checkbox"/>	Idle	WineVarietyUpdates	8	0	8	0	Unknown
<input checked="" type="checkbox"/>	Idle	WineCommentUpdates	5	0	5	0	Unknown

OPERATIONS

Import

Execute

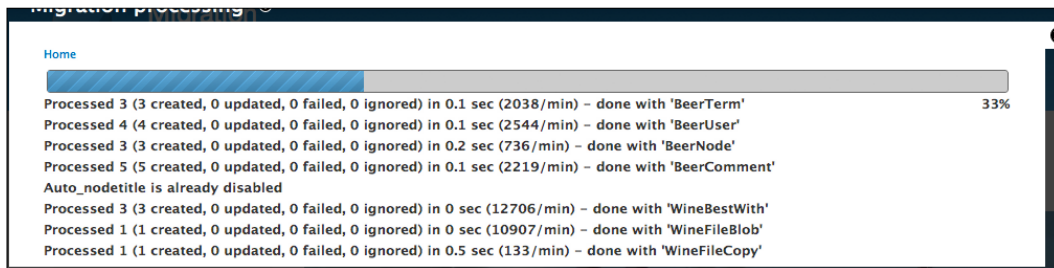
Choose an operation to run on all migrations selected above:

- Import – Imports all previously unimported records from the source, plus any records marked for update, into destination Drupal objects.
- Rollback – Deletes all Drupal objects created by the migration.
- Rollback and Import – Performs the Rollback operation, immediately followed by the Import operation.
- Stop – Cleanly interrupts any import or rollback processes that may currently be running.
- Reset – Sometimes a migration process may fail to stop cleanly, and be left stuck in an Importing or Rolling Back status. Choose Reset to clear the status and permit other operations to proceed.

► **OPTIONS**

Click on the **Execute** button to run the migration and import the content.

You will see a progress bar showing the migration process running:



When the migration process completes you'll see a success table appear showing all of the successful migrations.

Also you'll notice that your migration table shows the last imported date and time and also has a throughput statistic showing for each entity.

Home » Administration » Content

Migrate Configure

- Processed 3 (3 created, 0 updated, 0 failed, 0 ignored) in 0.1 sec (2038/min) – done with 'BeerTerm'
- Processed 4 (4 created, 0 updated, 0 failed, 0 ignored) in 0.1 sec (2544/min) – done with 'BeerUser'
- Processed 3 (3 created, 0 updated, 0 failed, 0 ignored) in 0.2 sec (736/min) – done with 'BeerNode'
- Processed 5 (5 created, 0 updated, 0 failed, 0 ignored) in 0.1 sec (2219/min) – done with 'BeerComment'
- Auto_nodetitle is already disabled
- Processed 3 (3 created, 0 updated, 0 failed, 0 ignored) in 0 sec (12706/min) – done with 'WineBestWith'
- Processed 1 (1 created, 0 updated, 0 failed, 0 ignored) in 0 sec (10907/min) – done with 'WineFileBlob'
- Processed 1 (1 created, 0 updated, 0 failed, 0 ignored) in 0.5 sec (133/min) – done with 'WineFileCopy'
- Processed 12 (12 created, 0 updated, 0 failed, 0 ignored) in 0.1 sec (9528/min) – done with 'WineRegion'
- Processed 2 (0 created, 0 updated, 0 failed, 0 ignored) in 0 sec (4406/min) – done with 'WineRole'
- Processed 3 (3 created, 0 updated, 0 failed, 0 ignored) in 0 sec (3647/min) – done with 'WineUser'
- Processed 8 (8 created, 0 updated, 0 failed, 0 ignored) in 0 sec (13076/min) – done with 'WineVariety'
- Processed 2 (2 created, 0 updated, 0 failed, 0 ignored) in 0 sec (7142/min) – done with 'WineProducer'
- Processed 2 (2 created, 0 updated, 0 failed, 0 ignored) in 0 sec (8387/min) – done with 'WineProducerMultiXML'
- Processed 1 (1 created, 0 updated, 0 failed, 0 ignored) in 0 sec (2019/min) – done with 'WineProducerXML'
- Processed 2 (2 created, 0 updated, 0 failed, 0 ignored) in 0 sec (7999/min) – done with 'WineProducerXMLPull'
- Processed 2 (2 created, 0 updated, 0 failed, 0 ignored) in 4.7 sec (26/min) – done with 'WineWine'
- Processed 5 (5 created, 0 updated, 0 failed, 0 ignored) in 0.1 sec (5451/min) – done with 'WineComment'
- auto_nodetitle was not originally enabled
- Processed 3 (3 created, 0 updated, 0 failed, 0 ignored) in 0.1 sec (3056/min) – done with 'WineTable'
- Processed 2 (0 created, 2 updated, 0 failed, 0 ignored) in 0.1 sec (2102/min) – done with 'WineUpdates'
- Processed 3 (0 created, 3 updated, 0 failed, 0 ignored) in 0 sec (5640/min) – done with 'WineUserUpdates'
- Processed 8 (0 created, 8 updated, 0 failed, 0 ignored) in 0.1 sec (9404/min) – done with 'WineVarietyUpdates'
- Processed 5 (0 created, 5 updated, 0 failed, 0 ignored) in 0 sec (7444/min) – done with 'WineCommentUpdates'

<input type="checkbox"/>	STATUS	MIGRATION	TOTAL ROWS	IMPORTED	UNIMPORTED	MESSAGES	THROUGHPUT	LAST IMPORTED
<input type="checkbox"/>	Idle	BeerTerm	3	3	0	0	2000/min	2012-09-08 22:29:48
<input type="checkbox"/>	Idle	BeerUser	4	4	0	1	2400/min	2012-09-08 22:29:48
<input type="checkbox"/>	Idle	BeerNode	3	3	0	0	750/min	2012-09-08 22:29:48
<input type="checkbox"/>	Idle	BeerComment	5	5	0	0	1667/min	2012-09-08 22:29:48
<input type="checkbox"/>	Idle	WinePrep	N/A	N/A	N/A	N/A	Unknown	2012-09-08 22:29:48
<input type="checkbox"/>	Idle	WineBestWith	3	3	0	0	18000/min	2012-09-08 22:29:48
<input type="checkbox"/>	Idle	WineFileBlob	1	1	0	0	6000/min	2012-09-08 22:29:48

You can now browse to your content administration to verify that the new Beer nodes have been created.

<input type="checkbox"/>	TITLE	TYPE	AUTHOR	STATUS	UPDATED	OPERATIONS
<input type="checkbox"/>	Boddington new	Beer	alice	published	09/08/2012 - 22:29	edit delete
<input type="checkbox"/>	Miller Lite new	Beer	alice	published	09/08/2012 - 22:29	edit delete
<input type="checkbox"/>	Heineken new	Beer	admin	published	09/08/2012 - 22:29	edit delete

Summary

In this chapter we successfully installed the **Migrate** module and used the **Migrate Example** module to run a migration process, and import nodes, terms, and users into our Drupal site. We looked in detail at the **Migration** module configuration and mapping mechanisms and functionality.

In the next chapter we're going to migrate content from a Drupal 6 site to a Drupal 7 site to demonstrate how you can handle content migration during a Drupal upgrade process.

8

Migrating Content from Earlier Drupal Versions

So far in the book we've showed how to migrate content from another content management system or a website application into Drupal 7. We've used a data model to migrate our content and the **Feeds** module to import our content. Then in *Chapter 7, Migration Using the Migrate Module*, we used the **Migration** module to show another method and model of migrating content to Drupal 7.

In some cases you may have a Drupal 6 site that contains thousands of nodes and files that you want to move into the Drupal 7 environment and architecture. This will be an upgrade process. This process will not only move your node and file content, but will also upgrade your entire Drupal 6 site to Drupal 7. You'll then have a site in a new version of Drupal and reap all the benefits of using Drupal 7. You'll also be in a good position to be ready to upgrade to Drupal 8 when it's officially released.

This chapter will show you how to migrate your Drupal 6 content into Drupal 7 with a focus on content. It's beyond the scope of the book to cover every nuance and detail of a Drupal 6 to 7 upgrade. Some upgrades will be complex based on the contributed modules you have installed and the number of custom modules that need to be ported. Our upgrade example will use a simple Drupal 6 core site with a few contributed modules, including **Content Construction Kit (CCK)**. The site we'll be upgrading contains no custom modules.

We'll be covering the following topics in this chapter:

- Upgrading Drupal 6 to 7
- Using the **Update Status** and **Upgrade Status** helper modules
- Migrating your Drupal 6 fields to Drupal 7
- Running the upgrade
- Migrating specific modules including **FileField** and **ImageField**

Upgrading Drupal 6 to 7

We have a Drupal 6 site that we want to upgrade to Drupal 7 and migrate its content into Drupal 7. This site is similar to the original Drupal 7 Fire Department site we've been using throughout the book. This site has two content types:

- Fire Department
- Organization Type

The Fire Department content type contains about 10 fields that include text, integer, node reference, link, file and image fields. We will use the Fire Department content type to reference the Organization type.

To get our data and content imported into Drupal 6 we used the **Feeds** module, just like we did in our Drupal 7 site. We have already executed the import and we're assuming here that you already have a Drupal 6 site with some content that you can use for these examples (or the code for this chapter, which is in the form of the entire Drupal 6 website).

Here's what our current content list looks like on the Drupal 6 site. It's very similar to the Drupal 7 site we've already built in this book in terms of content.

<input type="checkbox"/>	Title	Type	Author	Status	Operations
<input type="checkbox"/>	Local (includes career combination and volunteer)	Organization Type	admin	published	edit
<input type="checkbox"/>	Federal Government - DOD	Organization Type	admin	published	edit
<input type="checkbox"/>	Auburn Fire Department new	Fire Department	Anonymous	published	edit
<input type="checkbox"/>	Atkinson Fire Department new	Fire Department	Anonymous	published	edit
<input type="checkbox"/>	Audubon Park Volunteer Fire Company new	Fire Department	Anonymous	published	edit
<input type="checkbox"/>	Atlas Fire CO new	Fire Department	Anonymous	published	edit
<input type="checkbox"/>	Atwell Township Volunteer Fire Department	Fire Department	Anonymous	published	edit
<input type="checkbox"/>	Auberry Volunteer Fire Department new	Fire Department	Anonymous	published	edit
<input type="checkbox"/>	Auburn Volunteer Fire Company new	Fire Department	Anonymous	published	edit
<input type="checkbox"/>	Atlanta Volunteer Fire Department	Fire Department	Anonymous	published	edit
<input type="checkbox"/>	Attawaugan Fire District new	Fire Department	Anonymous	published	edit
<input type="checkbox"/>	Auburn Fire Department new	Fire Department	Anonymous	published	edit
<input type="checkbox"/>	Atkinson Fire and Rescue new	Fire Department	Anonymous	published	edit
<input type="checkbox"/>	Audubon Fire Department new	Fire Department	Anonymous	published	edit
<input type="checkbox"/>	Atlantic Volunteer Fire and Rescue Company Inc. new	Fire Department	Anonymous	published	edit
<input type="checkbox"/>	Atwater Fire Department new	Fire Department	Anonymous	published	edit
<input type="checkbox"/>	Auburn NY Fire Department new	Fire Department	Anonymous	published	edit
<input type="checkbox"/>	Atlanta Texas Fire Department new	Fire Department	Anonymous	published	edit
<input type="checkbox"/>	Augusta County Fire & Rescue new	Fire Department	Anonymous	published	edit

Here's an example node from our Drupal 6 site. Notice that it contains an image that's been uploaded using the **ImageField** module and an attached PDF that's been uploaded with the **FileField** module:

Arnold Volunteer Fire Department (Anne Arundel County FD) [View](#) [Edit](#)

Operating in off-line mode.

Fri, 09/14/2012 - 17:25 — Anonymous

Headquarters Address 1:
1505 Ritchie HWY

Number of Stations:
1


Headquarters City:
Arnold


Headquarters State:
MD

Headquarters Zip:
21012

Website:
<http://www.arnoldvfd.com/>

Fire Department Image:



Fire Department PDF:
 [sample-pdf.pdf](#)

We originally imported this content using the **Feeds** module, and our Feed importer in Drupal 6 resembles almost exactly the Feeds importer we built on our Drupal 7 site:

Home > Administer > Site building > Feed importers

Edit importer: Fire Department Importer

Operating in off-line mode.

Basic settings
Attached to: [none] [Settings](#)
Refresh: every 30 min
Import on create

Fetcher [Change](#)
File upload [Settings](#)
Upload content from a local file.

Parser [Change](#)
CSV parser [Settings](#)
Parse data in Comma Separated Value format.

Processor [Change](#)
Node processor [Settings](#) [Mapping](#)
Create and update nodes.

Mapping for Node processor [Help](#)
Define which elements of a single item of a feed (= Sources) map to which content pieces in Drupal (= Targets). Make sure that at least one definition has a *Unique target*. A unique target means that a value for a target can only occur once. E. g. only one item with the URL *http://example.com/story/1* can exist.

Source	Target	Unique target	
guid	GUID	<input checked="" type="checkbox"/>	Remove
fire_dept_name	Title		Remove
hq_addr_2	Headquarters Address 2		Remove
hq_addr_1	Headquarters Address 1		Remove
hq_city	Headquarters City		Remove
hq_state	Headquarters State		Remove
hq_zip	Headquarters Zip		Remove
number_of_stations	Number of Stations		Remove
website	Website (URL)		Remove
Name of source field	Select a target		Add

[Legend](#)

Finally here's our Fire Department content type administration screen, showing you the fields we've added to our content type using the **CCK** module:

Fire Department [Edit](#) [Manage fields](#) [Display fields](#)

[Fire Department Image](#)
[Fire Department PDF](#)
[Headquarters Address 1](#)
[Headquarters Address 2](#)
[Headquarters City](#)
[Headquarters State](#)
[Headquarters Zip](#)
[Number of Stations](#)
[Organization Type](#)
[Website](#)

Operating in off-line mode.

Add fields and groups to the content type, and arrange them on content display and input forms.
 You can add a field to a group by dragging it below and to the right of the group.
 Note: Installing the [Advanced help](#) module will let you access more and better help.

Label	Name	Type	Operations
+ Fire Department Name	Node module form.		
+ Menu settings	Menu module form.		
+ Body	Node module form.		
+ Revision information	Node module form.		
+ Authoring information	Node module form.		
+ Publishing options	Node module form.		
+ Comment settings	Comment module form.		
+ File attachments	Upload module form.		
+ Headquarters Address 1	field_fire_hq_address_1	Text	Configure Remove
+ Number of Stations	field_fire_number_of_stations	Integer	Configure Remove
+ Headquarters Address 2	field_fire_hq_addr2	Text	Configure Remove
+ Headquarters City	field_fire_hq_city	Text	Configure Remove
+ Headquarters State	field_fire_hq_state	Text	Configure Remove
+ Headquarters Zip	field_fire_hq_zip	Text	Configure Remove
+ Organization Type	field_fire_organization_type	Node reference	Configure Remove
+ Website	field_fire_website	Link	Configure Remove
+ Fire Department Image	field_fire_dept_image	File	Configure Remove
+ Fire Department PDF	field_fire_dept_pdf	File	Configure Remove

Besides the **Feeds** module, we have the following modules enabled on our Drupal 6 site:

- **Administration menu**
- **Content Construction Kit (CCK):** This includes the following modules:
 - **FileField**
 - **ImageField**
 - **FileField**
 - **ImageField**
 - **Link**
 - **Node Reference**

- **CTools**
- **Feeds**
- **Job Scheduler**



Before we begin the upgrade process, make sure you run a full backup of your Drupal 6 site directory (all the Drupal 6 code base) and also make a backup of your Drupal 6 site database. This way if anything goes wrong during the upgrade process you can easily revert back to your original site and DB. You can use the **Backup and Migrate** module to run the DB backup or use phpMyAdmin.

Upgrade Status and Upgrade Assist

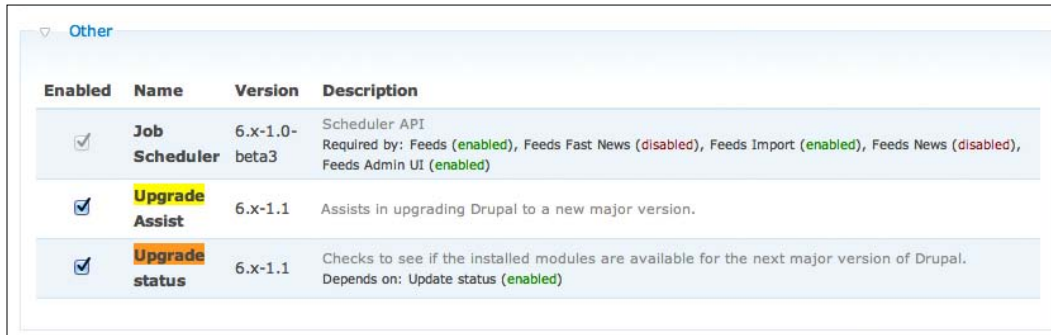
The Drupal 6 to 7 upgrade process can be complex, but there are two helper modules we can use that will outline the entire process for us and give us a helper checklist to use for the entire upgrade and migration process. The Upgrade Status project gives us two modules, **Upgrade Status** and **Upgrade Assist**:

- The **Upgrade Status** module does an inventory for us of our entire core and contributed Drupal 6 modules and shows us whether these modules are available in Drupal 7. This is very helpful information as we will be only upgrading the modules that are available and have either development or stable versions.
- The other information this module will give us is a checklist with step-by-step instructions for our upgrade process. The **Upgrade Assist** module will show a block in our left-hand sidebar that will contain this upgrade checklist. So, we always have it visible during the upgrade.

Let's install the latest version of **Upgrade Assist**:

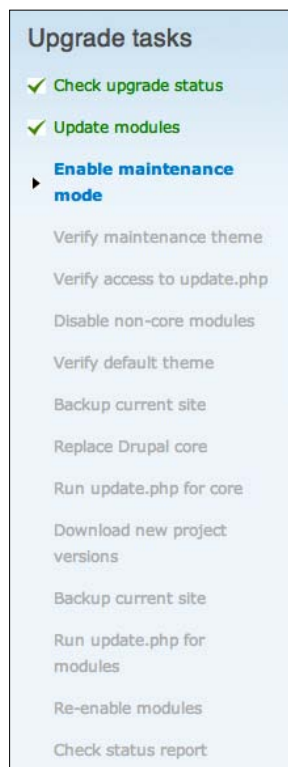
1. First make sure your core Drupal 6 **Upgrade Status** module is enabled on your module's admin screen. Update Status is a required module and dependency of Upgrade Status.
2. Now download the latest Upgrade Status module from its project page: http://drupal.org/project/upgrade_status. The latest version is 6.x-1.1. Install the module as you would do any contributed module in the `/sites/all/modules` directory.

3. Now enable the **Upgrade Assist** and **Upgrade status** modules on your modules admin screen:



Enabled	Name	Version	Description
<input checked="" type="checkbox"/>	Job Scheduler	6.x-1.0-beta3	Scheduler API Required by: Feeds (enabled), Feeds Fast News (disabled), Feeds Import (enabled), Feeds News (disabled), Feeds Admin UI (enabled)
<input checked="" type="checkbox"/>	Upgrade Assist	6.x-1.1	Assists in upgrading Drupal to a new major version.
<input checked="" type="checkbox"/>	Upgrade status	6.x-1.1	Checks to see if the installed modules are available for the next major version of Drupal. Depends on: Update status (enabled)

4. Once you install the module you'll notice a new block in the left-hand sidebar called **Upgrade tasks** with a few tasks in the checklist checked off already. Specifically you'll see the **Check upgrade status** and **Update modules** checkboxes checked:



Upgrade tasks

- ☒ Check upgrade status
- ☒ Update modules
- ☒ Enable maintenance mode
 - Verify maintenance theme
 - Verify access to update.php
 - Disable non-core modules
 - Verify default theme
 - Backup current site
 - Replace Drupal core
 - Run update.php for core
 - Download new project versions
 - Backup current site
 - Run update.php for modules
 - Re-enable modules
 - Check status report

- Each task in the **Upgrade tasks** block will be linked to a task page. If you click on the **Check upgrade status** link you'll launch a **Available updates** screen powered by the **Upgrade status** module. But this screen will have additional information about each of your Drupal 6 modules and core showing you whether there is an available Drupal 7 version of our module. This is a very helpful screen as it will call out any module that does not have an update for Drupal 7. All of our modules have either development or stable versions:

Available updates

ListSettingsUpgrade status

Operating in off-line mode.

Clicking on any of the modules' boxes will expand the area and show you a link to download the new version of the project, as well as read its release notes.

Last checked: 11 sec ago (Check manually)

Target version of Drupal core:
7.x
Select the version of Drupal core you wish to check for project status.
Change

Drupal core

▶ Drupal core 6.26 Available ✓

Modules

▶ Administration menu 6.x-1.8 In development
Recommended version: 7.x-3.0-rc3 (2012-May-17) Download Release notes
Includes: Administration menu

▶ Chaos tool suite (ctools) 6.x-1.9 Available ✓
Recommended version: 7.x-1.2 (2012-Aug-18) Download Release notes
Includes: Chaos tools

▶ Content Construction Kit (CKK) 6.x-2.9 In core ⚠

▶ Feeds 6.x-1.0-beta12 In development
Recommended version: 7.x-2.0-alpha5 (2012-May-28) Download Release notes
Includes: Feeds, Feeds Admin UI, Feeds Import

▶ FileField 6.x-3.10 In core ⚠

▶ ImageField 6.x-3.10 In core ✓

▶ Job Scheduler 6.x-1.0-beta3 In development
Recommended version: 7.x-2.0-alpha3 (2012-May-08) Download Release notes
Includes: Job Scheduler

▶ Link 6.x-2.9 Available ✓

6. Clicking on the **Update modules** link (in the **Upgrade tasks** block) will load your Drupal 6 modules update screen, showing you whether all of your Drupal 6 modules are up-to-date.



One note about the Drupal 6 to 7 upgrade process. It's the best practice to get your all your core and contributed modules updated to their latest 6 versions first before attempting the upgrade to Drupal 7. This will ultimately give you a smoother upgrade process and less prone to error. So just make sure you've upgraded all your Drupal 6 modules to their latest versions first.

7. Now make sure you have run a full backup of your Drupal 6 site and database before moving forward and putting your site in maintenance mode.
8. The next link allows you to enable maintenance mode on your site. This is important if you are running this upgrade on a production level website. You will want to place your site in maintenance mode or offline mode so your site visitors know that you are performing maintenance. Click on this link to enable the **Maintenance** mode.
9. Verify that the **Maintenance** theme allows you to choose a specific theme to use while you are running the upgrade. It's recommended to just keep this set to a basic core theme such as **Garland**.
10. The upgrade assist module will then verify that you have permissions and access as the admin user to your `update.php` script as you'll need to run a database update after you complete the upgrade.

11. Now click on the **Disable non-core modules** link. This will launch a screen that shows you all the contributed modules that you currently have enabled on the site including all your **CCK** widgets, **Feeds** modules, **FileField**, and **ImageField**. You should see the following screen:

Disable non-core modules

Operating in off-line mode.

Modules to disable:

- ☒ admin_menu: Administration menu
- ☒ cck: Content
- ☒ cck: Content Copy
- ☒ cck: Fieldgroup
- ☒ cck: Node Reference
- ☒ cck: Number
- ☒ cck: Option Widgets
- ☒ cck: Text
- ☒ cck: User Reference
- ☒ ctools: Chaos tools
- ☒ feeds: Feeds
- ☒ feeds: Feeds Admin UI
- ☒ feeds: Feeds Import
- ☒ filefield: FileField
- ☒ imagefield: ImageField
- ☒ job_scheduler: Job Scheduler
- ☒ link: Link
- ☒ upgrade_status: Upgrade status

All disabled modules are tracked and may be re-enabled after upgrading Drupal core.

12. Go ahead and leave all of the contributed modules selected and then click on the **Disable** button. One of the reasons we're disabling these contributed modules is that if they are enabled they can cause errors during the upgrade process. Also, in Drupal 7 the **CCK** module is now built into core Drupal so much of the architecture surrounding the **CCK** module and its fields has changed.

13. You should see a message telling you the selected modules have been disabled. You can verify that this worked by visiting your module's administration screen. You'll see the modules have been disabled on this screen as well.

You're now ready to start the actual upgrade. Also disable the core **Update Status** module as you no longer need that module to check your contributed module statuses.

Upgrading Drupal core

We're now going to run our upgrade of Drupal core. To do this click on the **Replace Drupal core** link in the **Upgrade tasks** block.

Replace Drupal core

Operating in off-line mode.

At this point, it is recommended to

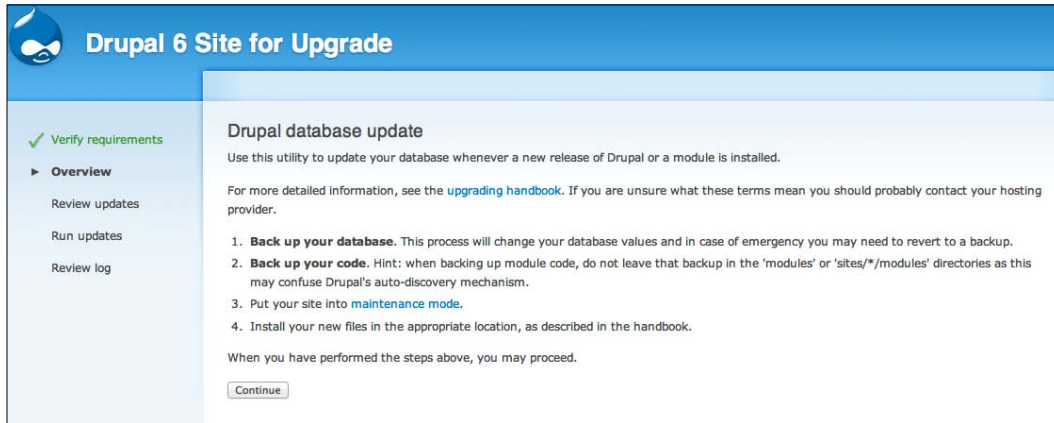
- Create a new virtual host (e.g., <http://d7.migration6.localhost>) on your webserver, pointing to a separate directory.
- Download and extract the new [Drupal core](#) into that directory.
- Copy all existing modules (for the previous version of Drupal core) into the identical locations.
- Copy `settings.php` and the files directory into the identical locations.
- Run <http://d7.migration6.localhost/update.php>.

If anything breaks, restore the current state from the backup you did earlier.

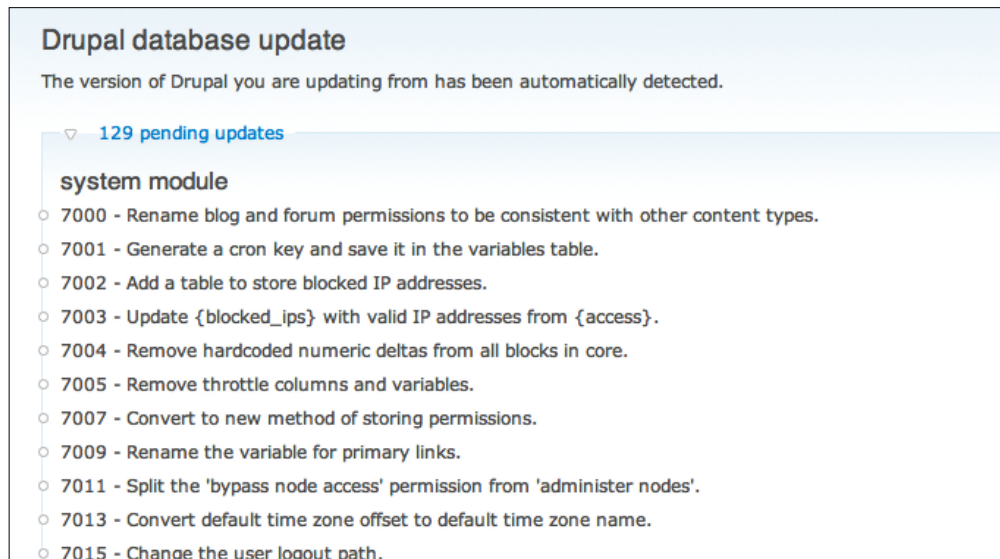
There are recommended steps here but I like to use the instructions in the `UPGRADE.txt` file that Drupal 7 provides:

1. Remove the `default.settings.php` file from your `/sites/default` folder.
2. Remove all of your core files and folders. Do not remove your `/sites` folder. Leave this and your contrib modules in place.
3. Remove your uninstalled contributed modules from `/sites/all/modules` and put them in another folder on your desktop called `contrib_backup`.
4. Now extract your Drupal 7 core and put the core files into your Drupal 6 site directory.
5. Make sure your **settings.php** file is writable.

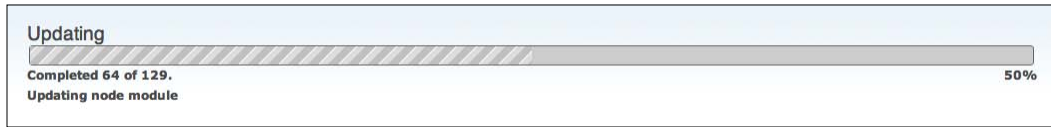
6. Now run your `update.php` script to update your core Drupal site. You should see the following:



7. Once you confirm you've backed up your code and DB, click on the **Continue** button.
8. A **pending updates** screen will load showing you all of the core updates that will run:



9. Click on the **Apply pending updates** button. A progress bar will show while the core upgrade runs.



10. When the upgrade completes you should see a success screen. You can click to return to the site's home page or administration screen. Your site core has been upgraded to Drupal 7.

Drupal database update

- Some user time zones have been emptied and need to be set to the correct values. Use the new [time zone options](#) to choose whether to remind users at login to set the correct time zone.
- The default time zone has been set to *America/Halifax*. Check the [date and time configuration page](#) to configure it correctly.

A new *Plain* text format has been created which will be available to all users. You can configure this text format on the [text format configuration page](#).

Updates were attempted. If you see no failures below, you may proceed happily back to your [site](#). Otherwise, you may need to update your database manually. All errors have been [logged](#).

- [Front page](#)
- [Administration pages](#)

The following updates returned messages

user module

Update #7000

- User passwords rehashed to improve security

Update #7002

- Migrated user time zones

Update #7014

- Renamed the 'post comments without approval' permission to 'skip comment approval'.

system module

Update #7007

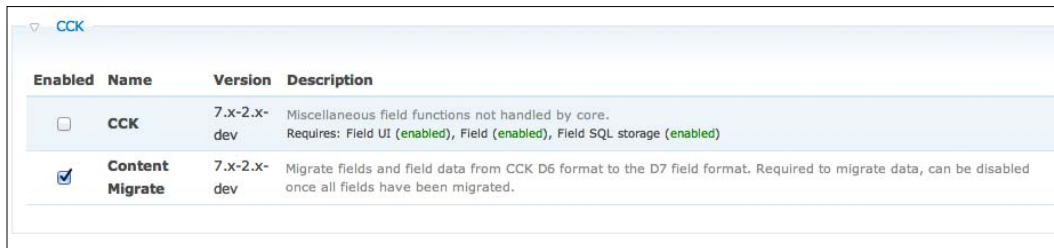
- Inserted into {role_permission} the permissions for role ID 1, Inserted into {role_permission} the permissions for role ID 2

11. If you check your status report you should see you have a Drupal 7 site now.

Migrating Drupal 6 CCK fields and content to Drupal 7

If you load your content admin screen from `/admin/content`, you should see a table showing you all the Fire Department nodes. If you click to inspect one of the nodes you will notice that you will not see any content or data on these nodes. This is because though we ran the core upgrade we did not actually migrate any of our Drupal 6 CCK powered fields yet. We need to now migrate those fields with the help of the **Content Migrate** module. This module is part of Drupal 7's version of CCK. We need to download the Drupal 7 CCK module so that we can use its Content migration path to get our Drupal 6 fields migrated correctly.

1. Download the current dev version of this module from its project page at <http://drupal.org/project/cck>.
2. Install it as you would any contributed module in `/sites/all/modules`.
3. Now enable the **Content Migrate** module:



The screenshot shows the Drupal 7 module manager interface. At the top, there is a tab labeled 'CCK'. Below it is a table with columns: 'Enabled', 'Name', 'Version', and 'Description'. There are two rows in the table. The first row is for the 'CCK' module, which is currently disabled (checkbox is empty). The second row is for the 'Content Migrate' module, which is currently enabled (checkbox is checked).

Enabled	Name	Version	Description
<input type="checkbox"/>	CCK	7.x-2.x-dev	Miscellaneous field functions not handled by core. Requires: Field UI (enabled), Field (enabled), Field SQL storage (enabled)
<input checked="" type="checkbox"/>	Content Migrate	7.x-2.x-dev	Migrate fields and field data from CCK D6 format to the D7 field format. Required to migrate data, can be disabled once all fields have been migrated.

4. Now browse to your **Migrate fields** screen from `/admin/structure/content_migrate`. You should see a similar screen to the following:

Migrate fields

Operating in maintenance mode. [Go online.](#)

Available fields

Fields that have not yet been migrated but are available for migration. Please carefully read the messages next to each field before migrating it to understand changes that might be made.

Field	Field type	Content type(s)	Other information
<input type="checkbox"/> field_fire_hq_addr2	text	<input type="radio"/> Fire Department	Invalid field/widget combination: The field 'field_fire_hq_addr2' in the bundle 'fire_department' is an unlimited length field using a textfield widget, not allowed in D7. The field length will be set to 255.
<input type="checkbox"/> field_fire_hq_address_1	text	<input type="radio"/> Fire Department	Invalid field/widget combination: The field 'field_fire_hq_address_1' in the bundle 'fire_department' is an unlimited length field using a textfield widget, not allowed in D7. The field length will be set to 255.
<input type="checkbox"/> field_fire_hq_city	text	<input type="radio"/> Fire Department	Invalid field/widget combination: The field 'field_fire_hq_city' in the bundle 'fire_department' is an unlimited length field using a textfield widget, not allowed in D7. The field length will be set to 255.
<input type="checkbox"/> field_fire_hq_state	text	<input type="radio"/> Fire Department	Invalid field/widget combination: The field 'field_fire_hq_state' in the bundle 'fire_department' is an unlimited length field using a textfield widget, not allowed in D7. The field length will be set to 255.
<input type="checkbox"/> field_fire_hq_zip	text	<input type="radio"/> Fire Department	Invalid field/widget combination: The field 'field_fire_hq_zip' in the bundle 'fire_department' is an unlimited length field using a textfield widget, not allowed in D7. The field length will be set to 255.
<input type="checkbox"/> field_fire_number_of_stations	number_integer	<input type="radio"/> Fire Department	

Migrate selected fields

Converted fields

Fields that have already been converted. You can choose to roll them back if the conversion did not work correctly. Note that rolling fields back will completely destroy the new field tables. **This operation cannot be undone!**

Field	Field type	Content type(s)	Other information
No fields are already converted.			

Roll back selected fields

- Our available fields for migration currently include the text and integer fields. We'll migrate these first. Our unavailable fields include the image, file (PDF), node reference, and link fields as those are all dependent on contributed modules that we have not upgraded yet.
- Let's go ahead and migrate the core text and integer fields first.

7. Check the checkboxes next to the fields you want to migrate in your **Available fields** section and then click on the **Migrate selected fields** button. You will see a progress bar showing you the migration. Once the migration runs, a report will show up. You can now open up your nodes to confirm that the content migrated. You should see the text and integer data now on each of your nodes.
8. Now go to your modules administration page and enable the File module. File handles all file fields in Drupal 7. So before we migrate our file field (PDF) we need to enable this module. Also enable the Image module so we can migrate our image field.
9. Now refresh your content migration **Migrate fields** screen and you should see two new available fields for your image and PDF. Check the boxes next to those fields and click on the **Migrate selected fields** button.

Migrate fields

Operating in maintenance mode. [Go online.](#)

Available fields

Fields that have not yet been migrated but are available for migration. Please carefully read the messages next to each field before migrating it to understand changes that might be made.

Field	Field type	Content type(s)	Other information
<input checked="" type="checkbox"/> field_fire_dept_image	image	Fire Department	<ul style="list-style-type: none">Changed field type: The 'field_fire_dept_image' field type will be changed from 'filefield' to 'image'.Missing widget: The 'image' widget is not available for the field_fire_dept_image field, it will be set to the default widget.
<input checked="" type="checkbox"/> field_fire_dept_pdf	file	Fire Department	<ul style="list-style-type: none">Changed field type: The 'field_fire_dept_pdf' field type will be changed from 'filefield' to 'file'.

10. Now load the nodes that have images and PDFs attached via these fields and you should see them appear now on the node. You've successfully migrated your file and image content.

Upgrading your contributed modules

The final upgrade process we need to complete involves our contributed modules. This will also allow us to migrate our Link field for our Fire Department website field. Recall that you have moved your Drupal 6 contributed modules into a backup folder. The only modules we're going to worry about upgrading right now are the **Link** and **Node Reference** modules, so that we can successfully complete the migration of our data. Note here that the new 7.x version of the **Node Reference** module is now called **References**. Its project page is at <http://drupal.org/project/references>.

Go ahead and download each of the Drupal 7 versions of these modules and extract them to your `/sites/all/modules` directory. The **Link** module is at version 7.x-1.0 and the **References** module is at 7.x-2.0.

Once you install the updated modules go ahead and first enable the 7.x versions of the modules and then run your `update.php` script to update the database schema for each new module.

Fields			
Enabled	Name	Version	Description
<input checked="" type="checkbox"/>	Link	7.x-1.0	Defines simple link field types.
<input checked="" type="checkbox"/>	Node Reference	7.x-2.0	Defines a field type for referencing one node from another. Requires: Field (enabled), Field SQL storage (enabled), References (disabled), Options (enabled)
<input checked="" type="checkbox"/>	References	7.x-2.0	Defines common base features for the various reference field types. Requires: Field (enabled), Field SQL storage (enabled), Options (enabled) Required by: Node Reference (disabled), User Reference (disabled)
<input checked="" type="checkbox"/>	User Reference	7.x-2.0	Defines a field type for referencing a user from a node. Requires: Field (enabled), Field SQL storage (enabled), References (disabled), Options (enabled)

Now go back and launch your **Migrate fields** administration screen again and this time you should see two fields available for migration, **node_reference** and the **link_field**:

Available fields			
Fields that have not yet been migrated but are available for migration. Please carefully read the messages next to understand changes that might be made.			
<input checked="" type="checkbox"/>	Field	Field type	Content type(s)
<input checked="" type="checkbox"/>	field_fire_organization_type	node_reference	Fire Department
<input checked="" type="checkbox"/>	field_fire_website	link_field	Fire Department
<input type="button" value="Migrate selected fields"/>			

Check their boxes to migrate and then click on the **Migrate selected fields** box.

Now launch one of your Fire Department nodes that contains a website link and a node reference to an organization type node and you should see the data. You have successfully migrated all of your Drupal 6 content and Drupal 6 fields to Drupal 7.

Summary

In this chapter we successfully migrated content and data from a Drupal 6 site to a Drupal 7 site, by running a full core and contributed module major upgrade from Drupal 6 to Drupal 7. This was a complex operation with many steps to it. We took content originally built using the CCK module in Drupal 6 and moved all of this content into the new Drupal 7 fields API model, which has CCK built into its core architecture. Not only did we learn to successfully migrate content from Drupal 6 to 7, but we also executed a successful Drupal major upgrade.

In the next chapter we're going to conclude the book by running a migration of content from a Wordpress website over to a Drupal 7 website. This will conclude the book by showing you the full power you will have to migrate content into the Drupal framework.

9

Migrating from WordPress

We've explored various migration processes and techniques throughout the book, specifically how to migrate data and content from legacy websites and applications, and how to migrate content from earlier versions of Drupal to Drupal 7. We have not focused on any specific content management system's migration but rather on general migrations of data that we can import via CSV files using the **Feeds** module. When we migrated our Drupal 6 site to Drupal 7 we used contributed modules to help the migration and upgrade processes.

In this final chapter we're going to turn our attention to one specific content management system and how to migrate the content to Drupal. The CMS we'll focus on is WordPress and the reason we're going to use WordPress as an example is because there is a contributed module called **WordPress Migrate** that can assist us in the migration process. There may be cases where you may want to take your WordPress blog content, for example, and periodically migrate this blog content to your Drupal site. This chapter will show you how to import the WordPress blog posts to Drupal.

These are the topics we'll be covering in this chapter:

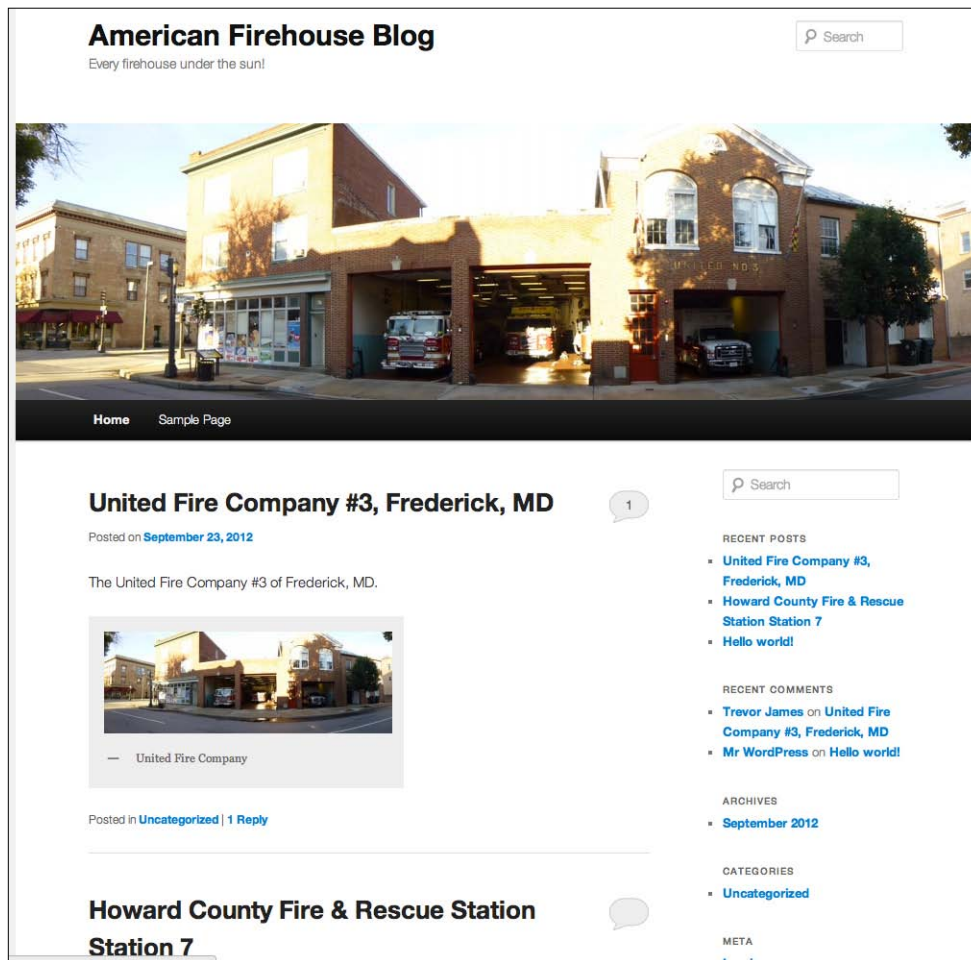
- Migrating content from WordPress to Drupal
- Installing and configuring the WordPress Migrate module
- Using the WordPress Migrate module

Migrating content from WordPress to Drupal

In this chapter we'll assume that you already have a WordPress blog site configured and that it contains a number of blog posts, comments, and other media. The site we'll be using in this chapter is a simple streamlined WordPress site with a couple of blog posts and some comments. We want to migrate this content, the comments, and blog posts to our Drupal 7 site. We'll want the blog posts to be inserted into the blog content type in our Drupal site and the comments to be migrated into the Drupal comments framework.

To migrate the content from WordPress to Drupal we're going to use a module called **WordPress Migrate**. The **WordPress Migrate** module will handle migration of blog posts, comments, tags, and attachments into our Drupal site. This module leverages and depends on the **Migrate** module that we installed in *Chapter 7, Migration Using the Migrate Module*.

Before we start to use this module and attempt to run our migration let's take a look at our WordPress site's homepage. You'll see here that we are posting blog posts with images, a title, and the blog's body text to our WordPress homepage. We're also allowing any replies or comments.



So we want to migrate the post titles, content, and comments into our Drupal site. Let's get started.

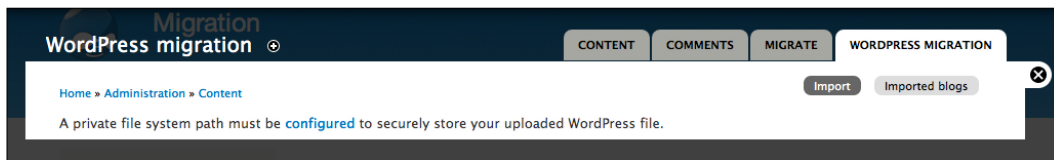
Installing and configuring WordPress Migrate

Let's begin with installing WordPress Migrate:

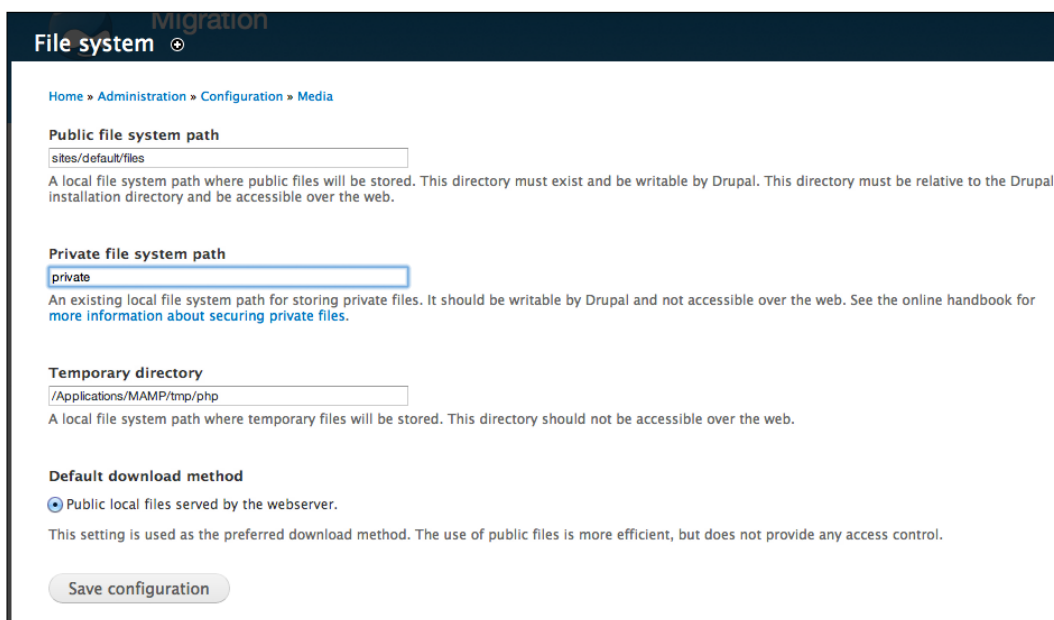
1. First we need to install the latest version of the WordPress Migrate module. The current version is 7.x-2.0. Download and install it from its project page at http://drupal.org/project/wordpress_migrate.
2. Install it as you would do for any contributed module. The module depends on the **Migrate** module that we installed back in *Chapter 7, Migration Using the Migrate Module*. So make sure you have the **Migrate** installed and enabled. Also install the **Migrate Extras** module from http://drupal.org/project/migrate_extras.
3. Once you install **WordPress Migrate**, enable the **Migration from WordPress** module via your modules administration screen:

DEVELOPMENT				
ENABLED	NAME	VERSION	DESCRIPTION	OPERATIONS
<input checked="" type="checkbox"/>	Migrate	7.x-2.4	Import content from external sources Required by: Migrate Example (enabled), migrate_example_baseball (disabled), Migrate example - Oracle (disabled), Migrate Extras (disabled), Migrate Extras Media (disabled), Migrate Extras Pathauto Example (disabled), Migrate Extras Profile2 Example (disabled), Migrate UI (enabled), Migration from WordPress (enabled)	
<input checked="" type="checkbox"/>	Migrate Example	7.x-2.4	Example migration data. Requires: Taxonomy (enabled), Options (enabled), Field (enabled), Field SQL storage (enabled), Image (enabled), File (enabled), Comment (enabled), Text (enabled), Migrate (enabled), List (enabled), Number (enabled)	
<input checked="" type="checkbox"/>	Migrate Extras	7.x-2.4	Adds migrate module integration with contrib modules and other miscellaneous tweaks. Requires: Migrate (enabled) Required by: Migrate Extras Media (disabled), Migrate Extras Pathauto Example (disabled), Migrate Extras Profile2 Example (disabled)	
<input checked="" type="checkbox"/>	Migrate UI	7.x-2.4	UI for managing migration processes Requires: Migrate (enabled) Required by: Migration from WordPress (enabled)	Permissions
<input checked="" type="checkbox"/>	Migration from WordPress	7.x-2.0	Support for migrating WordPress blogs into Drupal Requires: Migrate (enabled), Migrate UI (enabled)	Permissions

4. Once enabled, you can launch the **WordPress migrate** configuration screen from **Content | WordPress migration**. In order to run a WordPress migration you'll first need to configure a private file location in your Drupal file settings. You'll see a message on the **WordPress migration** screen:



5. To configure a private file system, create a folder outside of your MAMP htdocs web root called `/private` and then enter this folder location on your **File system** settings configuration page at `/admin/config/media/file-system`:



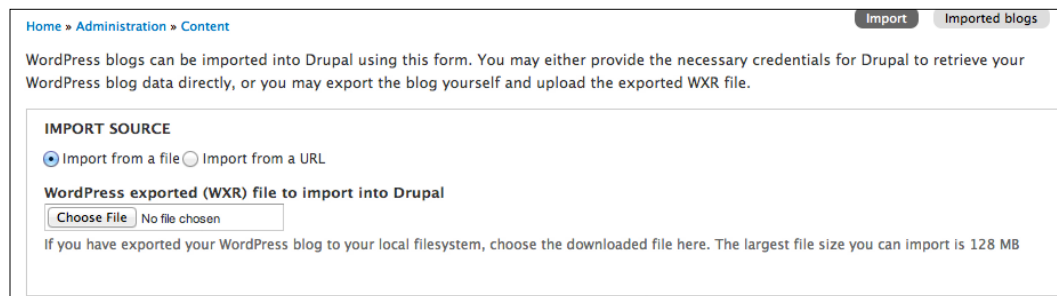
6. Save your configuration and now refresh the **WordPress migration** configuration page.

WordPress Migrate configuration

You should now see a configuration screen that contains various settings. You can choose to import from a file that WordPress can give you as an export from your WordPress site. This file is called a **WXR file**. Or you can configure the import to run from your WordPress site's URL.

If you choose to export and import a WXR file you'll first run the export from your WordPress site. If you choose to run the import from a URL you'll need specific API credentials that you can get from WordPress.


For this example we're going to import our content via a WXR file so let's leave that radio button selected:



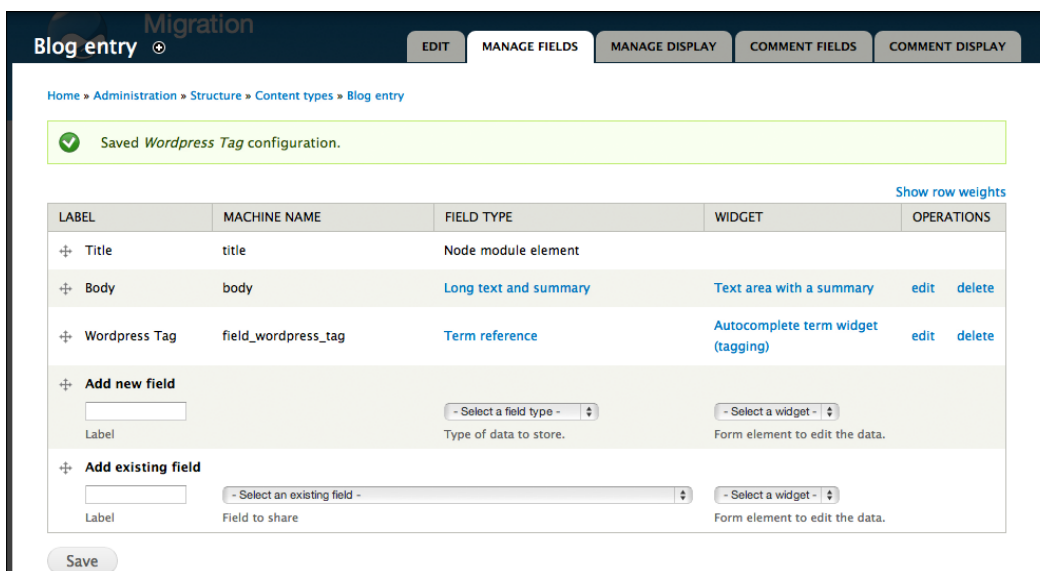
The screenshot shows the 'WordPress Migrate' configuration screen. At the top, there is a breadcrumb trail: 'Home » Administration » Content'. To the right, there are two tabs: 'Import' (active) and 'Imported blogs'. Below the tabs, a text block explains that WordPress blogs can be imported into Drupal using this form, either by providing credentials for Drupal to retrieve data directly or by exporting the blog yourself and uploading the WXR file. The 'IMPORT SOURCE' section contains two radio buttons: 'Import from a file' (selected) and 'Import from a URL'. Below this, a label reads 'WordPress exported (WXR) file to import into Drupal'. Underneath is a file selection interface with a 'Choose File' button and the text 'No file chosen'. A note at the bottom states: 'If you have exported your WordPress blog to your local filesystem, choose the downloaded file here. The largest file size you can import is 128 MB'.

Now scroll down until you see the import settings. Let's configure our import to do the following:

- Convert WordPress pages to Basic page
- Convert WordPress posts to Blog entry
- Default format for text fields – Full HTML
- Default format for comment text fields – Filtered HTML
- Convert WordPress tags to the vocabulary – Wordpress Import


 I added this vocabulary first to our Drupal site. Also make sure this vocabulary is mapped already to the Blog content type on your Drupal site so that you can then leverage this mapping in your WordPress migrate import.

We will do this by adding a **WordPress Tag** term reference field to the **Blog entry** content type in Drupal.



Home » Administration » Structure » Content types » Blog entry

✓ Saved Wordpress Tag configuration.

Show row weights

LABEL	MACHINE NAME	FIELD TYPE	WIDGET	OPERATIONS
✚ Title	title	Node module element		
✚ Body	body	Long text and summary	Text area with a summary	edit delete
✚ Wordpress Tag	field_wordpress_tag	Term reference	Autocomplete term widget (tagging)	edit delete

✚ Add new field

Label - Select a field type - - Select a widget -

Type of data to store. Form element to edit the data.

✚ Add existing field

Label - Select an existing field - - Select a widget -

Field to share Form element to edit the data.

Save

Once you add the term reference to your **Blog entry** content type, you can then select to map Wordpress tags to a vocabulary when configuring Wordpress Migrate.

If you are using Pathauto and want to map your original WordPress URL aliases to the same alias in Drupal, you can select the **Set path aliases to their original WordPress values** radio button. You should now see the following settings screen with your completed configuration for the import:

WordPress blogs can be imported into Drupal using this form. You may either provide the necessary credentials for Drupal to retrieve your WordPress blog data directly, or you may export the blog yourself and upload the exported WXR file.

IMPORT SOURCE

☒ Import from a file ☐ Import from a URL

WordPress exported (WXR) file to import into Drupal

No file chosen

If you have exported your WordPress blog to your local filesystem, choose the downloaded file here. The largest file size you can import is 128 MB

IMPORT SETTINGS

Convert WordPress pages to

Convert WordPress posts to

Default format for text fields

Default format for comment text fields

Convert WordPress tags to the vocabulary

Convert WordPress categories to the vocabulary

Path alias handling

☐ Do not set path aliases

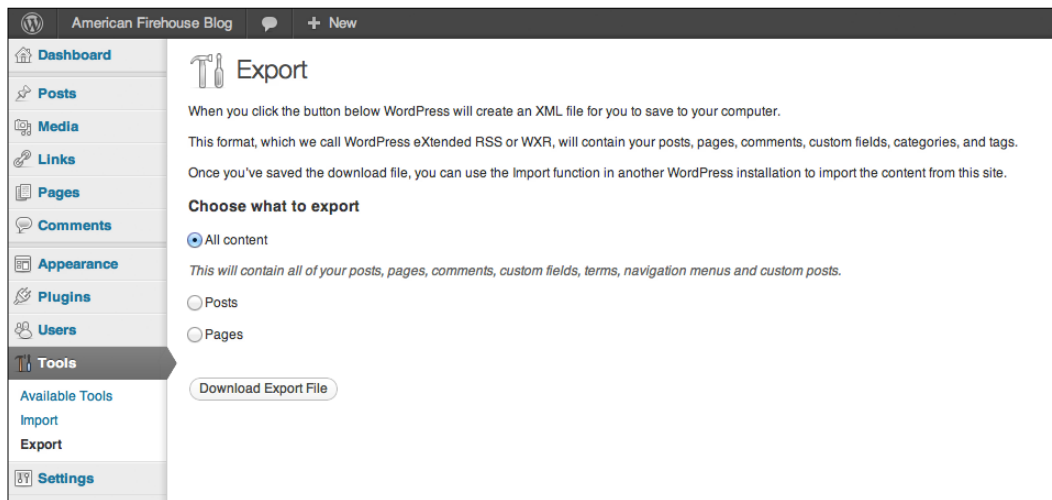
☒ Set path aliases to their original WordPress values

Select how path aliases for imported nodes will be set.

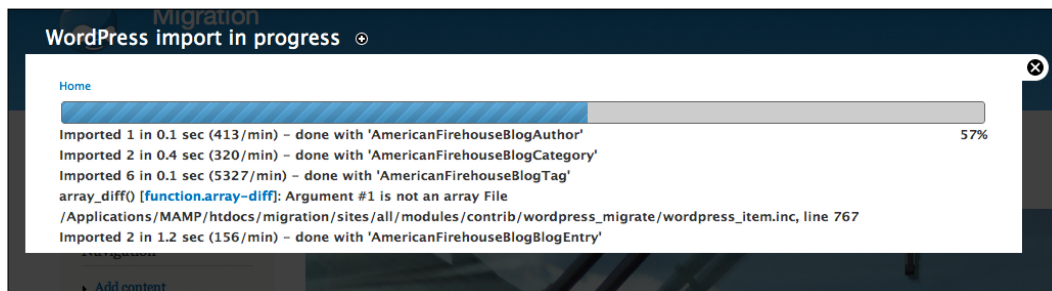
Exporting a WXR file from WordPress

Now we need to export our WordPress content file so we can upload and import it into our Drupal site. To do this, follow these steps:

1. Log in to your WordPress administration interface.
2. Go to **Tools | Export**.

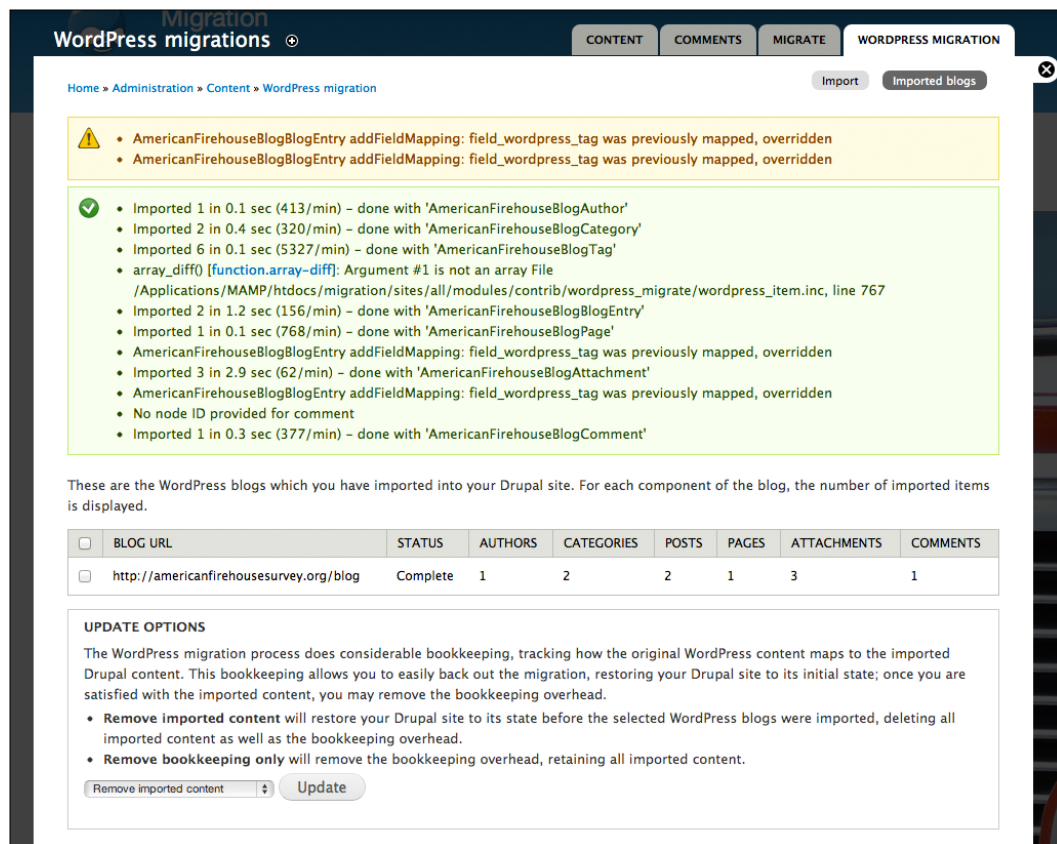


3. Select what you want to export. In our case it will be **All content**. Now click on the **Download Export file** button.
4. This will download an XML file to your local desktop.
5. Now go back to your WordPress Migrate administration screen in your Drupal site and upload the XML file by clicking on the **Choose File** button in the **Import Source** section.
6. Now scroll down and click on the **Import WordPress blog** button. Your import will start to run. You will see a progress bar that looks as follows:



When the import completes you should see a success screen that reports on the import telling you how quickly the import process executed, whether any issues came up during the process, and the specific blog content that was imported.

In our case, the blog content imported had one author, two categories, two blog posts, one page, three attachments, and one comment. You should see the following screen:



WordPress Migration

CONTENT COMMENTS MIGRATE WORDPRESS MIGRATION

Home » Administration » Content » WordPress migration

Import Imported blogs

- AmericanFirehouseBlogBlogEntry addFieldMapping: field_wordpress_tag was previously mapped, overridden
- AmericanFirehouseBlogBlogEntry addFieldMapping: field_wordpress_tag was previously mapped, overridden

- Imported 1 in 0.1 sec (413/min) – done with 'AmericanFirehouseBlogAuthor'
- Imported 2 in 0.4 sec (320/min) – done with 'AmericanFirehouseBlogCategory'
- Imported 6 in 0.1 sec (5327/min) – done with 'AmericanFirehouseBlogTag'
- array_diff() [function.array-diff]: Argument #1 is not an array File /Applications/MAMP/htdocs/migration/sites/all/modules/contrib/wordpress_migrate/wordpress_item.inc, line 767
- Imported 2 in 1.2 sec (156/min) – done with 'AmericanFirehouseBlogBlogEntry'
- Imported 1 in 0.1 sec (768/min) – done with 'AmericanFirehouseBlogPage'
- AmericanFirehouseBlogBlogEntry addFieldMapping: field_wordpress_tag was previously mapped, overridden
- Imported 3 in 2.9 sec (62/min) – done with 'AmericanFirehouseBlogAttachment'
- AmericanFirehouseBlogBlogEntry addFieldMapping: field_wordpress_tag was previously mapped, overridden
- No node ID provided for comment
- Imported 1 in 0.3 sec (377/min) – done with 'AmericanFirehouseBlogComment'

These are the WordPress blogs which you have imported into your Drupal site. For each component of the blog, the number of imported items is displayed.

<input type="checkbox"/>	BLOG URL	STATUS	AUTHORS	CATEGORIES	POSTS	PAGES	ATTACHMENTS	COMMENTS
<input type="checkbox"/>	http://americanfirehousesurvey.org/blog	Complete	1	2	2	1	3	1

UPDATE OPTIONS


The WordPress migration process does considerable bookkeeping, tracking how the original WordPress content maps to the imported Drupal content. This bookkeeping allows you to easily back out the migration, restoring your Drupal site to its initial state; once you are satisfied with the imported content, you may remove the bookkeeping overhead.

- Remove imported content** will restore your Drupal site to its state before the selected WordPress blogs were imported, deleting all imported content as well as the bookkeeping overhead.
- Remove bookkeeping only** will remove the bookkeeping overhead, retaining all imported content.

Remove imported content Update


If for some reason you received errors or other issues during the import process you can simply select **Remove imported content** at the bottom of that report screen in the **Update options** section, making sure to select the corresponding checkbox of the blog that you want to remove content from first. Then click on the **Update** button. Then you can fix any configuration issues and re-run the import again.

Go ahead and review your imported blog content. You should now see the imported WordPress blog posts in your Drupal site as blog entries. You should also see the imported tags, categories, and comments. Here is one of the imported blog posts on the Drupal site:

 Migration [My account](#) [Log out](#)

Home

Home » Blogs » admin's blog



Navigation


- ▶ [Add content](#)
- ▶ [Import](#)

United Fire Company #3, Frederick, MD

[View](#) [Edit](#) [Log](#)

Submitted by [admin](#) on Sun, 09/23/2012 - 16:33

The United Fire Company #3 of Frederick, MD.



United Fire Company

Wordpress Tag:
[Fire Departments](#) [Frederick County Fire and Rescue](#) [United](#) [United Fire Company](#)

[admin's blog](#)

Comments

[admin](#)
Sun,
09/23/2012
- 16:44
[permalink](#)

new

Great photo of the United Fire Company!

[delete](#) [edit](#) [reply](#)

Notice here that the blog title, content, image, tags, and comment have been successfully imported. If you view the main blog page on your Drupal site you should see all the imported blog posts:

admin's blog

+ [Post new blog entry.](#)

United Fire Company #3, Frederick, MD

Submitted by [admin](#) on Sun, 09/23/2012 - 16:33

The United Fire Company #3 of Frederick, MD.



United Fire Company

[Read more](#)

Howard County Fire & Rescue Station Station 7

Submitted by [admin](#) on Thu, 09/20/2012 - 21:20

Station 7: *Banneker*

5815 Banneker Road
Columbia, MD 21044



Ladder 7 HCFRS Banneker Fire Station

[Read more](#)

Summary

In this chapter we successfully migrated our WordPress blog content, images, comments, and tags into our Drupal architecture. We used the WordPress Migrate module to run the migration.

Index

A

additional contributed modules 23

Administration Menu module

about 9, 10

download link 9

B

basic settings, importer configuration 43, 44

BeerTerm migration 102

C

CCK module 118

Chaos Tool Suite module. *See* CTools module

cloning process, importers 73-76

content

migrating, from WordPress to

Drupal 128, 129

Content Construction Kit (CCK) 109

content type

about 19

creating 19-21

display formatter, changing on node

reference field 36, 37

fields, adding 25

Node reference field, testing 35

summary 38, 39

term reference field, testing 37

testing 35

content type fields

planning for 21, 22

contributed modules

about 9, 23

Administration Menu module 9

Link module 24

Location Feeds module 24

Location module 23

upgrading 124, 125

convert_case_hq_city 65

Convert case of HQ City plugin 66

Convert case plugin

adding 64

core modules

Field 8

Field SQL Storage 8

Field UI 8

Node 8

Taxonomy 8

CTools module

about 10

download link 10

installing 10

custom feature module 86, 87

D

Diff module

installing 88, 89

URL 89

Drupal

content, migrating from WordPress 128

content type 19

contributed modules 9

core modules 8

existing data, preparing for migration 17

feeds importer, creating 42

feeds importer, mapping 49

Feeds module, installing 12

Feeds Tamper module, installing 15

Feeds Tamper module, using 62

- import process, running 54
 - Migrate module 17
 - migration, preparing for 7, 8
 - Drupal 6**
 - upgrading, to Drupal 7 110-113
 - Drupal 6 CCK fields and content**
 - migrating, to Drupal 7 122-124
 - Drupal 6 to 7 upgrade process**
 - about 110
 - CCK fields and content, migrating 122-124
 - contributed modules, upgrading 124, 125
 - Drupal core, upgrading 119-121
 - Upgrade Assist 114
 - Upgrade Status 114
 - Drupal core**
 - upgrading 119-121
- ## F
- Features module**
 - about 12, 81
 - content type feature, building 82
 - creating 83-85
 - download link 12
 - enabling 85
 - installing 82
 - migrating, to another Drupal site 95
 - override, reviewing 89, 90
 - overriding 88
 - recreating 90-92
 - reverting 92
 - Feed importer configuration 93**
 - Feed Importer feature 93-95**
 - feeds importer**
 - about 14
 - cloning 42
 - configuration 43
 - creating 42
 - deleting 42
 - exporting 42
 - Feeds module**
 - about 12
 - download link 13
 - installing 12, 14
 - using 110
 - Feeds Tamper module**
 - about 62
 - configuring 62
 - download link 16, 62
 - installing 15-17
 - tamper plugin, adding 64-66
 - Feeds Tamper plugins**
 - URL, for documentation 68
 - Fetcher settings, importer configuration 45**
 - Field module 8**
 - fields, content type**
 - about 25
 - integer fields 29, 30
 - location fields 25
 - location fields, adding 25-29
 - Node reference field 33
 - Term reference field 31
 - text field 28
 - Field SQL Storage module 8**
 - Field UI module 8**
 - Fire Department content type 110**
 - Fire Department importer**
 - configuring 43
 - creating 42
- ## H
- hq_city values 67**
- ## I
- images and files**
 - migrating 39, 40
 - importer configuration**
 - about 43
 - basic settings 43, 44
 - Fetcher settings 45
 - Parser settings 46
 - Processor settings 47
 - import process**
 - running 54-59
 - imports**
 - running, tamper plugin used 67
 - updating 72
 - Import screen 15**
 - import update**
 - running 76, 78
 - integer fields**
 - adding, to content type 29

J

Job Scheduler module

- about 11
- download link 11

L

Link module

- about 24, 125
- download link 24

Location Feeds module

- about 24
- download link 24

Location field

- about 25
- adding, to content type 25-27

Location module

- about 23
- download link 23, 24

M

maintenance mode

- enabling 117

mapping, feeds importer

- creating 49
- source and target, adding 51-53

Migrate module

- about 17, 97
- configuring 100-102
- download link 98
- installing 98, 99

Migrate module configuration

- about 100, 101
- Destination tab 103
- migration, running 106-108
- Source tab 104
- specific migration mapping, viewing 102-105

migration

- running 106-108

N

Node module 8

Node reference field

- adding, to content type 33, 34

- display formatter, changing 36, 37

- testing 35

Node Reference module 124

Node references 33

O

Organization Type 110

override, Features module

- reviewing 89, 90

P

Parser settings, importer configuration 46

Processor settings, importer

- configuration 47, 48

R

References module

- download link 33
- installing 33

results, tamper plugins

- testing 68

S

source and target

- adding, to importer 51, 52

stages, mapping

- Mapping: Client Issues tab 105
- Mapping: DNM tab 105
- Mapping: Done tab 104

T

tamper plugin

- about 64
- adding 64-66
- results, testing 68
- used, for running imports 67

Tamper plugins screen 63

Taxonomy module 8

Term reference field

- about 31
- adding, to content type 32
- testing 37
- vocabulary, adding 31

text field

adding, to content type 28

TextWrangler

URL 57

tid 103

U

update process, imports

cloning process 73-76

importers, cloning 72

import update, running 76, 79

Upgrade Assist module

enabling 115

installing 114

Upgrade Status module

about 114

downloading 114

enabling 115

Upgrade tasks block

about 115, 116

Update modules link 117

V

Views module

about 11

download link 11

W

WordPress

content, migrating to Drupal 128

WXR file, exporting from 134-138

WordPress Migrate

configuring 131-133

installing 130

URL 130

WXR file

about 132

exporting, from WordPress 134-138



Thank you for buying **Migrating to Drupal 7**

About Packt Publishing

Packt, pronounced 'packed', published its first book "*Mastering phpMyAdmin for Effective MySQL Management*" in April 2004 and subsequently continued to specialize in publishing highly focused books on specific technologies and solutions.

Our books and publications share the experiences of your fellow IT professionals in adapting and customizing today's systems, applications, and frameworks. Our solution based books give you the knowledge and power to customize the software and technologies you're using to get the job done. Packt books are more specific and less general than the IT books you have seen in the past. Our unique business model allows us to bring you more focused information, giving you more of what you need to know, and less of what you don't.

Packt is a modern, yet unique publishing company, which focuses on producing quality, cutting-edge books for communities of developers, administrators, and newbies alike. For more information, please visit our website: www.packtpub.com.

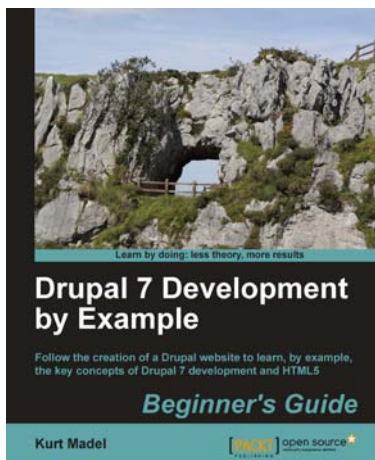
About Packt Open Source

In 2010, Packt launched two new brands, Packt Open Source and Packt Enterprise, in order to continue its focus on specialization. This book is part of the Packt Open Source brand, home to books published on software built around Open Source licences, and offering information to anybody from advanced developers to budding web designers. The Open Source brand also runs Packt's Open Source Royalty Scheme, by which Packt gives a royalty to each Open Source project about whose software a book is sold.

Writing for Packt

We welcome all inquiries from people who are interested in authoring. Book proposals should be sent to author@packtpub.com. If your book idea is still at an early stage and you would like to discuss it first before writing a formal book proposal, contact us; one of our commissioning editors will get in touch with you.

We're not just looking for published authors; if you have strong technical skills but no writing experience, our experienced editors can help you develop a writing career, or simply get some additional reward for your expertise.



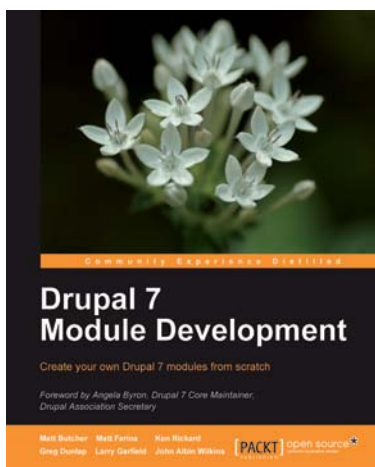
Drupal 7 Development by Example Beginner's Guide

ISBN: 978-1-84951-680-8

Paperback: 366 pages

Follows the creation of a Drupal website to learn, by example, the key concepts of Drupal 7 development and HTML5

1. A hands-on, example-driven guide to programming Drupal websites
2. Discover a number of new features for Drupal 7 through practical and interesting examples while building a fully functional recipe sharing website
3. Learn about web content management, multi-media integration, and e-commerce in Drupal 7



Drupal 7 Module Development

ISBN: 978-1-84951-116-2

Paperback: 420 pages

Create your own Drupal 7 modules for scratch

1. Specifically written for Drupal 7 development
2. Write your own Drupal modules, themes, and libraries
3. Discover the powerful new tools introduced in Drupal 7
4. Learn the programming secrets of six experienced Drupal developers
5. Get practical with this book's project-based format

Please check www.PacktPub.com for information on our titles



Drupal 7 Cookbook

ISBN: 978-1-84951-796-6 Paperback: 324 pages

Over 70 recipes that will advance your Drupal skills from novice to pro

1. Install, set up, and manage a Drupal site and discover how to get the most out of creating and displaying content
2. Become familiar with creating new content types and use them to create and publish content using Views, Blocks, and Panels
3. Learn how to work with images, documents, and video and how to integrate them with Facebook, Twitter, and Add this



Drush User's Guide

ISBN: 978-1-84951-798-0 Paperback: 140 pages

A practical guide to Drush, Drupal's command line interface, helping you work with your Drupal sites more effectively

1. Stop clicking around administration pages and start issuing commands straight to your Drupal sites.
2. Write your own commands, hook in to alter existing ones and extend the toolkit with a long list of contributed modules.
3. A practical guide full of examples and step-by-step instructions to start using Drush right from Chapter 1.

Please check www.PacktPub.com for information on our titles