Drupal 8 Configuration Management

With all the complex configuration options of Drupal, the Configuration Management system allows you to keep track of your configuration and move configuration changes between several development environments with ease.

Drupal 8 Configuration Management will teach you everything you need to know about the Configuration Management system and get you started with using its features quickly. For more advanced users, the book explains how to use the API to add custom configuration and how to upgrade custom modules to use the new Configuration Management system, covering some essential tools and best practices. It will teach you to write schema files for custom configuration and upgrade modules to the new Configuration Management system. Additionally, you will learn how to migrate configuration from Drupal 6 and 7 to Drupal 8 and how to manage configuration for multilingual websites.

Who this book is written for

Drupal 8 Configuration Management is intended for people who use Drupal 8 to build websites, whether you are a hobbyist using Drupal for the first time, a long-time Drupal site builder, or a professional web developer.

What you will learn from this book

- Understand Configuration Management in Drupal
- Set up development environments with ease
- Manage and version control your configuration in code
- Use best practices in your development process
- Upgrade your modules' configuration from Drupal 7 to Drupal 8
- Use configuration in your own custom module
- Discover how the translation of configuration works
- Explore the structure and write your own schema files for configuration

In this package, you will find:

- The authors biography
- A preview chapter from the book, Chapter 1 'Understanding Configuration Management'
- A synopsis of the book’s content
- More information on Drupal 8 Configuration Management

About the Authors

**Stefan Borchert** has been working with Drupal for more than 9 years. In the community, he is better known by his nickname stBorchert. He contributes to Drupal by writing contributed modules, helping with Drupal Core, and providing help to new contributors as a project application review administrator. He is a founding partner and senior Drupal developer at undpaul, a Drupal Digital Agency based in Germany.

**Anja Schirwinsk**i got to know Drupal more than 8 years ago as a themer/site builder and went on to build several very different web applications with it for the company she worked for. She has been a participating member of the Drupal community since 2007, known by the nickname aschiwi.

From 2009-2010, Anja was the deputy chair of the Drupal Initiative, a registered association that promotes Drupal in Germany. She is the cofounder and CEO of undpaul, one of the first Drupal-only digital agencies in Germany. She founded the company in 2010 with friends she met at a local Drupal user group.
Drupal 8 Configuration Management

In professional web development, especially when working in teams of any size, configuration management is one of the most important tasks when it comes to keeping track of configuration changes.

The Wikipedia article for Software Configuration Management states that "In software engineering, software configuration management (SCM) is the task of tracking and controlling changes in the software, which is part of the larger cross-discipline field of configuration management. SCM practices include revision control and the establishment of baselines. If something goes wrong, SCM can determine what was changed and who changed it. If a configuration is working well, SCM can determine how to replicate it across many hosts."

So what is configuration in Drupal terms?

In Drupal, configuration includes topics such as content types, fields, menus, or text formats. Creating or changing a configuration on a live site poses a high risk and makes changes untraceable. Questions such as who made a change, and when and why it was made, cannot be answered.

Up until Drupal 7, Drupal had all configuration stored in the database. By Drupal 7, most professional Drupal developers kept track of their configuration changes by exporting them to code, the most popular option being the Features module, and version-controlling it with a version control system such as Git.

How it works in Drupal 8

When planning for Drupal 8, the so-called Configuration Management Initiative was led by Greg Dunlap in order to make developers' lives easier. Configuration still lives in the database, but can be easily exported to YAML text files. You can now deploy a configuration from one environment to another (between cloned instances of the same site). This capability replaces the need for various contributed modules such as Features, Strongarm, and Context.

This book will teach you everything you need to know about Drupal 8's brand new configuration system. We hope you enjoy it.
What This Book Covers

Chapter 1, *Understanding Configuration Management*, will give you a quick overview of Drupal 8's hottest new feature: Configuration Management. You will learn what types of configuration exist, why managing configuration is a good idea, and how to get started with it. We will introduce you to version control and show you some best practices. We will provide a look at the several ways in which configuration was managed in Drupal 7 and then show how Drupal 8 approaches the problem.

Chapter 2, *Configuration Management for Administrators*, provides an introduction on how to use Configuration Management for users who are not developers, but administrators of a Drupal website who want to make use of the advantages of this new feature. We will show you how to use the Configuration Management interface and how to create a copy of your website, and you will learn how to move a configuration made on one site to another site.

Chapter 3, *Drupal 8's Take on Configuration Management*, will show you the inner workings of the Configuration Management system in Drupal 8. You will learn about config and schema files, and read about the difference between simple configuration and configuration entities.

Chapter 4, *The Configuration Management API*, will teach you how to get your hands dirty and learn about the Configuration Management API of Drupal 8. Here, you will dive into the Simple Configuration API and learn how configuration can be overridden. Later, you will take a closer look at how to create custom configuration entity types, and we'll also teach you about the configuration's context system.

Chapter 5, *The Anatomy of Schema Files*, covers schema files and explains how Drupal uses them for Configuration Management. You will learn about the structure of schema files used by Drupal and write your own schema for custom configuration.

Chapter 6, *Adding Configuration Management to Your Module*, will teach you how to access configuration objects and how schema files are structured in the previous chapters. (You will surely want to know how to get all this fancy stuff into your shiny new module for Drupal 8). You will learn how to include the default configuration in custom modules, how to define and use your own configuration, and how to create configuration forms.

Chapter 7, *Upgrading Your Drupal 7 Variables to the Drupal 8 Configuration*, will show you ways to convert your Drupal 7 variables into Drupal 8 Configuration objects and how to provide an upgrade path in your modules.
Chapter 8, *Managing Configuration for Multilingual Websites*, allows you to build comprehensive multilingual websites in which you can display a site's content in different languages and translate the user interface. While many features were built into Drupal's core in previous versions, building multilingual sites remained a very painful task. In this chapter, we will take a look at how Drupal 7 deals with different languages on a site and how Drupal 8 is trying to fix weaknesses from previous versions.

Chapter 9, *Useful Tools and Getting Help*, provides a list of links and tools provided by the Drupal community; these will be useful if you reach a point where you need help when dealing with Configuration Management.
In this first chapter, we will give you a quick overview of Drupal 8's hottest new feature: Configuration Management. You will learn what types of configuration exist, why managing configuration is a good idea, and how to get started with it. We will introduce you to version control and show some best practices. We will also provide a look at the several ways in which configuration was managed in Drupal 7, and then show how Drupal 8 approaches the problem.

**An introduction to Configuration Management**

The general definition of the term "Configuration Management" is somewhat different from the definition of Configuration Management in Drupal 8. To make things easier, we will focus on explaining what Configuration Management is in Drupal terms.

Configuration Management in Drupal 8 aims at making configuration manageable across different environments by allowing us to store configuration in files instead of the database.

Let's start by defining what configuration is, and what other types of information exist in Drupal 8.
Understanding Configuration Management

**Configuration**
Configuration is the information about your site that is not content and is meant to be more permanent, such as the name of your site, the content types, fields, and views you have defined.

**Content**
Content is the information meant to be displayed on your site, such as articles, basic pages, images, files, and so on.

**Session**
This is the information about an individual user's interactions with the site, such as whether they are logged in.

**State**
This is information of a temporary nature about the current state of your site. Examples include the time when Cron was last run, whether node access permissions need rebuilding, and so on.

**Why manage configuration?**
It's simple to explain why configuration that is only saved in the database is bad. You can't keep track of any changes (who made what change, and when); it's hard to work with a group of people (you simply can't get their changes without using their SQL dump, and using their dump would delete your work); and, if you build something on a development environment, how do you get it to the live site? You get the gist. We want our configuration in files, and Drupal 8 gives us just that.

Before Drupal 8, a variety of methods were used to transport configuration from one environment to another—for example, from a development environment to a production environment.
These included some rather bad methods such as writing down the process to manually recreate the same configuration, which is error-prone; dumping the development database in the live site, which loses all content created in the meantime; and some better but rather time-consuming methods, such as writing update hooks or using the contributed module Features to export configuration to a module. The latter is one of the most used methods in Drupal 7 because it works well most of the time, produces well-arranged files, and can be used without having to write any code, which is good because anyone can create a Feature without having to know how to code.

Even though you can use the new Configuration Management system without a version control system such as Git, it's at its best when used with one. Version-controlling your configuration allows you to track document changes. Later in this chapter, we will show you how to get the best out of version-controlling your configuration. Version-controlled Configuration Management is crucial to developing and maintaining quality in a Drupal project, especially when working with a team of developers. Exposing all developers to the same code and providing a history for the code increases efficiency a lot.

At first, it might seem frustrating to have to learn something new. However, software tends to change over time, and changes are hard to track using just your memory. This is really one of the best ways to improve your project and save your time and money, so make sure you learn it!

**Tracking configuration changes**

Drupal 8's new Configuration Management system can be used without a version control system, but if you want to really improve your process, you should use it in combination with version control. Having organized and versioned code helps prevent mistakes and duplicated efforts between multiple developers; it serves as documentation of the project's history and can show who worked on what and, very importantly, why.

There are others, but we are going to talk about Git as our example version control tool because it's used by the Drupal community and offers everything we need in terms of functionality, scalability, and ease-of-use.

Use a version control tool such as Git to get the best out of the Configuration Management system!
Understanding Configuration Management

The best time to start with versioned Configuration Management is at the beginning of the development. However, it's never too late, even if your project has been started or even finished for a while. Check your Drupal site configuration, organize it, and put everything in a Git repository. Now, you have a good starting point from which to manage and document any changes that will be made to the project in the future.

**Some version control best practices**

So let's see what will really improve the development process when using version control.

**Using a project management tool**

You will achieve the best results if you put your work tasks in a project management tool such as the free and open source tool Redmine. If you're not used to working with a project management tool, it might take some discipline to keep track of your work this way, but it has so many advantages. The ticket holds information about what needs to be done and you can use the ticket's comments to discuss requirements, give status updates, or report problems.

Most project management tools also have some sort of ID for each ticket. You can use the ticket ID in your Git commit messages, which is a very good way to know why a commit was made.
Meaningful commit messages

Commit messages are a very important part of your code documentation when working with version control. When looking for something that was done in the past, you will first scan through the commit messages, as shown in the following screenshot:

<table>
<thead>
<tr>
<th>Date</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>yesterday at 15:51</td>
<td>Issue #7466: Changed map type to google-roadmap</td>
</tr>
<tr>
<td>yesterday at 15:51</td>
<td>Issue #7444: Enable Field Templates for view modes</td>
</tr>
<tr>
<td>yesterday at 15:51</td>
<td>Issue #7444: Added deletion of field_location_route_planner to update script</td>
</tr>
<tr>
<td>yesterday at 15:51</td>
<td>Issue #7444: Remove field_location_route_planner</td>
</tr>
<tr>
<td>yesterday at 15:51</td>
<td>Issue #7466: Disabled tooltip, changed label of popup field to be more logic</td>
</tr>
<tr>
<td>yesterday at 15:51</td>
<td>Issue #7444: Only show locations with a company group selected</td>
</tr>
<tr>
<td>yesterday at 15:51</td>
<td>Issue #7466: Fixed labels and descriptions of location, removed title location in map view</td>
</tr>
</tbody>
</table>

It makes no sense at all to just use a commit message such as *stuff* or even *asdf*. You might laugh, but we've seen both of these in real-world projects. When you start out with version control, it will take some discipline to write meaningful commit messages, but it's really worth it when you come across a bug and are looking for code that might have caused it. Make sure you always use the ticket ID that your project management tool provides and put it at the beginning of your commit message. When you find the commit that causes the problem, the ID will give you more information about what was done there and for what reason.

![Small and well-structured commits are more effective.](image)

Also, make commits small! Do not wait until your workday is over to commit everything you did on that day. This will make it more difficult to go through the changes in that specific commit. For example, make each new contributed module you add to your project a separate commit; do not add 5 modules at once or a module together with other code or configuration.
Meaningful branches

Tickets that require a lot of work should be worked on in a separate branch. When you name that branch, make sure you use your ticket ID at the beginning—for instance, 1234-publications, as shown in the following screenshot:

![REMOTES]

A look back at Drupal 7

Configuration Management in Drupal 7 isn’t as simple as its equivalent in Drupal 8. In Drupal 7, almost the entire configuration set on a site is stored in the database by default. This includes simple variables, content types and field configuration, settings from custom or contributed modules, and so on.

Using the database to store settings makes it really hard to track configuration changes or roll back a bunch of settings to a state defined earlier.

Unfortunately, there is no real standard for Configuration Management in Drupal 7, but there are several ways to manage site and module settings in the code.

We will take a short look at the following five different approaches:

- Manual Configuration Management
- The `hook_install()` / `hook_update_N()` function
- The Features module
- The Configuration Management module
- Storing configuration variables in `settings.php`
Manual Configuration Management

Many users of Drupal manage their configuration manually. They try to remember each setting they've made in the local development environment and then recreate every step on the live site. At first sight, this seems to be very fast and easy, but if you have to manually set permissions for some roles multiple times, you'll never want to do this manually again after hearing there are much better ways.

Additionally, you will never know if a setting has changed and all of your configuration will not be version-controlled (because it only exists in the database). Also, it makes working in a team much more painful than necessary.

If you ever want to share configuration between two or more instances of a site, don't do this.

[Don't use manual Configuration Management!]

The hook_install()/hook_update_N() function

Install and update hooks are the simplest way to manage configuration on a Drupal 7 site in code. The basic idea behind this approach is to set configuration values while installing a module or running update.php. Within the .install file of a custom module, you implement hook_install() and/or hook_update_N(), and add the code needed to set the configuration to these functions:

```php
<?php

/**
 * Implements hook_install().
 */
function my_module_install() {
    // Set site name.
    variable_set('site_name', 'Configuration Management');
}
```

In this example, we simply set the variable site_name to the Configuration Management value, so the name of our site will be updated to this value after enabling the module. The possibilities given here are nearly endless. In addition to setting simple variables, you might add new roles, update block settings, or even create new content types. However, while it's technically possible, it is not recommended and not very simple to export complex configuration (think of fields or views). Also, you need a developer to actually write the code.
Unfortunately, this is one-way configuration management, so there is no way to automatically save changes that you have made on the site's configuration back to code. You have to update the code manually with the new settings (for example, add a new implementation of hook_update_N()).

Additionally, you do not have any chance to see which settings were changed by a user. If you want to save the current state of configuration, you need to go through all settings set in hook_install() or hook_update_N() and compare them with the current settings on the site.

**The Features module**

To manage configuration in Drupal 7, most people use the Features module (https://drupal.org/project/features). If you need a simple tool to export your configuration and put it under version control, Features is the module to work with in Drupal 7.

**What is the Features module?**

To quote James Sansbury from Lullabot:

"The Features module is a module that creates other modules called features."

In other words, Features helps you to put your site's configuration into code so that you can keep track of changes and simply share it with other sites. It was originally created to serve another purpose: to group multiple configurations for one use-case so you could package actual site features and use them in different sites. However, due to a lack of alternatives, it ended up becoming popular as a tool to manage Drupal configuration.

Features works by using so-called components that hold information about configuration objects provided by Drupal itself or contributed modules.

Features uses different types of components: configuration objects that live in code without the need for an instance in the database (exportable components) and so-called faux-exportable components that must exist in the database. Exports of faux-exportable components are used to synchronize configuration objects in the database, so the settings are always up-to-date.

To make an object exportable, you can write a module and use your own default hook handling and export generation. The default hook provides a default state of your configuration object that is directly used on the site or synchronized with the database (depending on the needs of this object).
A very simple example of an object exported using a default hook is a content type. Custom modules can provide their own content types using `hook_node_info()`:

```php
<?php

/**
 * Implements hook_node_info().
 */
function cm_blog_node_info() {
  return array(
    'blog' => array(
      'name' => t('Blog'),
      'base' => 'blog',
      'description' => t('Use for multi-user blogs.'),
    ),
  );
}
?>
```

This simple example (taken from api.drupal.org) defines a new content type with the machine name `blog`. Additionally, it sets the human-readable name to `Blog` and adds a short description to the type, so users know about its purpose.

A better way to make custom configuration objects exportable is to integrate the module with the CTools Export API.

[The CTools Export API has been designed to provide a standardized way to export and import objects in Drupal. Developers simply add some special keys to the object's schema and implement a load function as well as a save function.]

Using the CTools Export API, Features will automatically integrate with your module and handle the export and synchronization of your components. Prominent representatives of contributed modules that implement this in Drupal 7 are Views and Panels.
Creating a Feature

Creating a Feature is very easy. Using the user interface of the Features module, you simply add the components you would like to export to the newly created module. While generating the new module, Features uses the defined default hooks or the CTools Export API to save the information about the components to code so you don't need to write the code yourself. While writing the code may be fairly easy for content types (as shown previously), writing down the complete configuration of a field, an image style, or even a view is not so simple, and you do not want to do this manually. With Features, you only need a few clicks to get the configuration into code. Take a look at the following screenshot:

In the preceding example, we selected the content type Blog along with some permissions. As you can see, Features automatically added the required dependencies to other modules along with the information about the fields of the content type and common variables related to the type.
After adding everything you want to include in the export, you can download the feature or let Features directly create the files on your disk.

If you create a new Feature, make sure you use a unique machine-readable name that does not conflict with any existing module. The best practice is to prepend the machine name with an abbreviation of your project or site name (in our example, cm_blog).

After downloading the Feature and enabling it in the same way as any other module, you are able to track changes to components in the Feature. For example, if you change the label of a field included in the Feature, the Feature will be shown as overridden. With the help of the Diff module, it even displays each modified component as follows:

You can then choose between reverting the Feature to its default state (that's what you have in the code of your Feature), which would undo the change you made to your field label, or you can update the Feature, which gives you the modified values in code, so you can share it with others or distribute it to another environment.

Both tasks can either be done using the Feature UI or Drush, which is much faster.

**The settings to export with Features**

Basically, all components that rely on the CTools Export API, or on modules that define default hooks, may be exported.
These include the following:

- **Variables**: These are exported using the Strongarm module, which implements the CTools Export API for all entries in the variables table of Drupal.

- **Views**: These are exported using the default hook: `hook_views_default_views()`.

- **Content types**: These are directly exported by the Features API using Drupal's `hook_node_info()`.

- **Field definitions**: These are exported using default hooks defined by Features itself.

- **And many more**: These include text formats, image styles, and rules ([http://www.drupal.org/project/rules](http://www.drupal.org/project/rules))

### The settings to not export with Features

While some components may theoretically be exportable, it is not always sensible to do this. For example, exporting cache variables or variables that store timestamps such as `cron_last`, which stores the date when the last cron was run, would result in constantly overridden Features. There is also no benefit in having components such as this in code, because you can't actively change it, and you don't need to know its value for anything.

As a general rule of thumb, you should never export components that change often, such as timestamps or status variables.

### The Configuration Management module

The Configuration Management module is the latest approach we will take a look at here. While Features was never really intended to do real Configuration Management, the Configuration Management module takes some core concepts from the Drupal 8 Configuration Management Initiative and makes them available for Drupal 7.

The main concept behind this module is the data storage architecture. It defines an activestore and a datastore to manage the configuration of a site. The activestore represents the current state of an individual configuration component (for example, a variable in the database) whereas the datastore is defined as the file that contains the default state of the component.
After changing the value of a component tracked by the Configuration Management module, you can save its value back to the datastore (the module updates the corresponding files for you) so that you can track the changes in your version control system.

Looking at the export of this configuration in the following screenshot, you will notice many similarities. This is due to the fact that both modules use the CTools Export API and nearly the same default hooks to import/export the data.

```php
$data = (object) array(
    'type' => 'blog',
    'name' => 'Blog',
    'description' => '',
    'has_title' => '1',
    'title_label' => 'Title',
    'base' => 'node_content',
    'help' => '',
);  
$dependencies = array();

function cm_blog_node_info() {
    $items = array(
        'blog' => array(
            'name' => t('Blog'),
            'base' => 'node_content',
            'description' => '',
            'has_title' => '1',
            'title_label' => t('Title'),
            'help' => '',
        ),
    );
    return $items;
}
```

The main advantage of the Configuration Management module in comparison to Features is the reduction to pure Configuration Management. There is no possibility for a developer to extend the export with custom code (that is hook_form_alter() or hook_menu()) as is done often when exporting configuration objects with Features. The export simply contains the components you want to put under version control and nothing more.

### Storing configuration variables in settings.php

There is one more way to store settings back in Drupal 7: your site's settings.php, which you know from storing your database details in it. The Drupal installation process and Drupal modules use the variables table to store different types of information that will be used at runtime. The values of these variables can be overridden in the settings.php file. Every module, when enabled, may add variables that can be altered in the configuration setting. One example is the variable named theme_default, which sets the default theme.
Understanding Configuration Management

Variables stored inside your `settings.php` file's `$conf` array will override whatever is in the variables table of your database. This is really useful when you need different configuration for different environments, such as local, staging, and production.

There is a complete list of default variables available on a fresh installation of Drupal at https://www.drupal.org/node/1525472.

How Drupal 8 takes care of Configuration Management

Drupal 8 totally changes the way configuration is managed on a site. The configuration can be stored in files instead of the database, so it is not a problem to put it under version control.

All default configuration defined and used by a module must be able to be stored in special configuration files using the YAML specification and the `.yml` file extension. YAML is short for YAML Ain't Markup Language; according to its creators, YAML is a human-friendly data serialization standard for all programming languages. In short, it's easier to read and write. Each module provides its own default configuration files in a special folder named `config`, which makes it easy to see which configuration a module provides. Taking the core system module as an example, you will find several files in the `config` directory responsible for all configurations that the system module handles on the site.

How to start using Configuration Management

By default, Drupal 8 stores configuration in the site's database. During installation of your Drupal site, Drupal adds a directory within `sites/default/files` called `config_HASH`, where HASH is a long random string of letters and numbers, as shown in the following screenshot:
Using version control to keep track of configuration changes

Inside this config directory, there are two more directories: active and staging. Both contain no configuration files by default, but they each contain a helpful README.txt.

The contents of the active directory's README.txt are as follows:

If you change the configuration system to use file storage instead of the database for the active Drupal site configuration, this directory will contain the active configuration. By default, this directory will be empty. If you are using files to store the active configuration, and you want to move it between environments, files from this directory should be placed in the staging directory on the target server. To make this configuration active, visit admin/config/development/configuration/sync on the target server. For information about how to deploy configuration between servers, see http://drupal.org/documentation/administer/config.

The staging directory's README.txt explains the following points:

In order to start using Configuration Management to keep track of your configuration changes, all you have to do is export your current configuration and place it inside the staging directory as follows:

1. Go to /admin/config/development/configuration/full/export and use the Export button to download an archive of your site configuration, as shown in the following screenshot:
Understanding Configuration Management

2. Save the archive inside the `sites/default/files/config_HASH/staging` folder of your Drupal source files and extract the contents of the archive. The result should look something like this:

If you're familiar with the Drupal command-line tool Drush, you can export configuration with a simple command. Check Chapter 9, *Useful Tools and Getting Help* for details.

You can find more detailed information in the next chapter, *Chapter 2, Configuration Management for Administrators.*

**Types of configuration**

There are two types of configuration in Drupal 8: simple configuration and configuration entities.

Simple configuration is basically the same as variables (that is, the site name or the number of nodes on the front page) and is used for single global settings.

Looking at the system module's configuration file `system.site.yml`, you see some examples for simple configuration. The file defines the default values for some of the main settings you will need on your site – that is, the site name or the default e-mail address:

```yaml
name: 'Configuration Management in Drupal 8'
mail: 'info@example.com'
slogan: ''
page:
  403: ''
  404: ''
front: user
langcode: en
```

As you can see, configuration can even be nested, so you can group settings.

Configuration entities are more complex than a simple configuration, and are used for objects that can have multiple copies such as content types or views.
Configuration storage and deploying between environments

Earlier in this chapter, we learned about the directory named staging. In this directory, you put the configuration you would like to import into a copy of your Drupal site—for example, to copy changes from your local environment to your production site. Simply export the new configuration from your local environment, place it in the staging directory of your production site (preferably by using version control), and import it later at admin/config/development/configuration/sync.

Note that, at the time of writing this book, the active directory is not used as originally intended. Its original purpose was to store the site's currently active configuration but, since that is now kept in the database, the active directory remains empty. This might change in future versions of Drupal 8.

Summary

Now you have a very complete overview of what Configuration Management is in Drupal 8 and why you should make use of it. You read about some best practices that show you how to best keep track of your changes with version control. You also learned about all the different ways to achieve some kind of Configuration Management in Drupal 7 and were given a basic introduction to the way it works in Drupal 8. Read on to find out how site administrators with no programming knowledge can use this system.
Where to buy this book

You can buy Drupal 8 Configuration Management from the Packt Publishing website.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.

Click here for ordering and shipping details.