

Textpattern Solutions: PHP-Based Content Management Made Easy

Kevin Potts, Robert Sable, and Nathan Smith
with Mary Fredborg and Cody Lindley



Textpattern Solutions: PHP-Based Content Management Made Easy

Copyright © 2007 by Kevin Potts, Robert Sable, Nathan Smith, Mary Fredborg, Cody Lindley

All rights reserved. No part of this work may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage or retrieval system, without the prior written permission of the copyright owner and the publisher.

ISBN-13 (pbk): 978-1-59059-832-0

ISBN-10 (pbk): 1-59059-832-6

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

Trademarked names may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, we use the names only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

Distributed to the book trade worldwide by Springer-Verlag New York, Inc., 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax 201-348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com.

For information on translations, please contact Apress directly at 2560 Ninth Street, Suite 219, Berkeley, CA 94710. Phone 510-549-5930, fax 510-549-5939, e-mail info@apress.com, or visit www.apress.com.

The information in this book is distributed on an “as is” basis, without warranty. Although every precaution has been taken in the preparation of this work, neither the author(s) nor Apress shall have any liability to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by the information contained in this work.

The source code for this book is freely available to readers at www.friendsofed.com in the Downloads section.

Credits

Lead Editors

Chris Mills, Matthew Moodie

Assistant Production Director

Kari Brooks-Copony

Technical Reviewer

Mary Fredborg

Production Editor

Ellie Fountain

Editorial Board

Steve Anglin, Ewan Buckingham,
Gary Cornell, Jason Gilmore,
Jonathan Gennick, Jonathan Hassell,
James Huddleston, Chris Mills,
Matthew Moodie, Jeff Pepper,
Dominic Shakeshaft, Matt Wade

Compositors

Dina Quan and Darryl Keck

Artist

April Milne

Proofreaders

Paulette McGee and Elizabeth Berry

Project Manager

Richard Dal Porto

Indexer

Julie Grady

Copy Edit Manager

Nicole Flores

Interior and Cover Designer

Kurt Krames

Copy Editor

Nancy Sixsmith

Manufacturing Director

Tom Debolski

14 WRITING PLUGINS

<p>Content</p> <p>Preferences Users</p> <p>Advanced</p> <p>Basic Advanced Language</p> <p>Admin</p> <p>How many articles should b</p> <p>Send "Las</p>	<p>Section name: <input type="text" value="plugin-test"/></p> <p>Section title: <input type="text" value="plugin-test"/></p> <p>Uses page: <input type="text" value="plugin-test"/> ▼</p> <p>Uses style: <input type="text" value="default"/> ▼ ?</p> <p>Selected by default? <input type="radio"/> No <input type="radio"/> Yes ?</p> <p>On front page? <input type="radio"/> No <input type="radio"/> Yes ?</p> <p>Syndicate? <input type="radio"/> No <input type="radio"/> Yes ?</p> <p>clude in site search? <input type="radio"/> No <input type="radio"/> Yes ?</p>	<pre># A basic Textpattern plugin. # Rob Sable # http://www.wilshireone.com/ # # This is a plugin for Textpat # To install: textpattern > ac # Paste the following text int # H4sIAAAAAAAAAA3VRW2vCMBR+bn5FCE w71+1xNGO5SeLV1R0rNOKMLSdE2Jsf 2VLypEr8zMr2hpV8dY+Dkb6yAJzGOC Kz1+ER9OM+eE6bFuhzfZR34EHLiTB1 /aN3Pv9B9OrPsIuhiWBG718aaTE89:</pre>
--	--	--

This chapter covers all the steps necessary to write, test, and release your own Textpattern plugins. It starts by walking you through the steps you need to take to set up a local plugin development environment. It then moves you through several plugin examples and references time-saving helper functions within the Textpattern source code that you can use as building blocks for your own plugins.

Before you start

If you're interested in writing plugins for Textpattern, you should first have an understanding of PHP and experience coding and testing PHP scripts. If you have experience programming in another language, PHP should be relatively easy to learn.

If you're looking for resources to begin learning PHP or just need to brush up on your skills, the following are good places to start:

- PHP Manual at www.php.net/manual/en/
- PHP Resources at www.friendsofed.com/book.html?isbn=1590597311

You should also have some familiarity with the MySQL database engine. Most plugins use data stored in the Textpattern database in some fashion, which means that you need to be familiar with SQL query syntax. While there are helper functions within the Textpattern code that can be used instead of writing all your queries from scratch, a basic understanding of SQL is necessary.

Digging in to the core code that drives Textpattern also helps you to gain an understanding of some of the functions and global variables you have at your disposal. Although you'll learn more about these subjects later in the chapter, here are some online resources that you'll find useful:

- Browse the Textpattern source code and revision history at <http://dev.textpattern.com/browser> and <http://dev.textpattern.com/timeline>.
- PHPXref generates source code documentation in an easily searchable HTML format at www.phpxref.com/xref/textpattern.

Finally, before spending time writing your own plugin, make sure that someone else hasn't already done the job. The Textpattern Forum at <http://forum.textpattern.com> and the Textpattern Resources site at www.textpattern.org are the best places to find details about previously released plugins. Reading through the code of plugins that others have developed can also help you to become familiar with different coding techniques.

Getting started

The first step of writing your own plugins is to set up a local development environment. In Chapter 2, you learned how to set up a local Textpattern installation, which gives you a testing ground for your new plugins. If you haven't already completed your local

Textpattern install, you should do that before reading any further. After you have a local Textpattern install up and running, it's time to start setting up your plugin development environment.

Textpattern plugin template

The Textpattern plugin template is the first building block to put in place. You can find the latest version in the Textpattern subversion repository at <http://svn.textpattern.com/development/4.0-plugin-template/>. You can also find the latest download package and discussion on the Textpattern forum at <http://forum.textpattern.com/viewtopic.php?id=10330>. Download the template files to the computer you'll be developing on.

The template actually includes three files:

- `zem_plugin.php` is the actual template you use to create the plugins. The file has been commented so that you can see where to place your PHP code, Textile-formatted help, and plugin metadata.
- `zem_plugin_example.php` contains examples of public and admin plugins. It isn't needed to create new plugins, but can be used for reference.
- `zem_tpl.php` is the plugin compiler that you use to build the plugins for installation into Textpattern. The compiler converts raw PHP code and help to a base64-encoded string that will be pasted into the Admin ► Plugins tab. It will be used when you're ready to release the plugins to the Textpattern community.

You'll dig in to the details of the plugin template files after you complete the local setup.

Local workspace setup

Once the plugin template has been downloaded locally, create a new directory that you'll use as the plugin-development workspace. For the rest of this chapter, `c:\txp` is used as the directory name. If you're working on an operating system other than Windows, just replace the directory name with the one you created. Now that you have a new empty directory, it is time to pull in the files you need to develop:

1. Copy the `zem_tpl.php` file into your `c:\txp` directory.
2. Copy the `classTextile.php` file from the `textpattern/lib` directory of the Textpattern installation into your `c:\txp` directory. This file enables you to create Textile-formatted help that will be included in the compiled plugin.

That's all you need to compile the plugins. Next, you configure the local Textpattern install so that you can start developing and testing the plugins.

Local Textpattern setup

There are several helpful features built in to Textpattern that make life easier for plugin developers. These features help you save time while you develop, test, and debug the plugins.

TEXTPATTERN SOLUTIONS: PHP-BASED CONTENT MANAGEMENT MADE EASY

The first feature is the Plugin cache directory. When configured, Textpattern looks in this directory for plugin files and loads them as pages are built. This saves you from having to reinstall the plugin each time you need to test new functionality. To set up the directory, follow these steps:

1. Create a new directory within your Textpattern installation. (Create a new directory called `plugins` in the `textpattern` directory of the install.)
2. In the Textpattern admin interface, navigate to the Admin ► Preferences ► Advanced page.
3. Find the preference called Plugin cache directory path in the Admin section, as shown in Figure 14-1.
4. Copy the full path of the directory you just created into that field. For example, my local path is `c:\apache\htdocs\textpattern\plugins`.
5. Save your preferences.

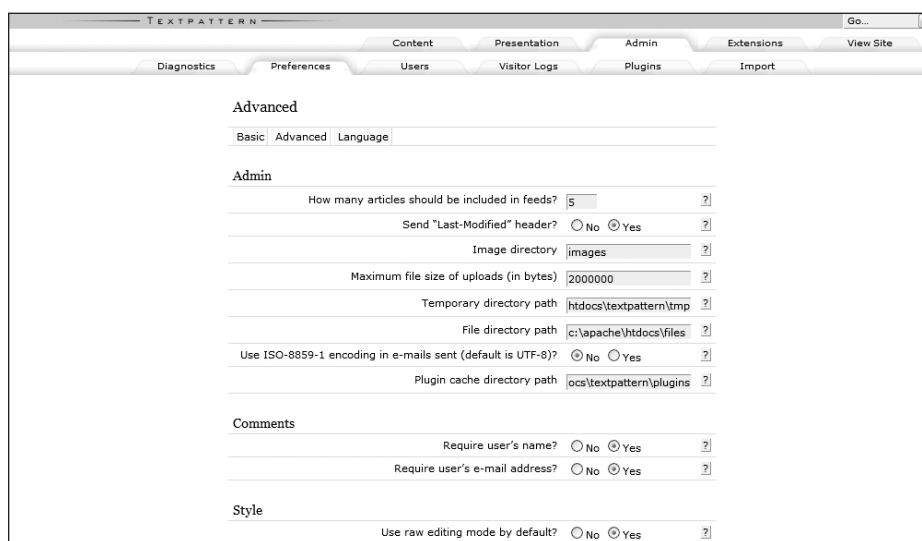


Figure 14-1. Setting the Plugin cache directory path on the Advanced Preferences page

Next, you want to make sure that Textpattern is running in debug mode. Debug mode gives you more-detailed error messages on the screen, which can help you fully test the plugins and ensure that no errors are hidden.

1. In the Textpattern admin interface, navigate to the Admin ► Preferences ► Basic page.
2. Find the preference called Production Status in the Publish section.
3. Set the value to Debugging, as shown in Figure 14-2.
4. Save your preferences.



Figure 14-2. Setting the Production Status on the Basic Preferences page

Plugin loading

Before you jump in and start coding the plugins, it is helpful to understand how Textpattern loads and uses plugins.

When plugins are installed in the Textpattern admin interface on the Admin ► Plugins tab, they are stored in the `txp_plugin` table in the Textpattern MySQL database. Each time Textpattern renders a public-side or admin-side page, plugin code is loaded from the database and evaluated. In the same way, plugin files that you place in the Plugin cache directory are loaded as pages are built. You can think of plugin loading as being analogous to the PHP `include` function. All the plugin code is simply included along with the base Textpattern code.

Plugins are loaded using the `load_plugins()` function that can be found in `/textpattern/lib/txplib_misc.php`. The specific point where plugins are loaded during the page-rendering process can be found on line 99 of `/textpattern/publish.php` for public-side plugins and line 89 of `/textpattern/index.php` for admin-side plugins. If you track down those lines of code, notice that there's a difference in the way the function is called.

As you'll see as you start working through the Textpattern plugin template, each plugin has a type. A type code of 0 indicates a public-side only plugin, while a plugin with a type code of 1 also includes admin-side code. Therefore, when the `load_plugins()` function is called during admin-side plugin loading, a parameter of 1 is passed in so that plugins used on the public side only are not loaded.

Basic plugin topics

The next section covers all the basics needed to write, test, compile, and release a simple Textpattern plugin. For that, turn back to the local development environment and use the `zem_plugin.php` file as the start to the plugin.

Textpattern plugin template explained

The `zem_plugin.php` file is the basis for all the plugins you create. The file contains several explanatory comments. In the following code, I removed the comments to shorten the size of the file. Once you're familiar with how the template functions, you can do the same. I'll start by stepping through this file to explain how it should be used.

```

1  <?php
2  # $plugin['name'] = 'abc_plugin';
3  # $plugin['allow_html_help'] = 0;
4
5  $plugin['version'] = '0.1';
6  $plugin['author'] = 'Alex Shiels';
7  $plugin['author_uri'] = 'http://thresholdstate.com/';
8  $plugin['description'] = 'Short description';
9  $plugin['type'] = 0;
10
11 if (!defined('txpinterface'))
12     @include_once('zem_tpl.php');
13
14 if (0) {
15 ?>
16 # --- BEGIN PLUGIN HELP ---
17 h1. Textile-formatted help goes here
18
19 # --- END PLUGIN HELP ---
20 <?php
21 }
22
23 # --- BEGIN PLUGIN CODE ---
24
25 // Plugin code goes here. No need to escape quotes.
26
27 # --- END PLUGIN CODE ---
28 ?>

```

The first construct you notice throughout the file is an array called `plugin`. You'll set several different values on this array in the plugin template. These values correspond directly to values in the `txp_plugin` table and will be shown when the plugin is installed on the Admin ► Plugins tab.

Your work starts on line 2 with the `$plugin['name']` field. By default, the plugin name is taken from the plugin file name. For example, if the plugin file is named `myplugin.php`, the plugin name is `myplugin` when installed into Textpattern. To maintain control over the plugin name, I recommend uncommenting line 2 and setting your plugin name specifically. Doing so enables you to maintain different versions of the file while keeping the same plugin name.

On line 3, you can change the format for plugin help. By default, the help text you write is passed through the Textile engine for formatting. By uncommenting line 3, the plugin help is interpreted as raw HTML and is not parsed by Textile before display. But per the comment in the plugin template, this is not a recommended setting. It is easiest to remove this line from your plugin unless you have a need to change the setting.

Lines 5–9 contain specific details about the plugin, including the plugin name and description, the plugin type, and the plugin author's name and website address. The first 4 lines of this section should be easy enough for you to determine, and line 9 refers to the plugin type covered earlier. Again, per the comments in the plugin template, the valid types are the following:

- 0 is used for a regular plugin. The plugin is loaded on the public side only.
- 1 is used for an admin plugin. The plugin is loaded on both the public and admin sides.
- 2 is used for a plugin library. The plugin is loaded only when `include_plugin()` or `require_plugin()` is called.

The bulk of the work you do in the plugin template occurs between lines 16 and 27 in the template file. All the help text goes between the lines that read `# --- BEGIN PLUGIN HELP ---` and `# --- END PLUGIN HELP ---` on lines 16 and 19. All the plugin code goes between the lines that read `# --- BEGIN PLUGIN CODE ---` and `# --- END PLUGIN CODE ---` on lines 23 and 27. Be sure you don't alter any of these lines because they are used by the plugin compiler.

Now that the plugin template has been reviewed, you can move on to creating the first basic plugin.

Writing a basic plugin

The first step of writing the plugin is to copy the plugin template into the Plugin cache directory. Once you copy the template, start by renaming the file so that you can identify your plugin and its version. In this case, name the file `rss_hello_world-0.1.php` so that you can easily tell that this is version 0.1 of the `rss_hello_world` plugin.

Notice that the name of the plugin is preceded by a three-character code. To easily identify the author of plugins and to prevent name collisions, each plugin author selects a unique three-character prefix that precedes the name of the plugins. Typically, the code is an author's initials or nickname, but that's up to you. For all the plugin examples in this chapter, I'll be using my standard prefix of `rss`, which represents my initials. Before you

TEXTPATTERN SOLUTIONS: PHP-BASED CONTENT MANAGEMENT MADE EASY

begin writing your own plugins, search through the list of existing plugins on the Textpattern Resources website at www.textpattern.org to ensure that the prefix you want to use hasn't been taken.

Now that you have the plugin template file set up, it is time to start coding. Here's the code for the first basic plugin. When called from a page template or form, the `<txp:rss_hello_world />` tag displays the text Hello Textpattern World, my name is Anonymous on the page.

```
<?php
$plugin['name'] = 'rss_hello_world';
$plugin['version'] = '0.1';
$plugin['author'] = 'Rob Sable';
$plugin['author_uri'] = 'http://www.wilshireone.com/';
$plugin['description'] = 'A basic Textpattern plugin.';
$plugin['type'] = 0;

if (!defined('txpinterface'))
    @include_once('zem_tpl.php');

if (0) {
?>
# --- BEGIN PLUGIN HELP ---

h1. Hello, Textpattern World

This is a basic Textpattern plugin.

# --- END PLUGIN HELP ---
<?php
}
# --- BEGIN PLUGIN CODE ---

function rss_hello_world($atts) {
    extract(1Aatts(array(
        'name' => 'Anonymous',
    ),$atts));

    return 'Hello Textpattern World, my name is '.$name;
}

# --- END PLUGIN CODE ---
?>
```

Starting from the top of the file, you see that the basic plugin metadata has been defined based on the name and version of the plugin, along with my name and website URL. Some basic help has been defined, but for now, skip down to the plugin code.

Plugins as tags

Each function you write in your plugin code corresponds to a tag that can be used on your Textpattern page templates and forms. In the first example, since a function called `rss_hello_world` was created, you can call the plugin from Textpattern pages using the tag `<txp:rss_hello_world/>`.

Self-closing vs. enclosing plugin tags

Just as with standard Textpattern tags, you have the ability to create a self-closing plugin tag or an enclosing plugin tag. In the example already begun, you are creating a self-closing tag. You can easily discern a self-closing tag from an enclosing tag based on the tag function's signature. If you attempt to use a self-closing tag as an enclosing tag, you receive a `tag_error` notification on the screen.

A **self-closing** tag accepts only an array of variables as a parameter. All self-closing plugins are called as follows:

```
<txp:rss_self_closing_tag />
```

An **enclosing tag** accepts an array of variables along with the addition of a second parameter to the function. The second parameter holds the content you'll display between the opening and closing tags of the plugin. That content can include plain text, HTML, core Textpattern tags, or plugin tags.

```
<txp:rss_enclosing_tag>
Content
</txp:rss_enclosing_tag>
```

You'll see an example of an enclosing plugin tag later in this chapter, but for now continue through the self-closing plugin example.

Plugin attributes

The first construct you see in the plugin code is as follows:

```
extract(lAtts(array(
    'name' => 'Anonymous',
), $atts));
```

Although just a few lines, the preceding code is used to initialize the local variables used in the plugin and to do some basic validation of the attributes specified on the tag. The `lAtts()` function can be found in `/textpattern/lib/txplib_misc.php` if you want to step through the source.

The first thing to notice is the definition of a name/value pair. Typically, plugins have several of these pairs defined. These pairs enumerate the valid attributes that can be specified when calling the plugin and the default values for those attributes if they are not specified.

For example, the preceding code defined a local variable called `name` that has a default value of `Anonymous`. When the plugin is called, if an attribute called `name` is specified, the value passed in to the plugin is assigned to the `name` variable. However, if the plugin is called without specifying the `name` attribute, the `name` variable is assigned a default value of `Anonymous`. Each `name` specified in the array translates to a local variable by the same name.

The preceding code also verifies that the attributes specified when calling the plugin are valid. In this example, an attribute called `name` is the only valid attribute that can be passed into the plugin. The array of name/value pairs that I specified for the plugin will be compared against the `$atts` array that is passed in to the plugin code from `Textpattern`. This array contains all plugin attributes and their values as specified when the plugin tag is called. So if I attempt to call the plugin using the call `<txp:rss_hello_world firstname="Rob" />`, I receive an error message to let me know that `firstname` is not a valid attribute of the plugin.

Once this code has been executed, you are left with a collection of local variables that can be used in the plugin. The values were determined either by those passed in when calling the plugin or by the defaults that you specified.

Plugin output

Each tag handler plugin can return output to the `Textpattern` page. In the example, you see this in the following line of code:

```
return 'Hello Textpattern World, my name is '.$name;
```

The output of the plugin is displayed as a string, so you should ensure that the value returned is a string, not a PHP object. Using the `return` statement is a must because it places the string within the flow of the page as it is built. If you were to use a PHP output function such as `echo` or `print_r`, the output would be placed outside of the `Textpattern` page and appear at the top of your browser window.

Testing the first basic plugin

Because you placed the plugin file in the Plugin cache directory, it is now available to be used within `Textpattern`. At this point, you add the plugin tag to a `Textpattern` page for testing.

1. Start by creating a new page on the Content ► Pages tab called `plugin-test`. Copy the existing default page and remove the HTML within the `<body>` tags, which leaves you with the shell of a basic `Textpattern` HTML page.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
    <meta http-equiv="content-type"
    content="text/html; charset=utf-8" />
```

```

<link rel="stylesheet" href="<txp:css />" ↵
type="text/css" media="screen" />
<title><txp:page_title /></title>
</head>
<body>

</body>
</html>

```

2. Next, create a new section on the Content ► Sections tab called plugin-test and assign the new plugin-test page to the section. The configured section is shown in Figure 14-3.

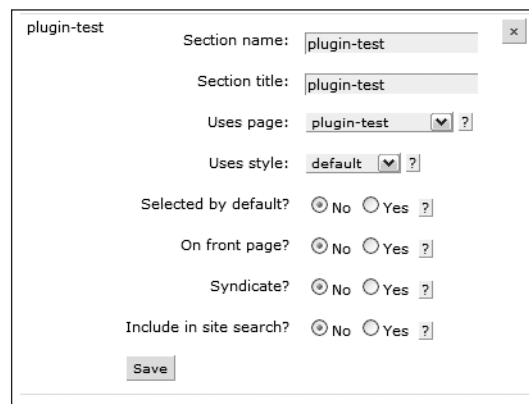


Figure 14-3. Creating a new section for plugin testing

3. Finally, add a call to the plugin within the <body> tags of the page and test the output. You can call the plugin in its simplest form as follows: <txp:rss_hello_world/>.

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
<meta http-equiv="content-type" ↵
content="text/html; charset=utf-8" />
<link rel="stylesheet" href="<txp:css />" ↵
type="text/css" media="screen" />
<title><txp:page_title /></title>
</head>
<body>
<txp:rss_hello_world />
</body>
</html>

```

This code gives you the HTML output shown in Figure 14-4.

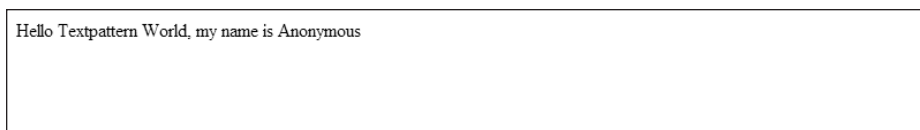


Figure 14-4. Basic plugin output with no attributes specified

The plugin call did not specify the name attribute, so the output includes the default value of Anonymous. At this point, you've created and used a new Textpattern plugin.

Calling the plugin with attributes

Next, you'll update the plugin call to pass in a name attribute. The code in the page template changes as follows:

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
  <meta http-equiv="content-type"
content="text/html; charset=utf-8" />
  <link rel="stylesheet" href="<txp:css />"
type="text/css" media="screen" />
  <title><txp:page_title /></title>
</head>
<body>
<txp:rss_hello_world name="Rob" />
</body>
</html>

```

This code results in a new page, as shown in Figure 14-5.



Figure 14-5. Basic plugin output with the name attribute specified

Since you now specified the name attribute, the default value has been overridden by the value you passed in when calling the plugin.

Calling the plugin with incorrect attributes

As discussed earlier, the plugin code validates the attributes passed in to the plugin. If an invalid attribute is used, an error is shown in your browser window. To demonstrate this, change the plugin to include an invalid attribute called `firstname`. The new page template code changes as follows:

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
  <meta http-equiv="content-type"
content="text/html; charset=utf-8" />
  <link rel="stylesheet" href="<txp:css />"
type="text/css" media="screen" />
  <title><txp:page_title /></title>
</head>
<body>
<txp:rss_hello_world firstname="Rob" />
</body>
</html>

```

This code results in a new page, as shown in Figure 14-6.

```

tag_error <txp:rss_hello_world firstname="Rob"/> -> Textpattern Notice: Unknown tag attribute: firstname on line 582
C:\apache2triad\htdocs\txpbase\textpattern\lib\txplib_misc.php:582 trigger_error()
C:\apache2triad\htdocs\txpbase\textpattern\plugins\rss_hello_world-0.1.php:28 latts()
C:\apache2triad\htdocs\txpbase\textpattern\publish.php:958 rss_hello_world()
processtags()
C:\apache2triad\htdocs\txpbase\textpattern\publish.php:917 preg_replace_callback()
C:\apache2triad\htdocs\txpbase\textpattern\publish.php:453 parse()
C:\apache2triad\htdocs\txpbase\index.php:34 textpattern()

Hello Textpattern World, my name is Anonymous

```

Figure 14-6. Basic plugin output with the invalid firstname attribute specified

In this case, the error message displayed is very clear. You have specified an unknown tag attribute called `firstname` when calling the plugin. The rest of the page traces the error back to its origin. The important part of this message is the second line, which tells you that the error occurred on line 28 of the `rss_hello_world` plugin when calling the `latts()` function. The error messages displayed by Textpattern are very helpful when debugging and testing plugins.

Plugin errors

Aside from the error message you just received, any error you encounter in your plugin code is displayed in your browser window. For example, if you hadn't properly ended a line of code with a semicolon, you would see an error similar to the one shown in Figure 14-7.

```

Parse error: parse error, unexpected '}' in C:\apache2triad\htdocs\txpbase\textpattern\plugins\rss_hello_world-0.1.php on line 31

```

Figure 14-7. A plugin parse error from a plugin in the Plugin cache directory

TEXTPATTERN SOLUTIONS: PHP-BASED CONTENT MANAGEMENT MADE EASY

The benefit of coding and testing the plugin while it is in the Plugin cache directory instead of installed into Textpattern as a standard plugin is that you receive a much clearer error message. If you were to install the plugin, instead of leaving it in the Plugin cache directory, the error would look as shown in Figure 14-8.

```
Parse error: parse error, unexpected ')' in C:\apache2triad\htdocs\txpbase\textpattern\lib\txplib_misc.php(512) : eval()'d code on line 7 The above errors were caused by the plugin:rss_hello_world

tag_error <txp:rss_hello_world name="Rob"/> -> Textpattern Warning: unknown_tag: rss_hello_world on line 968

C:\apache2triad\htdocs\txpbase\textpattern\publish.php:968 trigger_error()
processtags()
C:\apache2triad\htdocs\txpbase\textpattern\publish.php:917 preg_replace_callback()
C:\apache2triad\htdocs\txpbase\textpattern\publish.php:453 parse()
C:\apache2triad\htdocs\txpbase\index.php:34 textpattern()
```

Figure 14-8. A plugin parse error from an installed plugin

As you can see, you no longer have the same level of visibility in the error message. While the first message gave you the exact line number where the error occurred, the most you learn from the second message is that the error came from somewhere within the `rss_hello_world` plugin. The second message is much less specific than the first message you received because the plugin is being loaded from the database instead of the file system.

Debugging

During the course of development, it is probably necessary to inspect the contents of variables within the code that would not normally be part of the plugin's output. Textpattern makes this easy with the convenient `dmp()` function. Based on the plugin code you wrote earlier, you could dump the attributes passed in to the plugin to the screen by using the following code:

```
dmp($atts);
```

That code results in the following output:

```
array (
  'name' => 'Rob',
)
```

Compiling and releasing the plugin

Now that the plugin is successfully written and tested in the Plugin cache directory, it is time to make it available for others to download. There are two ways in which most plugins are made available to others in the Textpattern community.

Since you've been working on the plugin in the Plugin cache directory, start by copying the `rss_hello_world-0.1.php` file back into the workspace at `c:\txp`. You now have three files in the directory: `zem_tpl.php`, `classTextile.php`, and `rss_hello_world-0.1.php`. The easiest way to make the new plugin available for download is to copy all the plugin files in the local workspace directory to a public directory on a website. Once the files are there, point the browser to the plugin file. The output in the browser window is the compiled plugin, as shown in Figure 14-9.

```
# rss_hello_world v0.1
# A basic Textpattern plugin.
# Rob Sable
# http://www.wilshireone.com/

# .....
# This is a plugin for Textpattern - http://textpattern.com/
# To install: textpattern > admin > plugins
# Paste the following text into the 'Install plugin' box:
# .....

H4sIAAAAAAAAA3VRW2vCMBR+bn5FCEIVpBcvtaZ08G3P22CPkqbpGmibkKTrRPzvO1EHuzDI
w7l+lxNG05SeLV1R0rNOKMLSdE2JsfbYiLZVx1GZtvL1DSXvwlpep8tKUMi1EcZJWxwjTI+
2VLypEr8zMr2hpV8dY+Dkb6yAJzGOU3jeBzHaJStbaQRqhcRV118XUopqYt1Rmp3p/NbB1wy
Kz1+ER9OM+eE6bFuhzfZR34EHLiTB1ZJE0hzSgBPG2EtKUqa3sSBj338Pra61TxAdr5RsGvS
/aN3Pv9B9OrPsIuhiWBG718aaTE89r+cXaxhGCF8V8dVdb/vAojqoeHf516ukEUOWMn1EA
kIZxN20PUJkyY9hpicIq9B8V4oc9Dg+96k+dGmw4R8FsfIudFcCjJXADaAmvXv5amePuhD2Q
NxFGEx8W6ILuf9tV62sESpNsK+pN1fN6y1NRL9ciE1nNqyVjizJYe7yCX80dIFFAgAA
```

Figure 14-9. The compiled plugin in a browser window

At this point, anyone can copy the contents of the browser window, paste them into the Install plugin textarea on the Admin ► Plugins tab, and click the Upload button to start the plugin install process. After uploading the plugin, the plugin code and Textiled help text can be previewed, as shown in Figure 14-10. Complete the plugin install process to see how the new plugin will look when it is installed in Textpattern.

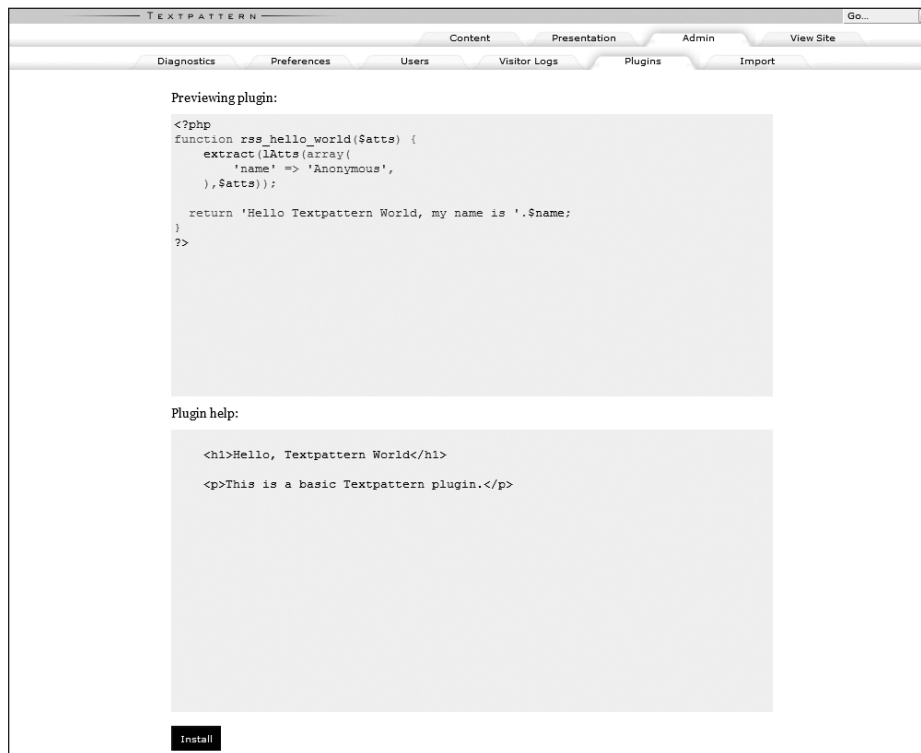


Figure 14-10. Plugin preview while installing the rss_hello_world plugin

After completing the install, the plugin must be activated. Once the plugin is active, it is now available for use.

New plugin installed in Textpattern

Now that you installed the new plugin, it is shown on the Admin ► Plugins tab. In Figure 14-11, you can see that `rss_hello_world` is the only installed plugin.

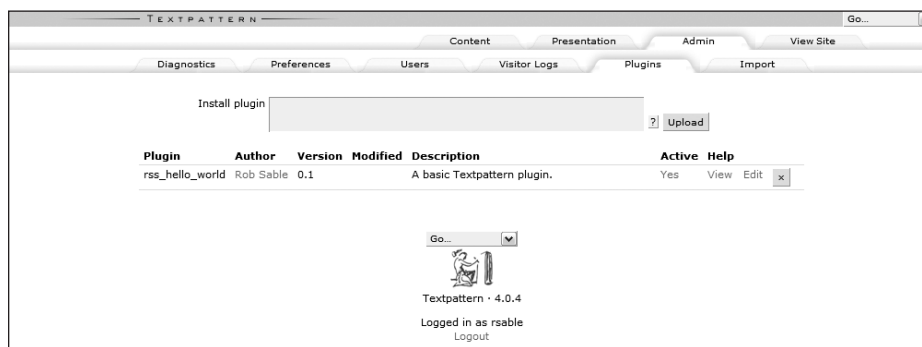


Figure 14-11. The `rss_hello_world` plugin after installation

The metadata you defined in the plugin template—including the plugin name, description, version, and author—translates directly to the information you see on this tab. The author name has also been hyperlinked using the URL that you provided in the template.

Viewing plugin help

The help text that you entered into the plugin template can be viewed (see Figure 14-12) by clicking the View link in the Help column.

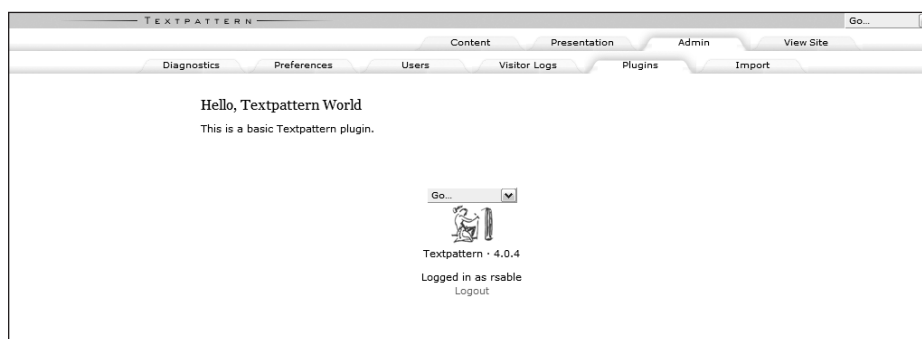


Figure 14-12. The `rss_hello_world` plugin help

While the `rss_hello_world` plugin doesn't have extensive help, you have the ability to add as much as you need to your plugin help. You also have the ability to add any HTML formatting necessary to make your help text easy to read and understand.

A basic enclosing plugin

Now that you created the first self-closing plugin tag, adapt the example to become an enclosing tag. In addition to converting `rss_hello_world` to an enclosing plugin, you can keep its capability to be a self-closing plugin as well. Here's the new code:

```
function rss_hello_world($atts, $thing = NULL) {
    extract(lAtts(array(
        'name' => 'Anonymous',
        'message' => 'Hello Textpattern World, my name is',
        'wraptag' => 'p',
    )), $atts);

    if ($thing === NULL) {
        return doTag($message.' '.$name, $wraptag);
    }

    return doTag(parse($thing).' '.$name, $wraptag);
}
```

The first change is to add the second parameter, called `$thing`, to the function. The addition of this second parameter is needed to make the plugin an enclosing tag. When a page is rendered, Textpattern assigns the content between the opening and closing tags of the plugin to the `$thing` variable.

Notice a few other additions to the plugin. You now have two additional attributes available when calling the plugin. The first is called `message` and is used to display default text when the tag is called as a self-closing tag. The second is called `wraptag` and is used to wrap the message in an HTML tag.

If you want to force the plugin to be called as an enclosing plugin, you can return an error message when the `$thing` variable is NULL. But to demonstrate both the self-closing and enclosing varieties of the plugin, update the page code as follows:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
    <meta http-equiv="content-type"
    content="text/html; charset=utf-8" />
    <link rel="stylesheet" href="txp:css />
    type="text/css" media="screen" />
    <title>txp:page_title /></title>
</head>
<body>
<txp:rss_hello_world wraptag="strong"/>
<br/>
<txp:rss_hello_world name="Rob">
What's up world, I'm
</txp:rss_hello_world>
</body>
</html>
```

The updated page template generates output as shown in Figure 14-13.

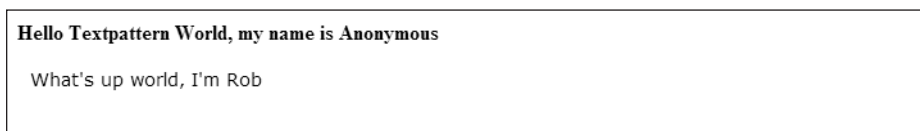


Figure 14-13. Plugin output as both a self-closing and an enclosing tag

In the first call to the plugin, you use a self-closing tag and specify the `wraptag` attribute, which gives you the default message wrapped with `` tags. In the second call to the plugin, you use enclosing tags and specify the `name` attribute. In this case, the name and message specified are displayed wrapped with the default `<p>` tags.

Advanced plugin topics

Now that you know the basics of writing plugins, you can move on to some more-advanced topics. You'll learn that there are endless ways in which you can enhance and extend Textpattern's core. You'll explore some examples with the concepts covered in this section.

Conditional tags

A **conditional tag** is a more-advanced form of enclosing tag because it enables you to control the execution of one piece of code when the condition is `true` and another piece of code when a condition is `false`. You can think of a conditional Textpattern tag as a typical `if/else` programming statement. There is a variety of conditional tags built in to the Textpattern core. You can easily find these tags as they will begin with `<txp:if`.

To demonstrate a simple conditional tag, create the `rss_if_positive` tag, which accepts a number as a parameter and performs a conditional check to determine whether the number is positive. The code for the tag is the following:

```
function rss_if_positive($atts, $thing) {
    extract(lAtts(array(
        'number' => 0,
    ),$atts));

    $condition = ($number > 0) true : false;
    return parse(EvalElse($thing, $condition));
}
```

The magic here happens in the `EvalElse()` function, which you can find in the `/textpattern/lib/txplib_misc.php` file. After evaluating the condition, you let Textpattern know whether it was `true` or `false` so the appropriate output can be parsed and displayed. Call the plugin as follows:

```
<txp:rss_if_positive number="1">
It's positive!
<txp:else />
It's negative!
</txp:rss_if_positive>
```

In this case, the phrase `It's positive!` is included in the page output because the condition is true. If, however, you changed the tag call as follows, the output would change to `It's negative!` because the condition would be false:

```
<txp:rss_if_positive number="-1">
It's positive!
<txp:else />
It's negative!
</txp:rss_if_positive>
```

Callback functions

A **callback function** is a function that is passed as an argument to another function. The callback function is then executed at some point by the function it was passed to. This powerful concept enables plugin authors to write code that will be executed by Textpattern based on certain events. There are several callback functions available on the public side and even more available on the admin side.

The `register_callback($func, $event, $step='', $pre=0)` function is located in the `/textpattern/lib/txplib_misc.php` file. The `$func` and `$event` arguments are required, while `$step` and `$pre` are optional for admin-side callbacks. The functions arguments are used as follows:

- `$func` is the function you want Textpattern to call. This will be where the bulk of your plugin code is located.
- `$event` is the Textpattern event that will call back into your function.
- `$step`. On the admin side, an event might have multiple steps. You can target specific steps within an event for a finer level of control.
- `$pre`. On the admin side, if `$pre` is set to 1, the callback function will be called before the page is rendered instead of after.

When you register a callback function, you have the opportunity to completely override base Textpattern functionality or add your own features on top of the Textpattern core.

Public-side callback events

Public-side events are not as easy to recognize as admin-side events because they are buried in the Textpattern source code. The callback events currently available on the public side are these:

- `pretext` is called at the beginning of the `pretext()` function, which parses the URL to initialize variables used to build the page.
- `textpattern` is called at the beginning of the `textpattern()` function, which builds the pages based on values set in the `pretext()` function.
- `comment.form` is called before the comment form is added to a page.
- `comment.save` is called before a comment is saved to the database.
- `rss_entry` is appended to the end of each entry in the Rich Site Summary (RSS) feed.
- `atom_entry` is appended to the end of each entry in your Atom feed.

The `pretext` and `textpattern` callback events can be used to override the standard processing in the `pretext()` and/or `textpattern()` functions. This process is commonly used to override Textpattern's built-in URL handling to enable additional URL schemes to be supported. For example, the `rss_suparchive` plugin uses this technique to support date-based archives, and the `rss_unlimited_categories` plugin uses it to support `/section/category/title` URL schemes.

The `comment.form` and `comment.save` events are primarily used to combat comment spam. For example, the `mrw_spamkeywords_urlcount` plugin analyzes keywords and link patterns in comment text to prevent spam from being posted and saved to the database.

To register a callback function with one of these public-side events, you need to add the following to your plugin code:

```
register_callback('rss_my_function', 'pretext');
```

Just replace the first argument with your function name and the second argument with the event to which you want to attach your callback. You typically find callbacks declared at the top of plugin's code, where they're easy to spot.

Admin-side callback events

The events and steps available on the admin side are much easier to find. In fact, you've been looking at them since the first time you logged into the admin interface, but you might not have known it. For example, take the URL of the [Content ► Write](#) tab, which ends in `textpattern/index.php?event=article`. You can clearly see from the URL that the `article` event is being used to identify this tab. If you pull up an article that you already saved in order to make changes, notice a URL ending in `/textpattern/index.php?event=article&step=edit&ID=nnn`. Again, you can see that the `article` event is being used, but you now have the addition of the `edit` step.

To register a callback function to be executed when you edit an existing article, make a call as follows:

```
register_callback('rss_my_admin_function', 'article', 'edit');
```

In addition to the step name that can be determined from looking at the URL, you can also determine step names by inspecting the admin interface source code. Each admin-side event is handled by a PHP script located at `/textpattern/include/txp_EventNameGoesHere.php`. For example, the `article` event is handled by a script located at `/textpattern/include/txp_article.php`. If you look at the top of that script, you'll find the following switch statement:

```
switch(strtolower($step)) {
    case "":      article_edit();  break;
    case "create": article_edit();  break;
    case "publish": article_post(); break;
    case "edit":  article_edit();  break;
    case "save":  article_save();  break;
}
```

By reading through the switch statement on the `$step` variable, you can see that there are steps called `create`, `publish`, `edit`, and `save` that are handled by the `article_edit()`, `article_post()`, and `article_save()` functions (also located in the `txp_article.php` file). To have a callback function executed when a new article is posted, the following call should be made:

```
register_callback('rss_my_admin_function', 'article', 'publish');
```

This type of call is used by the `rss_unlimited_categories` plugin to save and retrieve categories from the database when articles are posted and edited. You'll learn more about that plugin later in this chapter.

All admin-side events and steps can be used to register callback functions with the exception of the `plugin` event. The `plugin` event is used to handle the `Admin > Plugins` tab and does not load plugins and therefore register for callbacks. This was done to allow for the deactivation of any plugins that might be misbehaving.

Admin-side tab registration

In addition to adding your own callback functions on the admin side, you also have the ability to add new tabs to the admin interface that will appear under the extensions menu (see Figure 14-14).

The new tabs can be used for any number of purposes, including setting preferences for a public-side plugin, managing and backing up your MySQL database, manipulating images, and customizing the look and feel of the admin interface.



Figure 14-14. Admin plugins that have registered new tabs under the Extensions tab

To register a new tab, call another function in the `/textpattern/lib/txplib_misc.php` file called `register_tab($area, $event, $title)`. After registering your new tab, you then register a callback function to handle the event for the new tab. The code that creates the tabs for the `rss_admin_db_manager` plugin (refer to Figure 14-14) is as follows:

```
if (@txpinterface == 'admin') {
    register_tab("extensions", "rss_db_man", "DB Manager");
    register_callback("rss_db_man", "rss_db_man");

    register_tab("extensions", "rss_sql_run", "Run SQL");
    register_callback("rss_sql_run", "rss_sql_run");

    register_tab("extensions", "rss_db_bk", "DB Backup");
    register_callback("rss_db_bk", "rss_db_bk");
}
```

The code is wrapped by an `if` statement, which ensures that the code is executed only within the admin interface. The arguments passed in to the `register_tab()` function are used as follows:

- `$area` is the top-level tab under which your new tab will be created. Plugins are typically added under the Extensions tab.
- `$event` is the new event that will be used for your tab.
- `$title` is the display name for the new tab.

After registering the new tab for a particular event, a callback is registered to handle that event.

Helper functions and global variables

In the examples presented throughout this chapter, you used several functions from the Textpattern source code. As plugins are loaded and executed, you'll have access to any and all functions and global variables that have been defined in the Textpattern source.

There is a variety of common functions that can be used for anything from querying the Textpattern database to generating HTML output. The more time you spend familiarizing yourself with the code that others have already written, the less time you'll spend writing your own code.

As your pages and forms are parsed and rendered, you'll also have access to global variables that can be used to generate custom output. Global variables are used to hold everything from general site preferences and configurations to page and form specific settings.

Appendix B contains a detailed listing of commonly used helper functions from the Textpattern source and the global variables that you'll have access to.

Real-world examples

Now that you know how to create Textpattern plugins, it is time to take a look at a few actual plugins to show how they can be used to enhance your Textpattern sites. One of the best ways to learn how to write plugins is to examine and learn the techniques of other plugin developers. The plugins listed cover a wide range of possibilities to get you started. All the plugins can be found on the Textpattern Resources site at www.textpattern.org or at the address noted in the following sections.

rss_unlimited_categories

This plugin contains examples of all the concepts covered in this chapter. It creates and uses its own database table, which enables you to attach an unlimited number of categories to an article above and beyond the standard two categories that Textpattern provides.

On the admin side, the plugin uses callbacks to enable you to set and edit article categories. It also registers its own admin-side tab that enables you to set preferences for the plugin.

On the public side, the plugin also uses a callback to support the `/section/category` and `/section/category/title` URL patterns. There is a collection of public-side tags of the self-closing, enclosing, and conditional variety.

www.wilshireone.com/textpattern-plugins/rss-unlimited-categories

rss_thumbpop

This plugin generates several different image gallery formats. The typical layout includes a listing of image thumbnails that display a full-size image when clicked. The gallery includes configurations that enable it to display the full images in a pop-up window, with or without a caption, or on the same page using JavaScript.

www.wilshireone.com/textpattern-plugins/rss-thumbpop

rss_auto_excerpt

This plugin automatically generates an article excerpt based on the number of characters, words, sentences, or paragraphs that you specify.

www.wilshireone.com/textpattern-plugins/rss_auto_excerpt

rss_admin_db_manager

This admin-side plugin adds three tabs to the admin interface that enable you to manage database backups, maintain your database tables, and execute SQL queries.

www.wilshireone.com/textpattern-plugins/rss_admin_db_manager

glx_admin_image

This admin-side plugin adds a wealth of helpful features to the Content ► Images tab, including the ability to rotate images and automatically create thumbnails.

http://grauhirn.org/txp/12/glx_admin_image_resize

ajw_if_comment_owner

This plugin enables you to generate different output if the current comment was posted by the owner of the site. It is commonly used by site owners to style their comments differently from others.

<http://compooter.org/2005/03/textpattern-plugin-ajw-if-comment-owner>

zem_contact_reborn

This plugin is one of the most widely used and can help you create anything from a simple contact form to a complex registration form.

<http://thebombsite.com/txplugins/408>

Summary

This chapter covered everything you need to know to start writing your own Textpattern plugins. While the Textpattern core offers a great set of basic features, the plugin framework enables you to extend its functionality for whatever you need. If you're interested in writing your own plugins, make sure that you start from the basics. Getting an appropriate workspace set up and learning how best to use the helper functions and global variables that Textpattern offers will save you time in the long run. And don't forget to check the Textpattern Resources site at www.textpattern.org to find other plugins that you can learn from.

Now that all the building blocks that make up the Textpattern system have been discussed, the next three chapters explore the creation of professional websites that were built with Textpattern. All the sites are completely different and demonstrate how flexible Textpattern really is.