11

Building Viral Marketing Tools

One of the best ways to spread the word about your website is to let your audience do it for you.

Search is the primary method by which people discover the content they're looking for, and so we optimize our websites for search engines. But sometimes people may want content that they don't even know exists. It's pretty hard to search for something you've never heard of.

We probably all have a story about a movie, band, website, or product that someone recommended, which turned out to be really amazing. Although it wasn't something you were actively searching for you were really happy you found it. Most of the time it's a friend, colleague, or family member who brings our attention to an unknown desire. Word-of-mouth recommendations fill a findability gap that search doesn't. The Web amplifies word-of-mouth recommendations, which makes it the perfect place to spread your message to a wide audience. With a little knowledge of basic psychology and the right tools, you can persuade your users to spread your marketing message like a virus from one person to the next.

A Virus You'll Want To Spread

Viruses are simple yet brilliant organisms. Their one goal in life is to locate hospitable hosts to pass on their genetic code. Once the host is infected, it will in turn pass the genetic code on to others. It's a clever system that can create exponential growth in a very short amount of time.

Ideas can behave in a similar fashion, traveling from one host to the next conveyed by word of mouth. In 1996 Jeffrey Rayport applied the concept of self-propagating viruses to marketing in his article in *Fast Company* magazine entitled "The Virus of Marketing" (**http://www.fastcompany.com/online/06/virus.html**).

Viral marketing is simply network-enhanced word-of-mouth advertising. Following a traditional direct-marketing model a company communicates directly to many consumers, as **FIGURE 11.1** shows.

Viral marketing minimizes direct company-to-consumer communication and instead relies on consumers to spread the message to people they know. This model has the potential to reach more people with greater speed than traditional direct marketing because the communication load is distributed to many people.

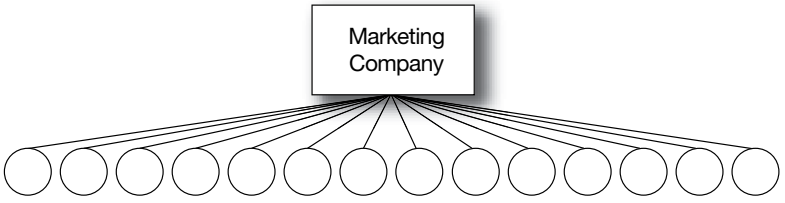

FIGURE 11.1 *Traditional marketing techniques require that the message be communicated to each consumer directly.*

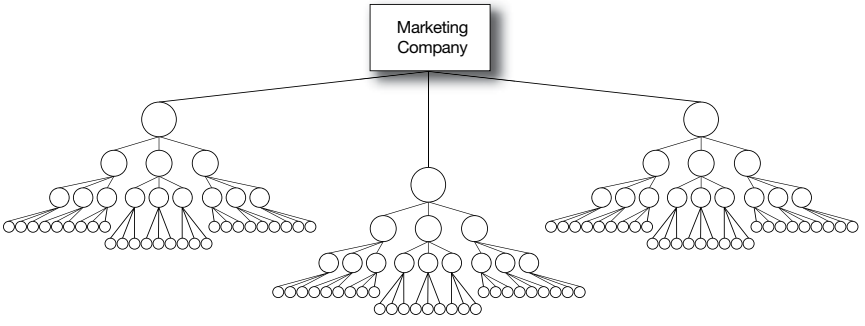

FIGURE 11.2 *Viral marketing spreads a message quickly by communicating to a few consumers who then feel compelled pass it on to others.*

As **FIGURE 11.2** shows, with viral marketing one person receives a message directly from a company, then tells other people, who in turn tell still more people. As the process repeats, the message spreads to many people with little effort from the original source.

Viral messages also tend to be more effective because they come from a trusted source, and don't appear to be a marketing message with ulterior motives. When friends, family, or colleagues tell you about a product or service, you're more likely to trust and take their advice than advice from an advertisement.

Viral campaigns perform exceptionally well on the Web because people are already connected. Email, social networking sites, and the various other communication conduits of the Web act as giant amplifiers of viral marketing messages that help them reach new people at staggering speeds. When you introduce an interesting idea, product, or service to the Web and provide the tools to let it spread, you can quickly build your audience base.

There are many techniques you can employ to make your content spread virally. In this chapter we'll take a look at what motivates users to spread viral messages, and build some viral marketing tools that can easily be added to any site.

Take a look at Kent Lewis' article on SitePoint entitled "Get the Bug: Viral Marketing Unmasked" to learn about the history of viral marketing and some common traits of successful campaigns (http://www.sitepoint.com/article/get-the-bug-viral-marketing).

Passing the Virus to New Hosts

Compelling content is at the root of any successful viral marketing campaign. Whether your content is a funny video, a cool product, or an interesting article, viral marketing campaigns have to tap into your users' desires and motivations in order to spread to new hosts. If your content is valuable, compelling, or entertaining, users will want to pass the message on for you.

YouTube's Simple Brilliance

Successful viral marketing campaigns also include a clear call to action to encourage users to pass content on to others. YouTube (**http://youtube.com**) does a nice job of this by displaying a "share" button on the screen after a video has played so users can easily tell their friends about it (see **FIGURE 11.3**). YouTube's call to action is effective because it is direct, brief, obvious, and displayed when users are most likely to take action.

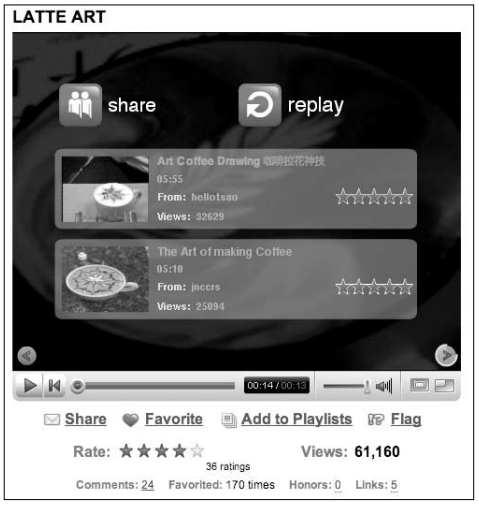

FIGURE 11.3 *YouTube (**http://youtube.com**) encourages users to tell others about their content with a "share" button displayed at the end of each video. This clear call to action fosters the viral exchange of content and increases traffic.*

It's important that your viral marketing tools make it effortless for your message to spread. When users click YouTube's "share" button to email friends about a video, a form is displayed with a quick, pre-populated message field so users can simply add an email address and send the message with little effort (see **FIGURE 11.4**).

The message itself is also well crafted, though it's only three words long. The phrase "This is awesome!" is informal, and sounds like something a user

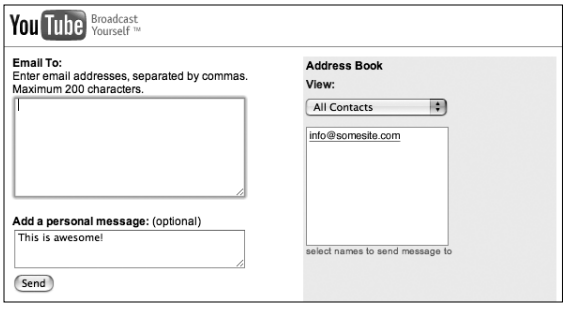

FIGURE 11.4 *YouTube (**http://youtube.com**) provides an efficient system for sharing videos with others. You can enter a series of email addresses or pick from those you've already sent messages to.*

would write, not a corporation. In order for viral messages to work they need to look like they are coming from a host (user), not from a company.

An address book is displayed on the side so you don't have to search for and type in the email addresses of people you've already sent videos to. The address book is quite brilliant because the more users share videos the easier it becomes to share them in the future. Because Web users move fast and are easily distracted, an efficient and brief sharing system is essential.

Successful viral marketing campaigns take into consideration what motivates users to spread a message, and try to use this to their advantage.

Motivating Users To Spread the Virus

Users from all demographics are commonly motivated to share content that

- is free or a special promotion
- is funny, entertaining, or unique
- boosts their ego or helps build their public identity
- rewards them for spreading the message
- serves their desires
- lets them share their perspective

Let's take a closer look at each of these common motivations to share content and see some examples of viral marketing campaigns that have made effective use of them.

Free Stuff and Special Promotions

Hotmail (**http://hotmail.com**) was one of the first free Web email services, and it grew to be one of the largest in a very short period of time. The Hotmail business model was simple and brilliantly viral. They provided a free email

service to attract new users, then included a line at the bottom of each email message that read "Get your private, free email at **http://www.hotmail.com**." With each email users sent they marketed to every recipient. This generated new signups that in turn began emailing the message to even more new users. The exponential spread of the message generated roughly 8.5 million users in about 18 months. The Hotmail marketing message was disseminated at a staggeringly fast rate that would have been impossible to achieve using traditional direct marketing.

You can apply the Hotmail viral marketing technique to all kinds of content and services. Simply give something valuable away for free and provide users a convenient method to tell others about it. This will bring in new users to your site who are likely to browse other sections and possibly complete one of your business goals, such as sign up for a mailing list, learn about your cause, or buy a product.

Of course sales, coupons, and great deals can work the same way. Ecommerce sites and online services can extend special promotions to users on a mailing list. These users can then extend the offer to friends and family. Netflix (**http://netflix.com**)—the popular online DVD-rental service—uses this technique to build their customer base. They send coupons to all of their customers to give one month of free rentals to friends and family. This technique also makes new customers feel special because the offer is exclusive.

Funny, Entertaining, or Unique

People love to share entertainment. The feelings invoked by funny, entertaining, or unique content are amplified and extended when you share it with others. After you email a friend a link to a funny video on YouTube, there's usually a good deal of equally entertaining discourse that follows. If the friend who received the link liked it, they're likely to pass it on to someone else.

This plays into a basic principle of humanity. There are certain experiences that are more potent when shared, and humor is one. JibJab (**http://jibjab.com**), a popular Flash cartoon site that pokes fun at politicians and current events, uses this principle to its advantage by encouraging users to tell others about their cartoons.

Users can share the laughs with their friends on social networking sites, send out emails to get others to check out cartoons, or even post cartoons on their own site. Some of their cartoons even allow users to become the star by uploading their picture and placing it on a character's body, as shown in **FIGURE 11.5.** Users can then add pictures of friends and family to flesh out the

FIGURE 11.5 *JibJab's (**http://jibjab.com**) Starring You cartoons let users map pictures of themselves, friends, and family into a humorous Flash-based narrative. Once you build your cartoon you can publish it on various social networking sites, email it to friends, or embed it in a page on your site. This cartoon stars me as a heavily armed, overzealous action hero ready to kick some butt. That's not so funny, right?*

cast of the story. This type of personalized comedy is a brilliantly unique way of engaging an audience, and inherently encourages users to share the content with others.

Boost the Users' Egos or Help Build Their Online Identity

Content and applications that feed users' egos encourage viral exchanges. The JibJab Starring You cartoon shown in Figure 9.5 is a great example of an ego booster. People get a kick out of seeing themselves and will want to show others the cartoon they've created with their picture.

It's especially alluring to share this ego-boosting cartoon on social networking sites like MySpace (**http://myspace.com**) or Facebook (**http://facebook.com**) where users construct profile pages and build their online identity. Though everyone else might have the typical fare on his or her Facebook page, you could feature your own custom cartoon of which you are the star.

The *Simpsons Movie* website (**http://www.simpsonsmovie.com**) launched an equally successful viral marketing campaign that lets users create a Simpsons-style avatar of themselves. Once you've created your Simpsons avatar you can email it to a friend, or download it so you can use it on other sites (see **FIGURE 11.6**). When the *Simpsons Movie* site was launched, yellow avatars popped up on websites and instant messaging clients everywhere, creating an immediate viral awareness of the movie. The portability of avatar images was the key to spreading the message.

FIGURE 11.6 *The* Simpsons Movie *website (**http://www.simpsonsmovie.com**) has an avatar design application that lets users create a Simpsons-style avatar of themselves. Upon its release users began posting their avatars on other websites, contributing to the buzz around the movie.*

Developing with the Facebook Platform and PHP
http://www.sitepoint.com/article/developing-facebook-platform

Facebook applications are another great way to spread your content virally by helping users build their online identity. With 35 million users and counting as of October 2007, Facebook has become a central collection point for content mashups. Facebook has created an API (Application Programming Interface) that lets developers create modules that move content from other Web application to Facebook. Users can add these application modules to their profile to combine their Flickr photos, Last.fm playlist history, Netflix movies, and much more into one central, online identity.

Check out the official Facebook developer's site for the documentation and tools you'll need to build your app http://developers.facebook.com/.

All of this content from different Web applications says something about the user. If your site provides services that let users create their own content, consider making it portable to Facebook so they can display it there too. When users install Facebook applications they're prompted to invite friends to contribute or install the application in their own profile. These applications are inherently viral, and can spread your content very quickly. In a short period of time you can generate many new users for your Web application.

Reward Viral Behavior

As history has continually shown us, bribery is an effective tool to get people to do something for you. It's also a clever tool to encourage the viral transfer of your message to others. That probably sounds a bit sketchy, but but it's not quite as bad as it might sound.

When your audience shares your content with others you can provide various rewards to thank them for their efforts. Amazon has done a brilliant job of this through their affiliates program (**https://affiliate-program.amazon.com**). When you sign up for the program you get a code to include within links to any product Amazon sells. Each time a user clicks the link to buy that product on Amazon, they log your code and share a percentage of the revenue earned from sales you've generated.

It's bribery of the moral kind in which all parties get something they want. Users who promote Amazon products make money, and Amazon dramatically increases its sales as well as its presence throughout the Web.

Media Temple (**http://mediatemple.com**)—a popular Web hosting company—has also tapped into the same principle with great success. With slick, glossy branding and advanced hosting features, Media Temple quickly caught on as the serious Web designer's preferred hosting company. You'll often find a Media Temple banner or logo on high-end Web design sites promoting the company, and providing the Web designer with affiliate revenues in the form of free hosting.

Media Temple's affiliates program has become so effective that its promotional banners are often seen as a badge of quality worn by those in the Web design "in" crowd. It rewards affiliates monetarily and builds their public image at the same time.

Serve Your Users' Desires

You can dramatically increase product sales on your site if you just help people communicate their desires. People these days aren't too subtle with their gift wish lists. When people find something they want, they tell their friends and family about it in hopes that they'll receive it when the next gift-giving occasion arises.

Make it easier for people to solicit gifts with a "Tell a Friend" feature. This is simply a form that allows users to email anyone a message and a link to the products they want. Tell a Friend messages are very likely to generate a sale because they come from someone the recipient knows, and at the same time expose new users to your site. Later in this chapter, we'll learn to build a tool like this that could be integrated into any site.

If you'd like to learn how to publish any content within a database to an RSS feed, check out Kai Blankenhorn's PHP class http://www.bitfolge.de/rsscreator-en.html.

Lists are another powerful tool that lets users communicate their desires. Wish lists and lists of favorite products make it easy and convenient for users to make a collection of many things they want. Publish an RSS feed of the list and users can then display the things they want on their personal website or blog.

Gift registries have a similar affect. When users assemble a gift registry they'll probably tell all of their friends and family about the list, which could potentially create a windfall of sales. It also exposes your site to many new users who may return in the future to buy more gifts or something for themselves.

If your site has a gift registry system, provide an email tool with which users can enter a delimited list of addresses to notify all of their friends and family with one quick message. You might also provide bundles of registry notification postcards you can mail to registrants upon request. These can be slipped into wedding or shower invitations to conveniently notify all guests of the gift registry. Your users will appreciate the thoughtful gesture that makes their life easier, and you'll enjoy the sales it generates.

The return on investment for a gift registry is really amazing. Although it may cost you thousands of dollars to develop a gift registry system, it may only take a couple of filled registries to make back your investment. Also, consider the amazing rate of message dissemination. One gift registrant can spread awareness of your site and its products to hundreds of people very quickly.

Let Your Users Share Their Perspectives

Chapter 4 presented a hypothetical story of a user named Tom, who was searching for guidance on how to set up a home theater system. In this story Tom found the answers he needed and the products for his dream system all on the same site. User reviews of each product helped inform his buying decisions, and inspired confidence in his purchases. Tom later returned to the site to write his own glowing reviews of the products he purchased, helping reassure other users about the quality of the products.

Although this story is intended to illustrate how quality content can improve findability in various ways, it also speaks to a basic principle of human nature. When people truly believe in an idea or product, they want to tell others. Product reviews let users evangelize the positive traits of your products. Users are more likely to take to heart a glowing product review written by a fellow shopper than one written by a member of your company. Although you may have ulterior motives to talk up your products, your users probably don't.

User reviews don't have the message mobility that a Tell a Friend system might, but they can certainly increase sales. Shoppers who may be equivocating over a product can be swayed to buy when a positive user review dispels doubts.

Be warned that this is a double-edged sword. If your products don't live up to consumer expectations, a user-generated review system can also sway shoppers to not buy. A small number of negative reviews among many positive ones can actually help you, though, as it lets users know you're not filtering out all of the negative comments.

Later in this chapter we'll see how to build a simple product review system that will let help you market your product. Now that you've got a foundation in the concepts of viral marketing, let's explore practical examples that could be incorporated into any website.

Building a Viral Product

Any of the viral marketing techniques discussed thus far could be implemented independently or in synchrony in your website to motivate your users to spread your message. Making it easy for your audience to tell others about your content, products, or services can increase traffic to your site, and inspire trust in your content.

Let's put some of this theory into practice by building a viral product. Imagine we're creating an ecommerce website for an artist to sell her or his work online. The product page is certainly one of the most important in the site because it's here that users learn about products and decide whether to make a purchase. It's a perfect place to include viral marketing tools.

We'll create three viral marketing tools, each motivating the user in a different way to tell others about the product:

- **Tell a Friend:** lets users communicate their desires to friends and family
- **Product reviews:** lets users communicate their personal perspective about the product and perhaps convince other users to buy
- **Social networking links:** lets users build their online identity by endorsing a product–the products they like says something about their beliefs

Each of these tools reaches new users in different ways. The Tell a Friend system will connect people who know and trust the sender to the products on the

site. The user-generated product reviews will let total strangers inspire trust in the products for all new customers. The social networking links will let users evangelize the product on other sites to both strangers and personal acquaintances alike.

Although we'll be applying these tools to the common scenario of selling products online, they could also be applied to any number of other situations. Once you've built these tools they can easily be ported to other projects with minimal setup time. **FIGURE 11.7** shows the product page we'll build with these three viral marketing tools.

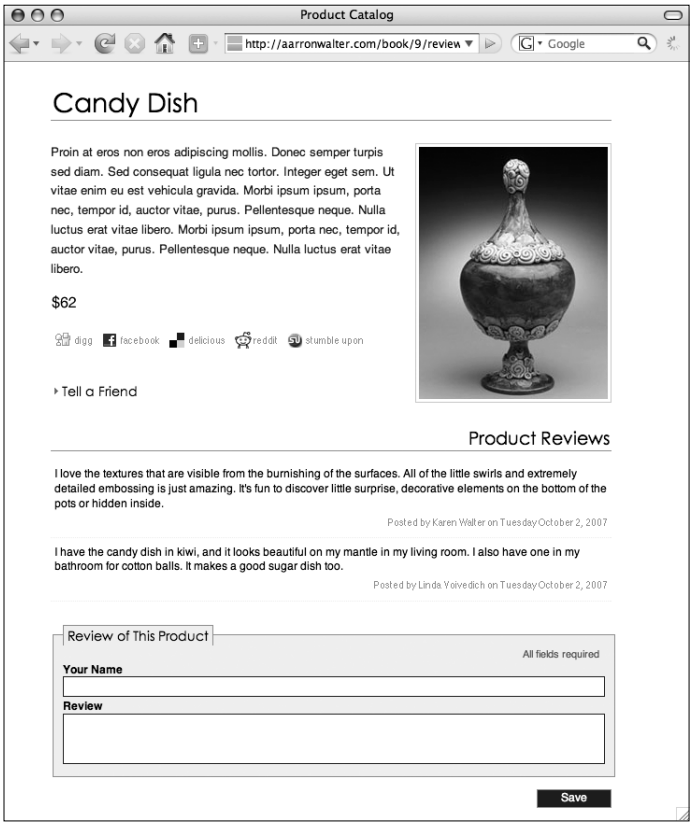

FIGURE 11.7 *Three different viral marketing tools—Tell a Friend, user-generated product reviews, and social networking links—will make it easy for users to tell others about this product.*

The interface for this page can become a bit overwhelming with all three of these viral marketing tools occupying space and vying for the users' attention. To simplify things for users, the Tell a Friend form can be collapsed by default with JavaScript. The arrow next to the label lets users know they can expand the Tell a Friend form when needed.

Since the code to expand and collapse a was already built in Chapter 7 in the section "Solving Scripted Style Problems," we'll skip it here. Because it was built following unobtrusive scripting principles, adding it to this example will only require a link to the external JavaScript file and a call to the function once the page has loaded.

Building a Tell a Friend Feature

The Tell a Friend tool is simply an enhanced contact form that sends an email to a friend along with a link to the product. The expanded Tell a Friend form is shown in **FIGURE 11.8**.

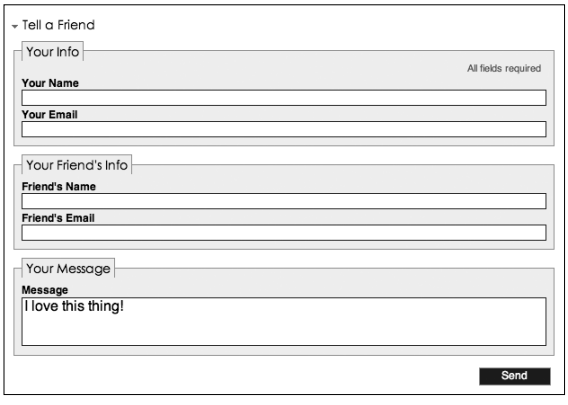

FIGURE 11.8 *The Tell a Friend tool is an enhanced contact form that emails a personal message and a link to the product page. The default message expedites the sending process so users can quickly tell their friends about the product.*

Like the YouTube Tell a Friend form shown in Figure 11.4, this form also includes a default message to expedite the sending process. Making the process faster and easier increases the chances that your users will actually tell their friends about your products. The message is kept informal so it reads like the voice of the user rather than a marketing message.

Building the Form

The first step to building the Tell a Friend tool is to construct the form to send the message. Create a new file called `product.php` with a basic HTML structure, and some product content. Under it add the following form:

```
<form id="tellfriend" action="<?=$_SERVER['PHP_SELF']?>"
method="post">
    <fieldset>
    <div class="notification">All fields required</div>
    <legend>Your Info</legend>
    <label for="name">Your Name</label>
```

```
    <input type="text" name="senders-name" id="sender-name"
value="<?=$_POST['senders-name']?>" />

    <label for="name">Your Email</label>
    <input type="text" name="senders-email" id="sender-email"
value="<?=$_POST['senders-email']?>" />
    </fieldset>

    <fieldset>
    <legend>Your Friend's Info</legend>
    <label for="name">Friend's Name</label>
    <input type="text" name="friends-name" id="friends-name"
value="<?=$_POST['friends-name']?>" />

    <label for="name">Friend's Email</label>
    <input type="text" name="friends-email" id="friends-email"
value="<?=$_POST['friends-email']?>" />
    </fieldset>

    <fieldset>
    <legend>Your Message</legend>
    <label for="message">Message</label>
    <textarea name="message" id="message">I love this thing!
<?=$_POST['message']?></textarea>
    </fieldset>

    <input type="hidden" name="product-id" value="12" />
    <input type="submit" name="submit-friend" value="Send"
class="btn" />
</form>
```

For some guidance on the elements used in this form to make it accessible, read Ian Lloyd's three-part tutorial at http://webstandards.org/learn/tutorials/accessible-forms/.

This form follows basic accessibility standards so the widest audience possible can use it. The `<input />` and `<textarea>` tags are all accompanied by a `<label>`. The three code sections of the form—the sender's info, their friend's info, and the message—are all grouped with `<fieldset>` then labeled with `<legend>`.

Amidst the HTML is interspersed a bit of PHP, which I've highlighted for your reference. The value of the action attribute in the `<form>` tag is dynamically defined with `<?=$_SERVER['PHP_SELF']?>`. This form will submit to itself in order to send the message or report errors. Rather than hard-coding the page's URL I've used PHP to find out the file name. This ensures that should this page be renamed or moved, the script will still work without modification.

Inside the value attribute of each `<input />` and between the `<textarea>` tags PHP will write in the content the user has written once the form has been submitted. This ensures that even after the user has submitted the form they don't lose the text they've written. If the user has made some mistake when filling out the form and the page refreshes with an error message, they won't have to retype everything. It will also expedite sending more messages to other friends because they'll only have to change the text in the friend's name and email fields once they've successfully sent a message.

At the end of the form, directly above the submit button, is a hidden `<input />` field that contains the product id. This is included in the form so the PHP script that will receive the input and send the message can build a link to the product page in the email.

We'll return to this page shortly to add some more necessary elements, but first let's build a PHP function to validate the form input and send the message.

Building the Tell a Friend Function

Create an include folder called `inc`, and inside it create a new file called `tell-friend.php`. The functionality of the Tell a Friend tool is kept in a separate file so it can be easily repurposed in other projects if needed.

The `tell-friend.php` file will contain a single function, which will receive no arguments, and will contain two key sections to do the following:

- Validate all user input
- Send the message

Here's the basic structure to start the function:

```
<?
function tellFriend(){
    // Validate user input
    $err = array();

    // If no errors send message
}
?>
```

Any validation errors will be logged in an array called `$err`. After all of the validation tests have been run, a conditional will count the number of elements in this array to determine if any errors were logged. By default this array is empty.

If it contains more than zero elements we'll know an error was encountered. If the user did make an error by neglecting to fill out a field or by providing an invalid email address, the script will stop and return an error message to the products page.

This user input validation is added to the function directly after the declaration of the `$err` variable. The `array_push()` PHP function is used to append an error message to the `$err` array, should a problem be encountered.

```
// Check for senders name
if( empty($_POST['senders-name']) ){
    array_push($err,"Please include your name");
}

// Check validity of senders email address
if( empty($_POST['senders-email']) ){
    array_push($err,"Please include your email address");
}elseif( !ereg("^[a-zA-Z0-9._-]+@[a-zA-Z0-9._-]+\.([a-zA-
Z]{2,4})$", $_POST['senders-email']) ){
     array_push($err,"Your email address is invalid");
}

// Check for friends name
if( empty($_POST['friends-name']) ){
    array_push($err,"Please include your friend's name");
}

// Check validity of friends email address
if( empty($_POST['friends-email']) ){
    array_push($err,"Please include your friend's email address");
}elseif( !ereg("^[a-zA-Z0-9._-]+@[a-zA-Z0-9._-]+\.([a-zA-
Z]{2,4})$", $_POST['friends-email']) ){
    array_push($err,"Your friend's email address is invalid");
}

// Check for message
if( empty($_POST['message']) ){
    array_push($err,"No message provided");
}
```

A majority of the validation conditionals are simply looking for empty fields. If the user leaves any field blank, an error will be logged. The email addresses are a little more advanced, though, and require a regular expression to evaluate the address for a pattern. As discussed in Chapter 3, regular expressions

are commonly used in many scripting and programming languages to identify patterns in strings.

Since email addresses follow a standard structure—numbers and letters on either side of an @ and a 2–4 character extension at the end—a regular expression can easily evaluate the validity of an email address.

After validating all user input, the script looks at the `$err` array to see if any error messages were recorded. If any error messages are detected within the array, a `for` loop runs to build a single error message to be returned for display on the product's page.

```
if(count($err) > 0){
    // Build and return error message
    for($i=0;$i<count($err);$i++){
        $message .= $err[$i] . '<br />';
    }
    return '<div class="error">'.$message.'</div>';
}else{

    // Send the email to friend

}
```

If no errors were detected, the script moves on to send the email. PHP's built-in `mail()` function makes sending email messages very simple. It accepts the following four parameters:

- to email address
- subject
- message
- optional additional email headers such as a reply-to address

To send the message the script will first assemble the message in a variable:

```
$message = $_POST['message']."
    Check it out: http://example.com/products/".$_POST['product-
id']."
    ".$_POST['senders-name'];
```

The URL for the product page is assembled within the message variable using the product id specified in the HTML form's hidden field. The URL in this example assumes that you are using the search engine friendly URL techniques outlined in Chapter 3, but you could just as easily modify this URL to pass the product id via a query string if necessary.

Once the message is built the mail() function is run from within a conditional. The mail() function returns true if the message is successfully sent and false if something went wrong. By running the mail() function inside a conditional the script can automatically return a success or failure message to let the user know the outcome of the process.

```
if(mail($_POST['senders-email'],"This is awesome!",
$message,"From:".$_POST['senders-email']."\r\nReply-to:".$_
POST['senders-email'])){
    return "Tada! Your message has been sent!";
}else{
    return "Duoh! Something went awry with your message.";
}
```

Let's take a look at the completed tellFriend() function:

```
<?php
function tellFriend(){

    // Validation ////////////////////////////////////////////
    $err = array(); // will store all errors found in form
submission

    // Check sender info
    if( empty($_POST['senders-name']) ){
        array_push($err,"Please include your name"); }

    if( empty($_POST['senders-email']) ){
        array_push($err,"Please include your email address");
    }elseif( !ereg("^[a-zA-Z0-9._-]+@[a-zA-Z0-9._-]+\.([a-zA-
Z]{2,4})$", $_POST['senders-email']) ){
        array_push($err,"Your email address is invalid");
    }

    // Check friend's info
    if( empty($_POST['friends-name']) ){
        array_push($err,"Please include your friend's name"); }

    if( empty($_POST['friends-email']) ){
        array_push($err,"Please include your friend's email
address");
    }elseif( !ereg("^[a-zA-Z0-9._-]+@[a-zA-Z0-9._-]+\.([a-zA-
Z]{2,4})$", $_POST['friends-email']) ){
        array_push($err,"Your friend's email address is invalid");
    }
```

```
    // Check for blank message
    if( empty($_POST['message']) ){
        array_push($err,"No message provided"); }

    if(count($err) > 0){
        // Build and return error message
        for($i=0;$i<count($err);$i++){
            $message .= $err[$i] . '<br />';
        }
        return '<div class="error">'.$message.'</div>';

    }else{

        // Send the email to friend
        $message = $_POST['message']."\r\rCheck it out: http://
example.com/products/".$_POST['product-id']."\r\r".$_POST['senders-
name'];

        if(mail($_POST['senders-email'],"This is awesome!",
$message,"From:".$_POST['senders-email']."\r\nReply-to:".$_
POST['senders-email'])){
            return "Tada! Your message has been sent!";
        }else{
            return "Duoh! Something went awry with your message.";
        }
    }
}
?>
```

That wraps up the `tell-friend.php` file. To put the finishing touches on the system we'll return to the `product.php` page.

Calling the tellFriend() Function

When the Tell a Friend form is submitted it will reload itself and run the `tellFriend()` function. In order to detect whether the form has been submitted, the `product.php` page will look for the presence of a value within one of the `$_POST` variables automatically created in PHP's memory when form data is sent to it via the `post` method. Any form element with a `name` and `value` attribute defined will generate a new `$_POST` variable. The variable we'll look for is the one created by the submit button - `$_POST['submit-friend']`.

Looking for the presence of this variable will allow the page to identify the specific form that has been submitted rather than just any `post` form submission.

When the product reviews form is added to the page the script will be smart enough to discern which of the two forms has been submitted, invoking the correct function.

Directly above the Tell a Friend form include the `tell-friend.php` file into the page. Next, add a `<div>` tag where the response from the script can be displayed—message successfully sent or errors occurred. Within this `<div>` add a conditional to call the `tellFriend()` function if the `$_POST['submit-friend']` is set.

```
<? require_once("inc/tell-friend.php"); ?>
<div class="response"><? if(isset($_POST['submit-friend'])){echo
tellFriend();}?></div>
```

Because the `tellFriend()` function uses the `return` keyword to send its response messages back to the location from which it's called, echo `tellFriend()` will automatically write success or error messages inside of this `<div>`.

The product page is now ready to let users tell their friends and family about the products they want. Let's move on to the user-generated review system.

Preventing Spam

Count on spam bots visiting your site to pollute your forms with their annoying messages. It's an unfortunate reality that requires preventative measures. You can fend off spam by requiring user input that is only possible by humans.

To solve the spam problem in the simplest fashion, add a spam challenge question field to your forms. Ask a question like "What color is the sky?" or "What's two plus two?" then use a simple conditional in your processing script to see if the answer provided is valid. A spam bot will have a tough time correctly filling out this field.

Another option is to use a Captcha image. Captchas are those crazy distorted images often seen accompanying signup forms. The user has to type the text contents of an image into field to prove they are not a spam bot, so the form can be processed.

Ed Eliot has created a brilliant PHP Captcha class (**http://www.ejeliot.com/pages/2**) that makes generating Captcha images in your forms very simple. His class also generates accessible audio alternatives for the visually impaired.

ReCaptcha (**http://recaptcha.net/**) provides an easy-to-use Captcha widget you can drop into any form. It culls text from scanned books, so with each completed Captcha a book is a few words closer to being digitized.

Both of the form examples in this chapter—Tell a Friend and Customer Reviews—could be enhanced with some sort of spam prevention. Choose an approach that's easiest for you to implement, and add it to these examples to keep your inbox a little less junky.

Building a Product Review Feature

The product review system is a little more complex than the Tell a Friend system as it requires some way of vetting the reviews submitted. It's highly likely that your site is going to get some spammy or undesirable review submissions. To solve the problem, an email containing the review will first be sent to the site administrator. Also included in the email will be a link to accept the review, and one to delete it. With a single click the admin can approve or delete the review submission.

To store all of the reviews we'll use a MySQL database table like the one shown in **TABLE 11.1**. The `reviewid` field will act as the primary key so we can uniquely identify each record. Set this field to auto-increment so each time a new record is added a unique number will automatically be stored in the `reviewid` field.

TABLE 11.1 My SQL Database Table

reviewed	productid	review	reviewersname	status	dateposted
int PK	int	text	text	int	int

On the `product.php` page we'll pull all of the approved reviews from the database and display them directly above the review submission form. Later in the chapter we'll add the PHP code to display the reviews, but first let's build the HTML form for submissions.

```
<div id="reviews-container">
    <h4>Product Reviews</h4>
    <!-- reviews will be displayed here -->
</div>

<div id="post-review">
    <form id="review-form" action="<?=$_SERVER['PHP_SELF']?>"
method="post">
        <fieldset>
        <div class="notification">All fields required</div>
        <legend>Review of This Product</legend>

        <label for="reviewersname">Your Name</label>
        <input type="text" name="reviewersname" id="reviewersname"
value="<?=$_POST['reviewersname']?>" />
```

```
        <label for="review">Review</label>
        <textarea name="review" id="review"><?=$_POST['review']?>
</textarea>
    </fieldset>

    <input type="hidden" name="product-id" value="12" />
    <input type="submit" name="submit-review" value="Save"
class="btn" />
    </form>
</div>
```

You may see a lot of similarities here with the previous form example. Like the Tell a Friend form, the review form also uses `<?=$_SERVER['PHP_SELF']?>` to submit the input to the same page where it will be processed by a linked script. The fields also maintain user input via the `$_POST` super-global variables automatically created when the form is submitted. If the user made a mistake, their review will still be present in the form even after they've attempted to submit it.

This form also includes a hidden product id field just like the Tell a Friend form. The product id will need to be associated with each review record in the database so we're able to grab only the reviews associated with a particular product.

We'll make some minor additions to this form just before completing the system to connect it to the reviews script that we'll create next.

Building the Reviews Script

The `reviews.php` script will contain four functions that will manage all tasks associated with the reviews. Again, because the core functionality will be kept externally you could easily plug this script into any project. Each function is named to describe the task it completes:

- `storeReview()`
- `getReviews()`
- `confirmReview()`
- `deleteReview()`

A task common to each of these four functions is some sort of interaction with the database. Rather than repeatedly making a connection inside each function, we'll establish a database connection before declaring any of the functions.

```
$con = mysql_connect('hostname','dbusername','dbpassword');
mysql_select_db('dbname',$con);
```

Be sure to plug in the correct host, username, password, and database name for your database server.

storeReview() The `storeReview()` function has four primary tasks to complete:

1. Validate all user input.
2. Store the review in the database.
3. Email the admin so they can delete or confirm the review.
4. Let the user know the outcome of the process.

The user input validation in this function will be very much like the validation created earlier for the Tell a Friend function. It uses an array to log any errors encountered.

This function will receive no parameters. It begins by declaring a variable to catch validation errors and a variable to define the email address to which the review should be sent.

```
function storeReview(){
    $adminemail = "you@example.com";
    $err = array();

    // Validation
    if(empty($_POST['reviewersname'])){
        array_push($err,"Please include your name"); }

    if(empty($_POST['review'])){
        array_push($err,"Don't forget to write a review!"); }

    if(count($err) > 0){

        // Build and return error message
    for($i=0;$i<count($err);$i++){
        $message .= $err[$i] . '<br />';
    }

        return '<div class="error">'.$message.'</div>';
    }

}
```

The validation is simply confirming that the user included a name and a review in their submission. If a problem was encountered, an error message is returned to `product.php` page, and the script will stop.

If validation went well the review gets stored in the database. To make sure the user input is safe for storage, we'll use some of PHP's built-in functions to clean things up. The `mysql_real_escape_string()` function adds slashes to quotes within strings to prevent SQL injection attacks—see the section "Creating a Custom Search Tool" in Chapter 6 for more info on this topic. The `strip_tags()` function removes any dangerous HTML and disables JavaScript that could affect the display of the review on the `product.php` page.

```
$reviewersname = mysql_real_escape_string(strip_tags($_
POST['reviewersname']));
$review = mysql_real_escape_string(strip_tags($_POST['review']));
$productid = $_POST['product-id'];
```

With the review ready for storage in the database, an SQL query can be built. Remember that even though we're storing the review in the database, it won't be visible on the site until the admin has received the review by email and clicked the link within to approve it. The SQL query will write `0` in the `status` field of the record. Any record with status `0` is still waiting for the administrator's approval and won't be pulled by the `getReviews()` function we'll create shortly.

```
$sql = "INSERT INTO reviews SET productid='$productid',
review='$review', reviewersname='$reviewersname', status='0',
dateposted='".time()."'";
```

The date the review was posted is recorded using PHP's `time()` function, which will get the number of seconds since January 1, 1970—known as the epoch. This assigns a unique number to each second so the dates can be easily sorted later in the `getReviews()` function. This large integer can be easily translated into a textual date users can read using PHP's `date()` function later when we need to display it.

The next step is to run the query, and watch for errors. The function will return an error message and stop the script if a problem was encountered, otherwise it will move on to sending the email.

```
if(!$result = mysql_query($sql)){
    return "Duoh! There was some trouble storing your review.";
}else{
    // Send email to admin

}
```

Inside the second branch of this conditional the message is assembled before it's sent to the admin. Earlier when the database table was set up, the `reviewid` field was set as the primary key to auto-increment. Once the query has been run and the review is saved in the table, it will automatically be assigned a unique `reviewid`. PHP has a handy, built-in function called `mysql_insert_id()` that will grab the id of the record from your last query. We'll need the `reviewid` to create the confirm and delete links in the email. These links will use a query string to pass the `reviewid` and the action to be performed on it.

Here's what the message assembly looks like:

```
// Grab reviewid for inclusion in email
$reviewid = mysql_insert_id();

// Send email to admin for review
$message = "
$reviewersname posted a new product review
------------------------------------------------------------
$review


Confirm: http://example.com/vet-review.php?action=confirm&reviewid
=$reviewid
Delete: http://example.com/vet-review.php?action=delete&reviewid=$
reviewid
";
```

The `vet-review.php` page alluded to in the confirm and delete links will be created shortly to help the administrator quickly confirm or delete all reviews submitted.

All that's left to complete in the `storeReview()` function is the email delivery. As was the case in the Tell a Friend system, the `mail()` function will run in a conditional to instantly return a message indicating an error or success.

```
if(mail($adminemail,"New product review to confirm",$message,"From
:$adminemail\r\nReply-to:$adminemail")){

    return "Thanks for posting your review! It will be read by the
site's head honcho before it will be added to the site.";

}else{
```

```
    return "Duoh! There was some trouble letting the site's head
honcho know about your review.";

}
```

The storeReview() function is all set, and we're now ready to build the getReviews() function.

getReviews() As you might have already guessed, the getReviews() function will grab all of the reviews for a particular product and return them as a string product.php page. It expects to receive a product id so it can grab the right set of reviews.

The function will only retrieve reviews that have the status of 1. This means they have been reviewed and approved. By default, all new reviews will have a status of 0, indicating they have not yet been approved. All of the reviews will be sorted with the newest ones on top of the stack.

After the function queries the database for the reviews, it will check to see if any errors occurred in the transaction. An error message will be returned if something went awry.

It's possible that even if the communication with the database went OK, the product may not have any reviews written about it yet. In this case a descriptive message will be returned to let the user know.

Once all of the error checking is out of the way, a while loop will build a string with all of the reviews assembled in tags with CSS classes assigned so they can be easily styled back on the product.php page.

Let's take a look at the assembled getReviews() function.

```
function getReviews($productid){

    $sql = "SELECT * FROM reviews WHERE productid='$productid'
            AND status='1' ORDER BY dateposted DESC";

    if(!$result = mysql_query($sql)){

        return '
            <span class="error">
            Duoh! There was a problem getting the product reviews.
            </span>';
```

```
    }elseif(mysql_num_rows($result) == 0){

        return 'No reviews are posted for this product';

    }else{

        // Grab all reviews
        while($row = mysql_fetch_array($result)){
            $reviews .= '
         <div class="review">'.stripslashes($row['review']).'
             <div class="review-meta">Posted by '.$row['reviewersn
ame'].' on '.date("l F j, Y",$row['dateposted']).'</div>
         </div>';

        }

        return $reviews;
    }
}
```

Because the `mysql_real_escape_string()` function was used to escape quotes in reviews when stored in the database, the `strip_slashes()` function is used here to remove them from the display.

If you'd like to display the date the review was posted in a different format, consult the PHP documentation for the `date()` function http://www.php.net/manual/en/function.date.php.

In the `storeReview()` function we used PHP's `time()` function to store the date and time the review was posted. The `date()` function is used here to transform the long integer it created into a textual date.

The `getReview()` function is now all set. The last two functions in the `reviews.php` script—`confirmReview()` and `deleteReview()`—will handle the review vetting process.

confirmReview() The `confirmReview()` function will be triggered when the admin clicks the confirm link in the review notification email. It will simply update a review record to change the status from `0` to `1`, which will cause it to be displayed on the `product.php` page. This function will need to be passed the `reviewid` so it can locate the review record to be updated.

The function starts by simply confirming that a `reviewid` has been passed to it when it was called. If not, it will return an error and exit the script. If no error is encountered, it will run an `UPDATE` query to change the review's status and display a success or failure message.

It's a fairly brief function so here it is in its entirety:

```
 function confirmReview($reviewid){
    if(!isset($reviewid)){
        return "Sorry, we couldn't confirm the review because no
review id was supplied";
    }

    $reviewid = mysql_real_escape_string($reviewid);
    $sql = "UPDATE reviews SET status='1' WHERE
reviewid='$reviewid'";

    if($result = mysql_query($sql)){
        return "The review has been posted!";
    }else{
        return "Duoh! There was some trouble when confirming the
review.";
    }
}
```

deleteReview() The `deleteReview()` function is a lot like the `confirmReview()` function, and is equally short. It too expects to receive a `reviewid` as a parameter, and will display an error if none is provided.

The rest of the function is exactly the same as the `confirmReview()` function except that instead of an `UPDATE` query, it uses `DELETE` to remove a review from the database.

```
function deleteReview($reviewid){
    if(!isset($reviewid)){
        return "Sorry, we couldn't delete the review because no
review id was supplied";
    }

    $reviewid = mysql_real_escape_string($reviewid);
    $sql = "DELETE FROM reviews WHERE reviewid='$reviewid'";

    if($result = mysql_query($sql)){
        return "The review has been deleted!";
    }else{
       return "Duoh! There was some trouble when deleting the
review.";
    }
}
```

The reviews.php script is now finally all wrapped up. Here's what it looks like when fully assembled:

```
<?
// Connect to database
$con = mysql_connect('hostname','dbusername','dbpassword');
mysql_select_db('dbname',$con);

function getReviews($productid){
    $sql = "SELECT * FROM reviews WHERE productid='$productid' AND
status='1' ORDER BY dateposted DESC";
    if(!$result = mysql_query($sql)){
        return '<span class="error">Duoh! There was a problem
retrieving the reviews for this product.</span>';
    }elseif(mysql_num_rows($result) == 0){
        return 'No reviews are posted for this product';
    }else{
        while($row = mysql_fetch_array($result)){
            $reviews .= '
            <div class="review">'.stripslashes($row['review']).'
            <div class="review-meta">Posted by '.$row['reviewersna
me'].' on '.date("l F j, Y",$row['dateposted']).'</div>
            </div>';
        }
        return $reviews;
    }
}

function getReviewsOld($productid){
    $sql = "SELECT * FROM reviews WHERE productid='$productid' AND
status='1' ORDER BY dateposted DESC";
    if($result = mysql_query($sql)){

        if(mysql_num_rows($result) > 0){
            $reviews = array();
            while($row = mysql_fetch_array($result)){
                $review = array('reviewersname'=>stripslashes($row
['reviewersname']),'review'=>stripslashes($row['review']),'datepos
ted'=>$row['dateposted']);
                array_push($reviews,$review);
            }
            return $reviews;
        }else{
            return "No reviews are posted for this product";
        }
```

```
    }else{
        return "Duoh! There was a problem retrieving the reviews
for this product.";
    }
}

function storeReview(){
    $adminemail = "you@example.com";
    $err = array();

    // Validation
    if(empty($_POST['reviewersname'])){
        array_push($err,"Please include your name"); }
    if(empty($_POST['review'])){
        array_push($err,"Don't forget to write a review!"); }
    if(count($err) > 0){
        // Build and return error message
        for($i=0;$i<count($err);$i++){
            $message .= $err[$i] . '<br />';
        }
        return '<div class="error">'.$message.'</div>';
    }

    $reviewersname = mysql_real_escape_string(strip_tags($_
POST['reviewersname']));
    $review = mysql_real_escape_string(strip_tags($_
POST['review']));

    $sql = "INSERT INTO reviews SET productid='$productid',
review='$review', reviewersname='$reviewersname', status='0',
dateposted='".time()."'";
    if(!$result = mysql_query($sql)){
        return "Duoh! There was some trouble storing your
review.";
    }else{
        // Grab reviewid for inclusion in email
        $reviewid = mysql_insert_id();

        // Send email to admin for review
        $message = "
        $reviewersname posted a new product review
        ------------------------------------------------------------
        $review
        ------------------------------------------------------------
```

```
        Confirm: http://example.com/vet-review.php?action=confirm&
reviewid=$reviewid
        Delete: http://example.com/vet-review.php?action=delete&re
viewid=$reviewid
        ";

        if(mail($adminemail,"New product review to confirm",$messa
ge,"From:$adminemail\r\nReply-to:$adminemail")){
            return "Thanks for posting your review! It will be
read by the site's head honcho before it will be added to the
site.";
        }else{
            return "Duoh! There was some trouble letting the
site's head honcho know about your review.";
        }
    }
}

function confirmReview($reviewid){
    if(!isset($reviewid)){
        return "Sorry, we couldn't confirm the review because no
review id was supplied";}

    $reviewid = mysql_real_escape_string($reviewid);
    $sql = "UPDATE reviews SET status='1' WHERE
reviewid='$reviewid'";
    if($result = mysql_query($sql)){
        return "The review has been posted!";
    }else{
        return "Duoh! There was some trouble when confirming the
review.";
    }
}

function deleteReview($reviewid){
    if(!isset($reviewid)){ return "Sorry, we couldn't delete the
review because no review id was supplied"; }

    $reviewid = mysql_real_escape_string($reviewid);
    $sql = "DELETE FROM reviews WHERE reviewid='$reviewid'";
    if($result = mysql_query($sql)){
        return "The review has been deleted!";
    }else{
```

```
        return "Duoh! There was some trouble when deleting the
review.";
    }
}
?>
```

To finish the review system we'll need to create a page to trigger the `confirmReview()` and `deleteReview()` functions when the admin clicks the links in a review notification email. Then we'll put the finishing touches on the `product.php` page to call the `getReviews()` and `storeReview()` functions.

Handling the Review Vetting Process

The email sent to the admin in the `storeReview()` function contained two links to a page called `vet-reviews.php` where the admin can instantly confirm or delete a review. The page toggles between the `confirmReview()` and `deleteReview()` functions by looking at a variable called `action` within the query string. Here's what those links might look like in the email:

```
http://example.com/vet-review.php?action=confirm&reviewid=12
http://example.com/vet-review.php?action=delete&reviewid=12
```

In addition to defining the action to take place, the URL also includes the `reviewid` of the target record.

Start the `vet-reviews.php` page with some basic HTML to define the structure. This page will first include the `reviews.php` script we've just created, then check to see if it has been supplied an `action` and `reviewid` `$_GET` variable via the URL. If it has what it needs, the script will run the `deleteReview()` or `confirmReview()` function depending on the action defined in the URL.

```
...
<div id="vet-message">
<?php
require_once("inc/reviews.php");
if(!isset($_GET['action']) or !isset($_GET['reviewid'])){
    echo "Sorry, this page can only be accessed by the
administrator";
}else{

    switch($_GET['action']){
        case "confirm":
           echo confirmReview($_GET['reviewid']); break;

        case "delete":
            echo deleteReview($_GET['reviewid']); break;
```

```
        default:
            echo "Whoa, pardner! There's no way to vet this review
if you don't provide a valid action."; break;
    }

}
?>
</div>
...
```

The switch statement looks for the value confirm or delete in the `$_GET['action']` variable. If neither value is in the `$_GET['action']` variable, the switch statement goes to the default branch, which will display a message indicating that a valid action has not been supplied.

FIGURE 11.10 shows what the admin will see when they click the link in their email to approve a new review.

That's all there is to the `vet-reviews.php` script. Remember, only the admin will know the page exists. Users aren't likely to stumble across it because it's only linked from the email sent to the admin. Even if they do discover the script they'd have to supply the correct query string to make it swing into action.

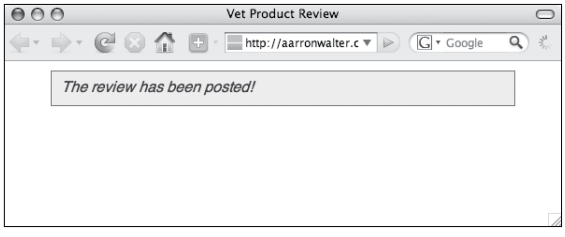

FIGURE 11.10 *When the admin clicks the confirm link in an email notification, they'll launch a browser window where the `vet-reviews.php` script can run and the review will be added to the site.*

Securing the vet-reviews.php Script

It's a good idea to include some additional security for the script such as a password to keep things private. Using a `.htaccess` file, Apache will require a username and password to access a file or directory. I'll leave this as an exercise for the reader to implement, but here are some resources that can get you started.

http://tools.dynamicdrive.com/password/

http://www.elated.com/articles/password-protecting-your-pages-with-htaccess/

http://www.mattcutts.com/blog/htaccess-101/

Calling the getReviews() and storeReview() Functions

With the various pieces of the reviews system completed, we'll connect the `product.php` page to the `review.php` script so the current reviews can be displayed, and new ones can be stored in the database.

As we began the reviews system, directly above the product review form a `<div>` was created to display all of the current reviews. Within that `<div>` we'll first include the `reviews.php` script.

```
<div id="reviews-container">
    <h4>Product Reviews</h4>
    <? require_once("inc/reviews.php"); ?>
</div>
```

Directly after the include we'll add a new `<div>` to contain the call to the `storeReview()` function. Just like the Tell a Friend system created earlier, the review system will look to see if a `$_POST` variable has been created in PHP's memory by the submit button. If it has, this indicates the form has been submitted so we'll need to call the `storeReview()` function and display the results it returns in the page.

```
<div class="response">
    <? if(isset($_POST['submit-review'])){echo storeReview();}?>
</div>
```

Lastly, just before the close of the `<div id="reviews-container">` add another `<div>` in which to display the reviews already stored in the database. Inside, the `getReviews()` function is called, passing the id of the product so its reviews can be retrieved.

```
<div class="reviews"><?=getReviews(12)?></div>
```

The user-generated product review system is finally complete and ready to go. For this example, I've hard-coded the product id throughout the `product.php` page. Using the search engine friendly URL techniques discussed in Chapter 3, it wouldn't be difficult to convert this page to pull the product id from the URL in order to make it flexible enough to accommodate an infinite number of products. Similarly, you could pull the product content from a database using the same product id in the URL to quickly transform the `product.php` page into the foundation of a practical ecommerce system.

Building Social Network Submission Links

The final viral marketing tool on the `product.php` page is a series of quick links that let users add this product information to their favorite social networking applications. The easier you make it for your users to share information about your product, the greater the exposure it will get.

Social networking sites not only let users submit content directly via their website, but also provide a submission service that lets other websites and applications submit content via a specially constructed URL that includes certain variables. Inside the URL you can pass a variety of information including the title of your page, your URL, a description of your content, and more.

The tricky part of submitting your content to social networking sites using a URL is that you have to know which variables are required, and there's no standard that's followed. Each site has its own approach.

I've compiled a list of submission URLs for some of the most popular social networking sites. The highlighted text describes the type of content you'll need to include in each variable.

Digg

```
http://digg.com/submit?phase=2&url=URL of your page&topic=Category
to submit to &bodytext=Description of content&title=Page title
```

Delicious

```
http://del.icio.us/post?noui&jump=close&url=URL of your
page&title=Page title
```

Facebook

```
http://www.facebook.com/sharer.php?u=URL of your page&t=Page title
```

Technorati

```
http://technorati.com/faves?add=URL of your page
```

Twitter

```
http://twitter.com/home?status=URL of your page plus any
additional descriptive text
```

Furl

```
http://furl.net/storeIt.jsp?u=URL of your page&t=Page title
```

Reddit

```
http://reddit.com/submit?url=URL of your page&title=Page title
```

StumbleUpon

```
http://www.stumbleupon.com/submit?url=URL of your page&title=Page title
```

To illustrate how these submission URLs are added to the `product.php` page, here's a simple example of a Facebook submission link. I've used some CSS to display a Facebook icon to the left of the link.

HTML

```
<a href="http://www.facebook.com/sharer.php?u=http://example.com/product.php&t=The Candy Dish" class="facebook">Facebook</a>
```

CSS

```
.facebook {padding-left:20px; background:url(../i/facebook.gif) left no-repeat;}
```

FIGURE 11.10 *AddThis (**http://addthis.com**) makes it easy to drop links in your page to social networking sites.*

If you find making links to all of these social networking sites a bit tedious you could take a short cut. AddThis (**http://addthis.com**) is a free service that provides a widget to add a wide array of social networking buttons to your page (see **FIGURE 11.10**). It also includes free statistics that will tell you which pages on your site are bookmarked most often.

That's a wrap on all three of these practical viral marketing tools. Users will now be able to share this product with friends and family through the Tell a Friend form, they can share their perspective on it via the user-generated reviews form, or share it on their favorite social networking platform.