# PHP HACKS™

*Tips & Tools for Creating Dynamic Web Sites*

O'REILLY®

*Jack D. Herrington*

**PHP Hacks™**
by Jack D. Herrington

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

| | | | |
|---|---|---|---|
| **Editor:** | Brett McLaughlin | **Cover Designer:** | Marcia Friedman |
| **Production Editor:** | Reba Libby | **Interior Designer:** | David Futato |

**Printing History:**

| | |
|---|---|
| December 2005: | First Edition. |

RepKover™ This book uses RepKover™, a durable and flexible lay-flat binding.

ISBN: 0-596-10139-2
[M]

CHAPTER SIX

# Application Design

Hacks 51–66

Sitting on top of the database and below the HTML is application logic. This chapter concentrates on hacks that will add stability and flexibility to your application logic. Topics covered include security and roles, password management, login and session management, and e-commerce.

## HACK #51 Create Modular Interfaces

Use dynamic loading to allow users to write snap-in modules for your application.

Most of the really popular PHP open source applications have an extension mechanism that allows for PHP coders to write small fragments of code that are dynamically loaded into the application. This hack demonstrates an XML-based drawing script that you can extend simply by placing new PHP classes into a modules directory; of course, the point is not as much the drawing code as the way you can extend it easily.

### The Code

Save the code in Example 6-1 as *modhost.php*.

*Example 6-1. The code that handles a modular PHP architecture*

```
<?php
class DrawingEnvironment
{
  private $img = null;
  private $x = null;
  private $y = null;
  private $colors = array( );

  public function __construct( $x, $y )
  {
    $this->img = imagecreatetruecolor( $x, $y );
```

*Example 6-1. The code that handles a modular PHP architecture (continued)*

```
    $this->addColor( 'white', 255, 255, 255 );
    $this->addColor( 'black', 0, 0, 0 );
    $this->addColor( 'red', 255, 0, 0 );
    $this->addColor( 'green', 0, 255, 0 );
    $this->addColor( 'blue', 0, 0, 255 );

    imagefilledrectangle( $this->image( ),
      0, 0, $x, $y, $this->color( 'white' ) );
  }

  public function image( ) { return $this->img; }
  public function size_x( ) { return $this->x; }
  public function size_y( ) { return $this->y; }
  public function color( $c ) { return $this->colors[$c]; }

  public function save( $file )
  {
    imagepng( $this->img, $file );
  }

  protected function addColor( $name, $r, $g, $b )
  {
    $col = imagecolorallocate($this->img, $r, $g, $b);
    $this->colors[ $name ] = $col;
  }
}

interface DrawingObject
{
  function drawObject( $env );
  function setParam( $name, $value );
}

function loadModules( $dir )
{
  $classes = array( );

  $dh = new DirectoryIterator( $dir );
  foreach( $dh as $file )
  {
    if( $file->isDir( ) == 0 && preg_match( "/[.]php$/", $file ) )
    {
      include_once( $dir."/".$file );
      $class = preg_replace( "/[.]php$/", "", $file );
      $classes []= $class;
    }
  }

  return $classes;
}
```

*Example 6-1. The code that handles a modular PHP architecture (continued)*

```
$classes = loadModules( "mods" );

$dom = new DOMDocument();
$dom->load( $argv[1] );
$nl = $dom->getElementsByTagName( "image" );
$root = $nl->item( 0 );

$size_x = $root->getAttribute( 'x' );
$size_y = $root->getAttribute( 'y' );
$file = $root->getAttribute( 'file' );

$de = new DrawingEnvironment( $size_x, $size_y );

$obs_spec = array();

$el = $root->firstChild;
while( $el != null )
{
  if ( $el->tagName != null )
  {
    $params = array();
    for( $i = 0; $i < $el->attributes->length; $i++ )
    {
      $p = $el->attributes->item( $i )->nodeName;
      $v = $el->attributes->item( $i )->nodeValue;
      $params[ $p ] = $v;
    }

    $obs_spec []= array(
      'type' => $el->tagName,
      'params' => $params
    );
  }
  $el = $el->nextSibling;
}

foreach( $obs_spec as $os )
{
  $ob = null;
  eval( '$ob = new '.$os['type'].'();' );
  foreach( $os[ 'params' ] as $key => $value )
    $ob->setParam( $key, $value );
  $ob->drawObject( $de );
}

$de->save( $file );
?>
```

Save the code in Example 6-2 as *mods/Circle.php*.

*Example 6-2. An example module that draws circles*

```php
<?php
class Circle implements DrawingObject
{
  private $radius = null;
  private $color = null;
  private $x = null;
  private $y = null;

  function drawObject( $env )
  {
    $r2 = $this->radius / 2;
    imagefilledellipse( $env->image( ),
      $this->x - $r2, $this->y - $r2,
      $this->radius, $this->radius,
      $env->color( $this->color )
    );
  }

  function setParam( $name, $value )
  {
    if ( $name == "radius" ) $this->radius = $value;
    if ( $name == "color" ) $this->color = $value;
    if ( $name == "x" ) $this->x = $value;
    if ( $name == "y" ) $this->y = $value;
  }
}
?>
```

## Running the Hack

This hack is run on the command line. The first thing to do is to create an XML test file:

```xml
<image x='100' y='100' file='out.png'>
        <Circle x='20' y='40' color='red' radius='15' />
        <Circle x='60' y='30' color='green' radius='30' />
        <Circle x='70' y='75' color='blue' radius='35' />
</image>
```

This XML file specifies that the image should be 100×100 pixels and named *out.png*, and that the image should have three circles, each of varying size and color.

With the XML in hand, run the script:

```
% php modhost.php test.xml
```

The first argument to the script is the name of the XML file that contains the image specifications. The output image file looks like Figure 6-1.

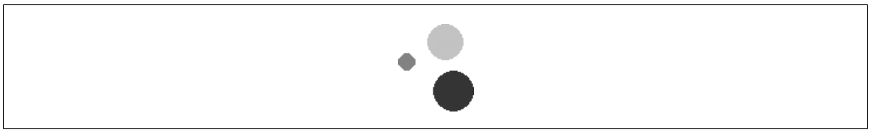


Figure 6-1. The output image

To explain a little about what happened here, let me start with the *modhost.php* file. At the start of the file, I've defined the `DrawingEnvironment` class, which is just a wrapper around an image with a few accessors. This environment will be passed to any drawing objects so that those objects can paint into the image. The next point of interest is the `DrawingObject` interface, which objects must conform to for drawing.

The real trick comes in the `loadModules( )` function, which loads all of the modules from the specified directory into the PHP environment. Then the script reads the XML file supplied to it, and parses it into the `$obs_spec` object, which is an array version of the XML file. The next step is to create the drawing environment and build the drawing objects based on the `$obs_spec` values; these values are then rendered into the image. Finally, the image is stored to a file.

Figure 6-2 shows the relationships between the `DrawingEnvironment` and the `DrawingObjects`, as well as how the dynamically loaded `Circle` class implements the `DrawingObject` interface.

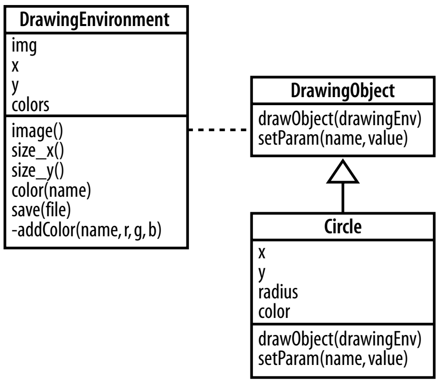


Figure 6-2. The structure of the drawing system

This is a simple illustration of this technique.

The specification of an interface makes this a PHP 5-specific script, but the `include_once( )` and `eval( )` functions were in PHP 4; it would take a bit of modification, but there is no reason that you can't do something similar to this in PHP 4.

I strongly recommend adding an extension mechanism such as this to any reasonably sized PHP application, especially when you expect deployment in multiple environments that you don't control. This approach gives end users the ability to customize the program to their requirements, without you having to go in and alter code directly for every new feature or object type.

## See Also

- "Create Objects with Abstract Factories" **[Hack #68]**
- "Observe Your Objects" **[Hack #67]**
- "Abstract Construction Code with a Builder" **[Hack #70]**

## HACK #52 Support Wiki Text

Make it easier for your customers to enter styled text into your application by supporting the Wiki syntax.

A new form of content management system for the Web, *Wikis* are a collection of pages, each titled with a *WikiWord*, which is a set of two or more capitalized words joined together without spaces. The ease with which you can install and update Wikis has made them extremely popular both on intranets and on the Internet. Perhaps the most famous Wiki is Wikipedia (*http://www.wikipedia.org/*). This is an encyclopedia on the Web that anyone can contribute content to by using just their web browser.

Another reason wikis are so popular is that formatting a Wiki page is a lot easier than writing the equivalent HTML code. For example, you specify a paragraph break by just typing two returns—there is no need to add `p` tags. In fact, most of the time tags aren't used at all. For example, you create a bulleted list by putting an asterisk at the start of each line; this is far easier than using the equivalent `ul` and `li` tags. This hack demonstrates using the wiki-formatting PEAR module in a PHP application.

### The Code

Save the code in Example 6-3 as *index.php*.

*Example 6-3. The page that allows you to edit Wiki text*

```
<html>
<body>
<form method="post" action="render.php">
<textarea name="text" cols="80" rows="20">
+ Header Level 1

Here's a paragraph and a link to AnotherPage.

* list item 1
* list item 2

Link to a NewPage like this.
</textarea><br/>
<input type="submit" />
</form>
</body>
</html>
```

Then save the code in Example 6-4 as *render.php*.

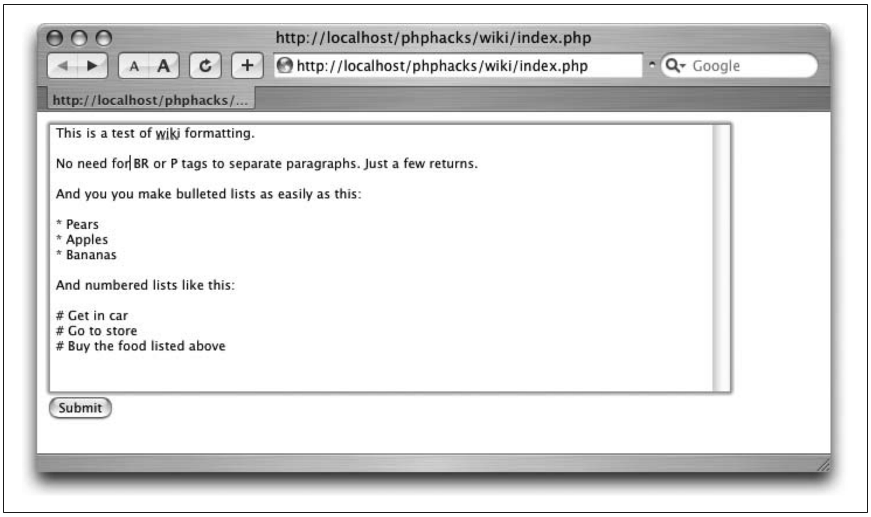
*Example 6-4. The PHP that renders the Wiki text*

```
<html>
<body>
<?php
// Include the Wiki Text Pear library
require_once( "Text/Wiki.php" );

// Create the Wiki object
$wiki = new Text_Wiki();

// Render the text field sent to us in the form
echo( $wiki->transform( $_POST["text" ], 'Xhtml' ) );
?>
</body>
</html>
```

## Running the Hack

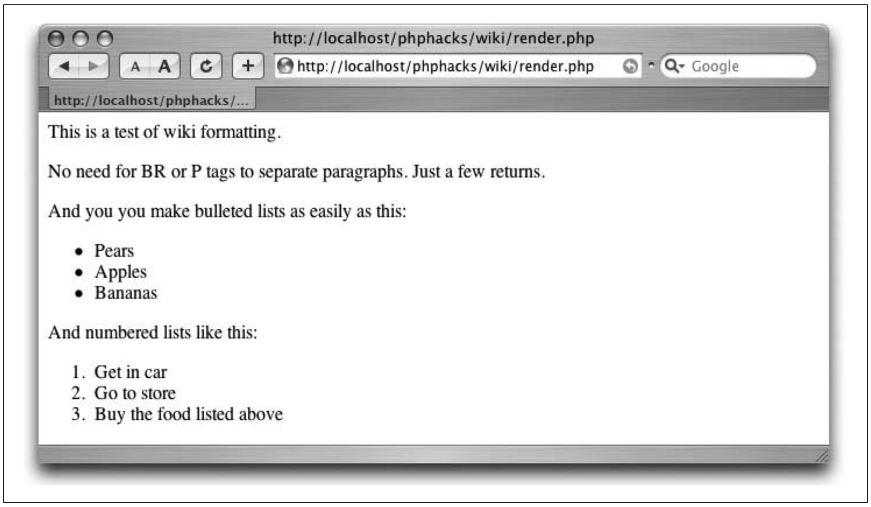
This code requires the Text_Wiki PEAR module **[Hack #2]**. After installing the module and creating the PHP files, navigate the browser to the *index.php* page shown in Figure 6-3.

Type some Wiki text into the form and click the Submit button. With the text shown in the example, the output looks like Figure 6-4.


http://localhost/phphacks/wiki/index.php
Google
http://localhost/phphacks/...
Submit

```
This is a test of wiki formatting.

No need for BR or P tags to separate paragraphs. Just a few returns.

And you you make bulleted lists as easily as this:

* Pears
* Apples
* Bananas

And numbered lists like this:

# Get in car
# Go to store
# Buy the food listed above
```

*Figure 6-3. The text input page for the Wiki text renderer*


http://localhost/phphacks/wiki/render.php
Google
http://localhost/phphacks/...

This is a test of wiki formatting.

No need for BR or P tags to separate paragraphs. Just a few returns.

And you you make bulleted lists as easily as this:

- Pears
- Apples
- Bananas

And numbered lists like this:

1. Get in car
2. Go to store
3. Buy the food listed above

*Figure 6-4. The rendered Wiki text*

A complete list of the wiki formatting rules is on the Text_ Wiki PEAR component home page at *http://wiki.ciaweb.net/yawiki/index.php?area=Text_Wiki&page=WikiRules*.

## Hacking the Hack

The `Text_Wiki` module is very well architected and written. New text formatting rules can be added, and the default formatting rules can be enabled and disabled to suit the application. The enableRule( ) and disableRule( ) methods enable and disable the built-in text formatting rules. The addRule( ) method adds new rules to the formatting engine.

## HACK #53 Turn Any Object into an Array

Use the Iterator interface in PHP 5 to turn any object into an array.

If you have ever used the DOM interface to read or write XML in PHP, you're already familiar with the DOMNodeList interface. Many methods in the DOM return an array of nodes. That array is implemented by the DOMNodeList object. To read the node list, you have to write code like this:

```
$dl = $doc->getElementsByTagName( "foo" );
for( $i = 0; $i < $dl->length; $i++ )
{
    $n = $dl->item( $i );
    ...
}
```

That's kind of unfortunate, isn't it, since PHP has that beautiful foreach operator that gives access to arrays with almost no potential for messing things up. Wouldn't it be great if the interface to DOM looked more like this?

```
foreach($doc->getElementsByTagName( "foo" ) as $n ) {
   ...
}
```

That is a lot cleaner and far less error prone.

Thanks to the additions in PHP 5, we can now allow foreach to work on any object, simply by having that class implement the Iterator interface. In this hack, I'll show how to implement an Observer pattern **[Hack #67]** using the Iterator interface.

### The Code

Save the code in Example 6-5 as *iterator.php*.

*Example 6-5. A class that uses PHP 5's new Iterator interface*

```
<?php
interface Listener
{
  public function invoke( $caller, $data );
}

class ListenerList implements Iterator
{
  private $listeners = array();
```

*Example 6-5. A class that uses PHP 5's new Iterator interface (continued)*

```
  public function __construct( )
  {
  }

  public function add( $listener )
  {
    $this->listeners []= $listener;
  }

  public function invoke( $caller, $data )
  {
    foreach( $this as $listener )
    {
      $listener->invoke( $caller, $data );
    }
  }

  public function rewind( )
  {
    reset($this->listeners);
  }

  public function current( )
  {
    return current($this->listeners);
  }

  public function key( )
  {
    return key($this->listeners);
  }

  public function next( )
  {
    return next($this->listeners);
  }

  public function valid( )
  {
    return ( $this->current( ) !== false );
  }
}

class SimpleListener implements Listener
{
  private $v;
  public function __construct( $v ) { $this->v = $v; }
  public function invoke( $caller, $data )
  {
    echo( $this->v." invoked with with '$data'\n" );
  }
```

*Example 6-5. A class that uses PHP 5's new Iterator interface (continued)*

```
  public function __tostring() { return "Listener ".$this->v; }
}

$ll = new ListenerList();

$ll->add( new SimpleListener( "a" ) );
$ll->add( new SimpleListener( "b" ) );
$ll->add( new SimpleListener( "c" ) );

print("Listeners:\n\n");
foreach( $ll as $listener )
{
  print( $listener );
  print( "\n" );
}

print("\nInvoking Listeners:\n\n");
$ll->invoke( null, "Some data" );
?>
```

The first section of the code defines a `Listener` interface for objects that are to be registered with `ListenerList`. The second part defines `ListenerList`, which is just a wrapper around an array with the addition of the `add()` and `invoke()` methods. The other methods all implement the `Iterator` interface. `SimpleListener` is just an implementation of the listener that prints when called.

Figure 6-5 shows the model for the code in this hack. `ListenerList` contains zero or more objects that implement the `Listener` interface. `SimpleListener` implements the `Listener` interface and just prints out a message whenever it's invoked.

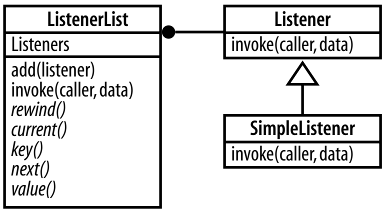


*Figure 6-5. The UML for ListenerList*

## Running the Hack

You run this hack on the command line using the command-line interpreter:

```
% php iterator.php
Listeners:
```

```
Listener a
Listener b
Listener c

Invoking Listeners:

a invoked with with 'Some data'
b invoked with with 'Some data'
c invoked with with 'Some data'
%
```

If you look at the end of the code from Example 6-5, you will see the tests that output here. The first test iterates through the list with a foreach statement. You see the result of this at the top of the run. The second section shows the result of the invoke( ) method being called on the ListenerList object.

The great thing about the Iterator interface is that you can now pass around complex interfaces in any case where you could only previously use arrays. Those array interfaces will still work, but now you can have additional methods as well.

### See Also

- "Observe Your Objects" **[Hack #67]**

## HACK #54 Create XML the Right Way

Use the XML DOM to create XML without errors.

Creating XML from your PHP web application is easy to get wrong. You can screw up the encoding so that special characters are not formatted properly, and you can miss start or end tags. Both of these problems, which are common in even simple PHP applications, will result in invalid XML and will keep the XML from being read properly by other XML consumers. Almost all of the problems result from working with XML as streams of characters instead of using an XML API such as DOM.

This hack will show you how to create XML DOMs in memory and then export them as text. This method of creating XML avoids all of these encoding and formatting issues, so your XML will be well-formed every time.

Figure 6-6 shows the in-memory XML tree that we will create in this hack. Each element is an object. The base of the system is DOMDocument, which points to the root node of the tree. From there, each DOMElement node can contain one or more child nodes and attribute nodes.

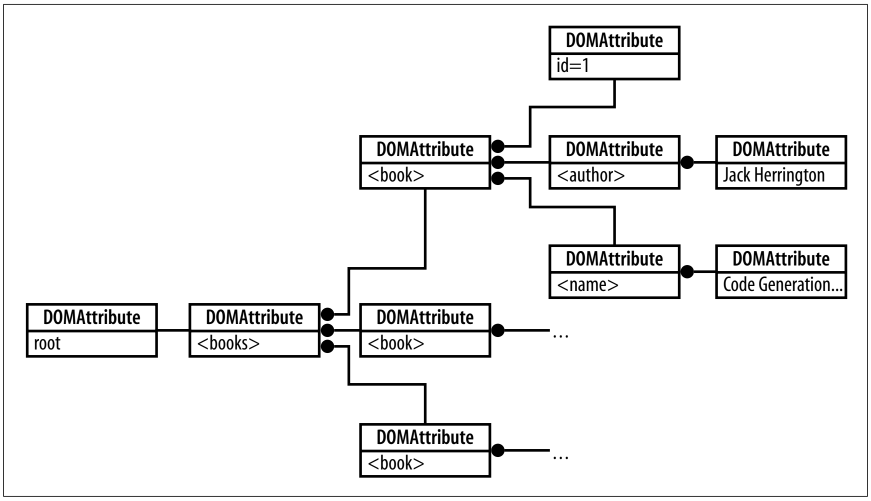


*Figure 6-6. The in-memory XML tree*

## The Code

Save the code in Example 6-6 as *xmldom.php*.

*Example 6-6. Sample code that builds XML the right way*

```
<?php
$books = array(
  array (
      id => 1,
      author => "Jack Herrington",
      name => "Code Generation in Action"
    ),
  array (
      id => 2,
      author => "Jack Herrington",
      name => "Podcasting Hacks"
    ),
  array (
      id => 3,
      author => "Jack Herrington",
      name => "PHP Hacks"
```

*Example 6-6. Sample code that builds XML the right way (continued)*

```
    )
  );

$dom = new DomDocument( );
$dom->formatOutput = true;

$root = $dom->createElement( "books" );
$dom->appendChild( $root );

foreach( $books as $book )
{
  $bn = $dom->createElement( "book" );
  $bn->setAttribute( 'id', $book['id'] );

  $author = $dom->createElement( "author" );
  $author->appendChild( $dom->createTextNode( $book['author'] ) );
  $bn->appendChild( $author );

  $name = $dom->createElement( "name" );
  $name->appendChild( $dom->createTextNode( $book['name'] ) );
  $bn->appendChild( $name );

  $root->appendChild( $bn );
}

header( "Content-type: text/xml" );
echo $dom->saveXML( );
?>
```

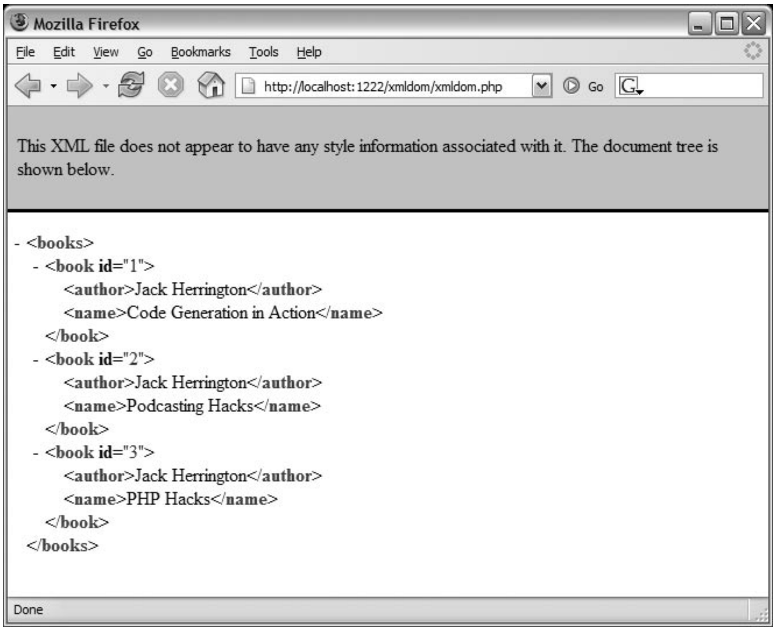
## Running the Hack

Upload this file to your server and surf to the *xmldom.php* page. You should see something like Figure 6-7.

This is nicely formatted and well-formed XML, and I didn't have to manually output a single tag name or attribute value. Instead, the DOM handles object creation and ties the objects together via the `appendChild( )` method. Finally, `saveXML( )` is used to export the XML as text. This is the easy and object-oriented way to create XML that is valid every time.

## See Also

- "Design Better SQL Schemas" **[Hack #34]**
- "Create Bulletproof Database Access" **[Hack #35]**


Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost:1222/xmldom/xmldom.php
Go
Done

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
- <books>
  - <book id="1">
      <author>Jack Herrington</author>
      <name>Code Generation in Action</name>
    </book>
  - <book id="2">
      <author>Jack Herrington</author>
      <name>Podcasting Hacks</name>
    </book>
  - <book id="3">
      <author>Jack Herrington</author>
      <name>PHP Hacks</name>
    </book>
  </books>
```

*Figure 6-7. The book XML shown in the Firefox browser*

## HACK #55 Fix the Double Submit Problem

Use a transaction table in your database to fix the classic double submit problem.

I have a couple of pet peeves when it comes to bad web application design. One of the biggest is the wealth of bad code written to fix "double submits." How often have you seen an e-commerce site that implores you, "Do not hit the submit button twice"?

This class problem results when a browser posts the contents of a web form to the server twice. However, if the user hits "submit" twice, this is exactly what the browser *should* do; it's the server that needs to determine whether this is an error.

Figure 6-8 shows the double submit problem graphically. The browser sends two requests because the user clicks twice. The first submit is accepted, and before the HTML is returned, the second submit goes out. Then the first response comes in, followed by the second response.

Figure 6-9 illustrates a fix to the double submit problem; the first request stores a unique ID in the page being processed. That way, when the second request comes in with the same ID, the redundant transaction is denied.

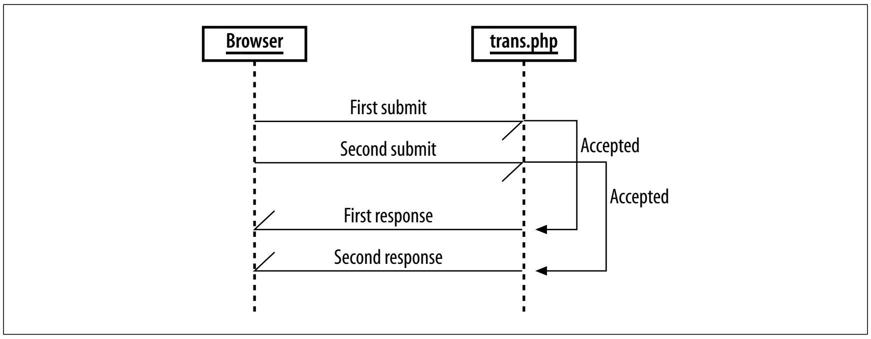


*Figure 6-8. The double submit problem sequence diagram*

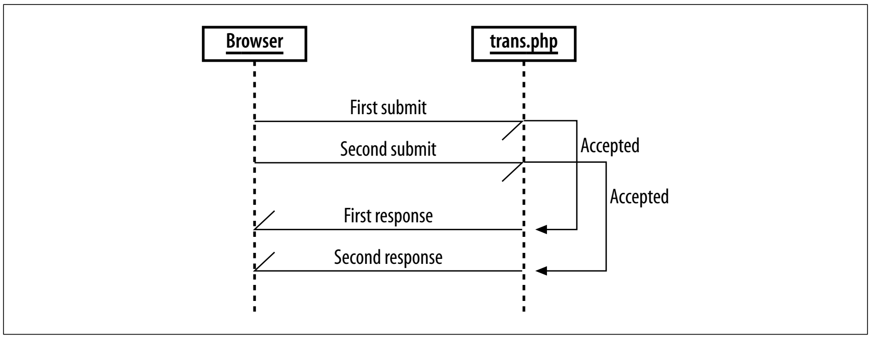


*Figure 6-9. The double submit solution requires denying the second request*

## The Code

Save the code in Example 6-7 as *db.sql*.

*Example 6-7. The database code for the transaction checker*

```
DROP TABLE IF EXISTS transcheck;
CREATE TABLE transcheck (
        transid TEXT,
        posted TIMESTAMP
        );
```

Save the code in Example 6-8 as *index.php*.

*Example 6-8. The HTML form that has the transaction ID*

```
<? require_once( "trans.php" ); ?>
<html>
<body>
<form action="handler.php" method="post">
<input type="hidden" name="transid" value="<?php echo( get_transid() ); ?>" />
```

*Example 6-8. The HTML form that has the transaction ID (continued)*

```
Name: <input type="text" /><br/>
Amount: <input type="text" size="5" /><br/>
<input type="submit" />
</format>
```

Save the code in Example 6-9 as *handler.php*.

*Example 6-9. The code that receives the form data and checks the transaction*

```
<? require_once( "trans.php" ); ?>
<html>
<body>
<?php if ( check_transid( $_POST["transid"] ) ) { ?>
This form has already been submitted.
<?php } else {
add_transid( $_POST["transid"] );
?>
Ok, you bought our marvelous product. Thanks!
<?php } ?>
</body>
</html>
```

Save the code in Example 6-10 as *trans.php*.

*Example 6-10. The transaction checking library*

```
<?php
require_once( "DB.php" );
$dsn = 'mysql://root:password@localhost/transtest';
$db =& DB::Connect( $dsn, array() );
if (PEAR::isError($db)) {
   die($db->getMessage());
}

function check_transid( $id )
{
  global $db;
  $res = $db->query( "SELECT COUNT(transid) FROM transcheck WHERE transid=?",
    array($id) );
  $res->fetchInto($row);
  return $row[0];
}

function add_transid( $id )
{
  global $db;
  $sth = $db->prepare( "INSERT INTO transcheck VALUES( ?, now() )" );
  $db->execute( $sth, array( $id ) );
}
```

*Example 6-10. The transaction checking library (continued)*

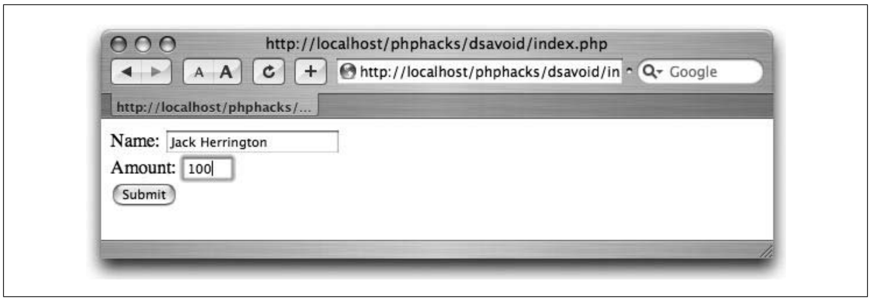
```
function get_transid( )
{
  $id = mt_rand( );
  while( check_transid( $id ) ) { $id = mt_rand(); }
  return $id;
}
?>
```

## Running the Hack

Upload the files to the server, and then use the `mysql` command to load the *db.sql* schema into your database:

```
    mysql --user=myuser --password=mypassword mydb < db.sql
```

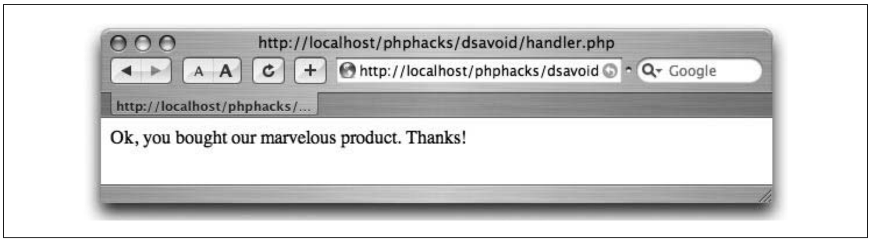
Next, navigate to the *index.php* page with your browser, and you will see the simple e-commerce form shown in Figure 6-10.


http://localhost/phphacks/dsavoid/index.php
http://localhost/phphacks/dsavoid/in
Google
http://localhost/phphacks/...

Name: Jack Herrington
Amount: 100
Submit

*Figure 6-10. The e-commerce form*

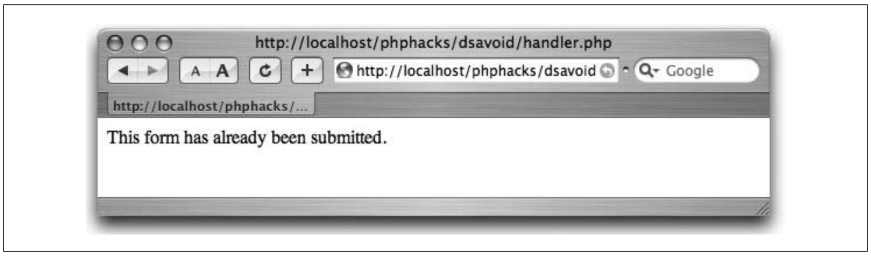
Fill in some bogus data and click Submit. You should see the result shown in Figure 6-11, which shows a successful transaction. This is a good start, as it shows that we can successfully complete a transaction. Now we'll move on to denying redundant transactions.


http://localhost/phphacks/dsavoid/handler.php
http://localhost/phphacks/dsavoid
Google
http://localhost/phphacks/...

Ok, you bought our marvelous product. Thanks!

*Figure 6-11. A successful purchase*

Click the Back button and click Submit again. You should see the result in Figure 6-12.


http://localhost/phphacks/dsavoid/handler.php
http://localhost/phphacks/dsavoid
Google
http://localhost/phphacks/...

This form has already been submitted.

*Figure 6-12. The result of a double submit*

What happened is that *index.php* has requested a unique ID from the *trans.php* script. The *handler.php* script, which receives the form variables, first checks the ID to see whether it has been used already by calling the `check_transid( )` function. If the ID has been used, the code should return the result shown in Figure 6-12.

If the ID is not in the database, we use the `add_transid( )` function to add the ID to the database, and tell the user that the processing has been successful, as shown in Figure 6-11.

The astute reader will note the race condition here. If another form submit comes in between the use of the `check_transid( )` function and the call to the `add_transid( )` function, you could get a double submit that *is* appropriate to process. If your database supports stored procedures, you can write a single transaction that will check to see whether the transaction has completed and then add the transaction to the completed list. This will avoid the race condition and ensure that you cannot have double submits.

At the time of this writing, MySQL did not support stored procedures, though it is in the feature request line for later releases.

## HACK #56 Create User-Customizable Reports

Use a PHP reporting engine that takes an XML definition file and creates a custom report.

Reporting engines allow end users to customize the reports generated in their applications. This is extremely valuable in enterprise applications because these systems rarely are exactly what the customer wants. The ability to tweak the reports, notifications, or other front-facing features is critical for a satisfying user experience.

A reporting engine gives the user a declarative method for specifying a report. The host page sets up the query, gets the data, and then runs the report engine to format the data. Some reporting engines, like RLIB (*http://rlib.sf.net/*), can export not only to HTML, but also to PDF, XML, and other formats. In this hack, I use the PHPReports system (*http://phpreports.sf.net/*) to implement a simple book report.

## The Code

Save the code in Example 6-11 as *index.php*.

*Example 6-11. The PHP that runs the report generator*

```
<?php
require_once( "PHPReportMaker.php" );
$rep = new PHPReportMaker( );
$rep->setUser( "root" );
$rep->setPassword( "" );
$rep->setDatabaseInterface( "mysql" );
$rep->setConnection( "localhost" );
$rep->setDatabase( "books" );
$rep->setSQL( "SELECT NAME,AUTHOR FROM BOOK ORDER BY NAME" );

$rep->setXML( "bookreport.xml" );

$rep->run( );
?>
```

Save the code in Example 6-12 as *bookreport.xml*.

*Example 6-12. The report's XML specification*

```
<REPORT MARGINWIDTH="5" MARGINHEIGHT="5">
<TITLE>Book Report</TITLE>
<CSS>report.css</CSS>
<PAGE BORDER="0" SIZE="10" CELLSPACING="0" CELLPADDING="5">
</PAGE>
<GROUPS>
<GROUP NAME="author" EXPRESSION="AUTHOR">
<HEADER>
<ROW>
<COL CELLCLASS="header"><XHTML><i>Name</i></XHTML></COL>
<COL CELLCLASS="header">Author</COL>
</ROW>
</HEADER>
<FIELDS>
<ROW>
<COL TYPE="FIELD">NAME</COL>
<COL TYPE="FIELD">AUTHOR</COL>
```

*Example 6-12. The report's XML specification (continued)*

```
</ROW>
</FIELDS>
<FOOTER>
</FOOTER>
</GROUP>
</GROUPS>
</REPORT>
```

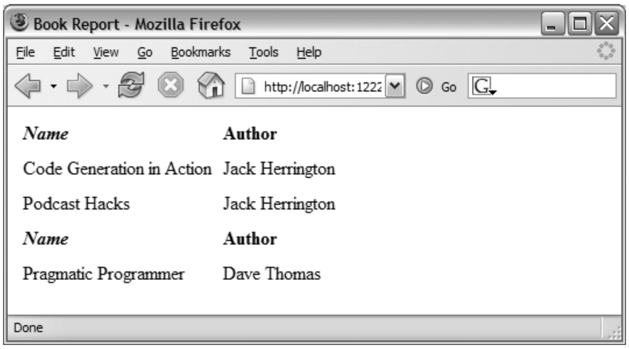
Save the code in Example 6-13 as *report.css*.

*Example 6-13. The CSS for the report*

```
.header { font-weight: bold; }
```

## Running the Hack

Download and install the PHPReports system per the instructions included with the download. Upload the files to the server and navigate to the *index.php* page in your browser. The result should look like Figure 6-13.


Book Report - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost:1222
Go
Done

| *Name* | **Author** |
|---|---|
| Code Generation in Action | Jack Herrington |
| Podcast Hacks | Jack Herrington |
| *Name* | **Author** |
| Pragmatic Programmer | Dave Thomas |

*Figure 6-13. The formatted book report*

Changing the look of the report is as easy as tweaking the XML in the *bookreport.xml* file. The PHPReports system provides for control over the color, fonts, and layout of the report through both HTML and CSS. It also allows for data grouping through the inclusion of dynamic HTML elements like anchor tags and scripts.

PHPReports also has a flexible back end, allowing you to render to a single page of HTML, text, or even a multipage HTML report (the HTML is broken into multiple pages, and the generated markup has navigation at the bottom of each page). You can also create your own output plug-in, allowing you to create whatever output form you need.

## See Also

- "Give Your Customers Formatting Control with XSL" **[Hack #7]**