Adobe® Dreamweaver™ CS5 with PHP
Training from the Source

David Powers

Adobe

Adobe® Dreamweaver™ CS5 with PHP: Training from the Source

David Powers

Adobe Press books are published by:
Peachpit
1249 Eighth Street
Berkeley, CA 94710
510/524-2178
800/283-9444

For the latest on Adobe Press books, go to www.adobepress.com
To report errors, please send a note to errata@peachpit.com
Peachpit is a division of Pearson Education.
Copyright © 2011 David Powers

Acquisitions Editor: Victor Gavenda
Project Editor: Rebecca Freed
Development Editor and Copyeditor: Anne Marie Walker
Production Editor: Becky Winter
Technical Editor: Tom Muck
Compositor: Danielle Foster
Indexer: Rebecca Plunkett
Cover Design: Charlene Charles-Will

ISBN-13: 978-0-321-71984-3
ISBN-10: 0-321-71984-0
9 8 7 6 5 4 3 2 1
Printed and bound in the United States of America

LESSON 7

What You Will Learn

In this lesson, you will:

- Install the Zend Framework and set up site-specific code hints
- Alter a database table to add a unique index and extra columns
- Validate user input on the server with Zend_Validate
- Preserve user input and display error messages when input fails validation
- Create reusable code with Dreamweaver's Server Behavior Builder
- Check for duplicate usernames
- Insert user input into a database with Zend_Db
- Create a login system with Zend_Auth

Approximate Time

This lesson takes approximately 3 hours to complete.

Lesson Files

Media Files:

styles/users.css
styles/users_wider.css

Starting Files:

lesson07/start/add_user.php
lesson07/start/login.php
lesson07/start/members_only.php

Completed Files:

lesson07/completed/add_user.php
lesson07/completed/add_user01.php
lesson07/completed/add_user02.php
lesson07/completed/add_user03.php
lesson07/completed/login.php
lesson07/completed/members_only.php
lesson07/completed/scripts/library.php
lesson07/completed/scripts/library_magic_quotes.php
lesson07/completed/scripts/restrict_access.php
lesson07/completed/scripts/user_authentication.php
lesson07/completed/scripts/user_authentication01.php
lesson07/completed/scripts/user_registration.php
lesson07/completed/scripts/user_registration01.php
lesson07/completed/scripts/user_registration02.php
lesson07/completed/scripts/user_registration03.php

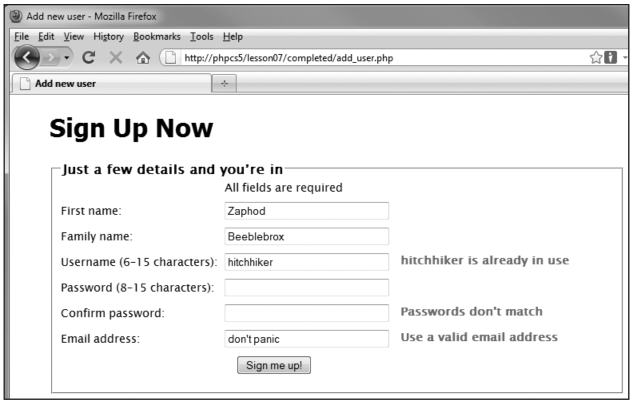
LESSON 7

Validating Input on the Server

Using JavaScript for validation is not enough on its own, because it takes only a few seconds to turn off JavaScript in a browser, rendering your validation script useless. Before inserting user input into a database, you must validate it on the server with a server-side language, such as PHP.

In this lesson, with the help of a powerful third-party script library—Zend Framework—you'll create a robust user registration system that validates input on the server. This involves writing your own PHP code rather than relying on Dreamweaver to generate it for you. Any inconvenience is more than made up for in greater functionality, and the process is simplified by site-specific code hints.

You'll also use Dreamweaver's Server Behavior Builder to speed up the insertion of frequently used PHP code.


Add new user - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://phpcs5/lesson07/completed/add_user.php
Add new user

Sign Up Now

Just a few details and you're in

All fields are required

First name:	Zaphod	
Family name:	Beeblebrox	
Username (6–15 characters):	hitchhiker	hitchhiker is already in use
Password (8–15 characters):		
Confirm password:		Passwords don't match
Email address:	don't panic	Use a valid email address

Sign me up!

The registration form alerts the user to errors before inserting details into the database.

Introducing the Zend Framework

The Zend Framework (ZF) is a huge open source library of PHP scripts. The "minimal" version of ZF 1.10 consists of more than 2,700 files in nearly 500 folders, and is more than 22 MB in size. Of course, size isn't everything. In fact, you might wonder why you need so many files to do a few basic tasks, such as inserting and updating records in a database, uploading files, and sending emails.

Frameworks provide a wide range of options, many of which you might never use. For example, ZF has 14 components that access different e-commerce services on Amazon.com. Most developers never use them, but they're indispensible if you have an Amazon affiliate account. Instead of writing a script of mind-numbing complexity, you can query the Amazon database with just a few lines of code.

Most tutorials assume you want to use ZF to build a web application using the Model-View-Controller (MVC) design pattern, which divides data handling (the model), output (the view), and conditional logic (the controller) among separate scripts. Unless you have considerable PHP experience, the MVC design pattern can be confusing, so this book takes a different approach, offering a gentler introduction to ZF.

ZF is a *loosely coupled* framework, which means each part is designed to work with minimal dependency on other parts. You can use just a few components without needing to learn the whole framework. But should you decide later to adopt MVC, your knowledge of key ZF components will speed up the transition.

Here are the main components you'll be using in the remaining lessons:

- `Zend_Auth` to check user credentials
- `Zend_Db` to communicate with a database
- `Zend_File` to upload files
- `Zend_Loader` to load ZF classes automatically
- `Zend_Mail` to send emails and attachments
- `Zend_Paginator` to page through a long series of database results
- `Zend_Service_ReCaptcha` to prevent spam input
- `Zend_Validate` to validate user input

The exercises continue using MySQL, but `Zend_Db` makes the code more flexible. Changing just one or two lines allows you to switch to Microsoft SQL Server, PostgreSQL, or another database system.

ZF has a number of other factors in its favor:

- Its principal sponsor is Zend Technologies, the company founded and run by PHP core contributors.
- Leading software companies, including Adobe, Google, and Microsoft, have contributed components or significant features. Adobe contributed `Zend_Amf`, which acts as a gateway between PHP and the Flash Player, using the binary Action Message Format (AMF).
- ZF was designed from the outset to use PHP 5 objects. Many frameworks were originally designed for compatibility with PHP 4, which is less efficient.
- The next major version of ZF is being designed to enhance interoperability with other frameworks, such as Symfony.

TIP: ZF is an object-oriented framework. If you're not familiar with PHP objects, now is a good time to review "Using Objects and Resources" in Lesson 3.

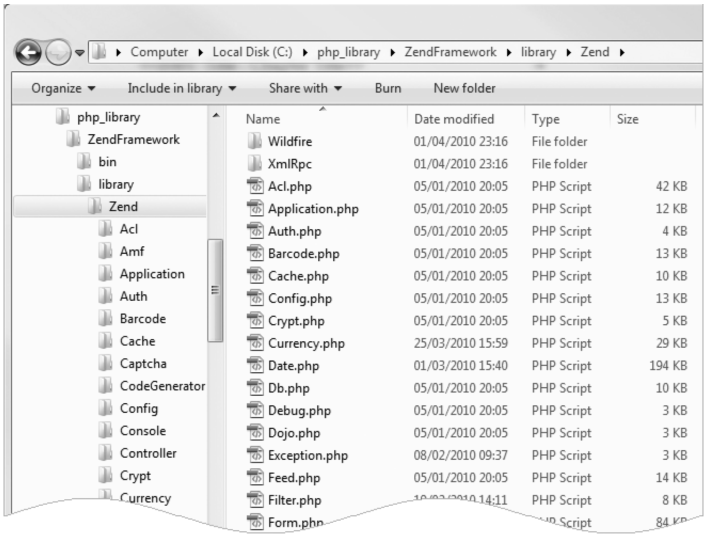
Installing the Zend Framework

The CD accompanying this book contains ZendFramework-1.10.6-minimal.zip. Alternatively, download the most recent version from http://framework.zend.com/download/latest. Choose the Minimal version of ZF, which contains all the files you need. Some download links require you to establish a Zend account. Like ZF, this is free and does not entail any obligations.

To install ZF, simply unzip the file to a suitable location on your hard disk. It's more efficient to locate it outside the `phpcs5` site root, so it's accessible to all sites in Dreamweaver. Create a folder called php_library at the top level of your C drive on Windows or in your home folder on a Mac, and unzip ZF there.

Two folders, bin and library, are inside the main folder. The framework is in a subfolder of library called Zend. Each component has a corresponding `.php` file in the Zend folder plus a folder of its own.

There's no need to open the files, but if you do, you'll see that each file is extensively commented, which partly explains why the framework is so big. As you gain more experience, you'll discover a lot of useful information in these comments. Sometimes they help explain features that are not fully documented.


Computer ▸ Local Disk (C:) ▸ php_library ▸ ZendFramework ▸ library ▸ Zend ▸
Organize
Include in library
Share with
Burn
New folder

php_library
ZendFramework
bin
library
Zend
Acl
Amf
Application
Auth
Barcode
Cache
Captcha
CodeGenerator
Config
Console
Controller
Crypt
Currency

Name	Date modified	Type	Size
Wildfire	01/04/2010 23:16	File folder	
XmlRpc	01/04/2010 23:16	File folder	
Acl.php	05/01/2010 20:05	PHP Script	42 KB
Application.php	05/01/2010 20:05	PHP Script	12 KB
Auth.php	05/01/2010 20:05	PHP Script	4 KB
Barcode.php	05/01/2010 20:05	PHP Script	13 KB
Cache.php	05/01/2010 20:05	PHP Script	10 KB
Config.php	05/01/2010 20:05	PHP Script	13 KB
Crypt.php	05/01/2010 20:05	PHP Script	5 KB
Currency.php	25/03/2010 15:59	PHP Script	29 KB
Date.php	01/03/2010 15:40	PHP Script	194 KB
Db.php	05/01/2010 20:05	PHP Script	10 KB
Debug.php	05/01/2010 20:05	PHP Script	3 KB
Dojo.php	05/01/2010 20:05	PHP Script	3 KB
Exception.php	08/02/2010 09:37	PHP Script	3 KB
Feed.php	05/01/2010 20:05	PHP Script	14 KB
Filter.php	10/03/2010 14:11	PHP Script	8 KB
Form.php		HP Script	84 KB

TIP: The ZF documentation at http://framework.zend.com/manual/en/ is extensive and contains many examples. Unfortunately, much of it is written on the assumption that you are using the MVC design pattern. It also expects you to have a solid understanding of PHP. However, considerable efforts have been made to improve it.

Setting up site-specific code hints for ZF

In Lesson 4, you learned how to create site-specific code hints for WordPress, Drupal, and Joomla! Dreamweaver automatically recognizes the structure of these CMSs. With other frameworks, including ZF, you need to tell Dreamweaver where to find the files and which ones you want to scan to generate code hints. You don't need the WordPress code hints again in this book, so the following instructions show how to set up a separate structure for ZF hints. This replaces the existing version of dw_php_codehinting.config in the phpcs5 site, but you can easily switch between different sets of code hints by selecting them in the Structure menu of the Site-Specific Code Hints dialog box.

TIP: You can also follow these steps in the final section of my Adobe TV video about PHP code hints at http://tv.adobe.com/watch/learn-dreamweaver-cs5/using-php-code-hinting-in-dreamweaver-cs5.

1 In the Dreamweaver Files panel, select the PHP CS5 site.

2 Make sure the active file in the Document window is from the current site, or close all files.

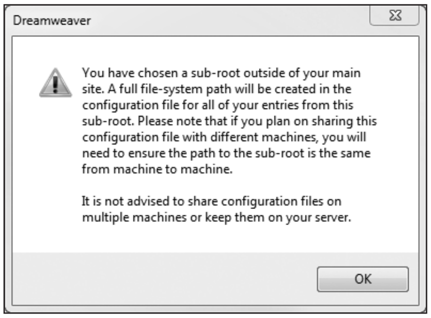
3 Choose Site > Site-Specific Code Hints.

4 Make sure the Structure menu is set to <New from Blank>.

5 Click the "Select sub-root folder" icon next to the Sub-root text box, and navigate to the library folder one level above the Zend folder.

✱ **NOTE:** The sub-root folder must be at least one level higher than any files or folders that you want to scan.

6 Click Select. Because the folder is outside the site root, Dreamweaver displays the following alert.

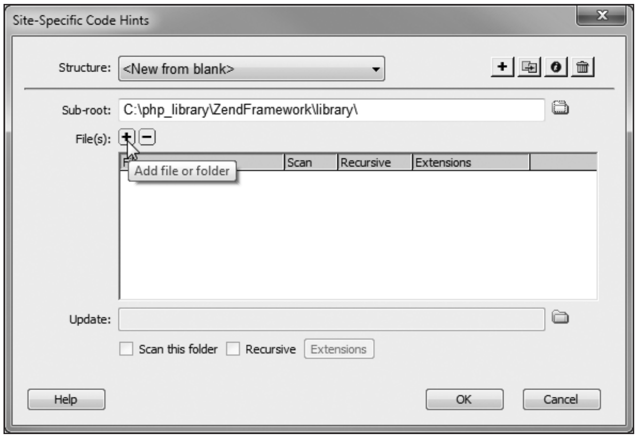


You have chosen a sub-root outside of your main site. A full file-system path will be created in the configuration file for all of your entries from this sub-root. Please note that if you plan on sharing this configuration file with different machines, you will need to ensure the path to the sub-root is the same from machine to machine.

It is not advised to share configuration files on multiple machines or keep them on your server.

Just click OK.

7 Click the plus (+) button above the File(s) area to open the Add File/Folder dialog box.



Structure: <New from blank>

Sub-root: C:\php_library\ZendFramework\library\

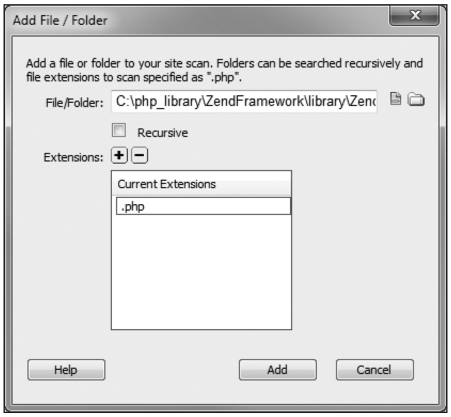
File(s):

Update:

8 Click the "Select folder" icon next to the File/Folder text box, and select the Zend folder.

9 The Recursive checkbox tells Dreamweaver whether to scan all subfolders. If you select the checkbox for the Zend folder, code hints are enabled for the entire framework. Because ZF contains so many files and folders, scanning all of them is not a good idea, so leave the checkbox deselected.

10 To make the scanning process more efficient, click the plus button above the Extensions area, and type **.php** in the highlighted section. This tells Dreamweaver to scan only `.php` files.



Add File / Folder

Add a file or folder to your site scan. Folders can be searched recursively and file extensions to scan specified as ".php".

File/Folder: C:\php_library\ZendFramework\library\Zenc

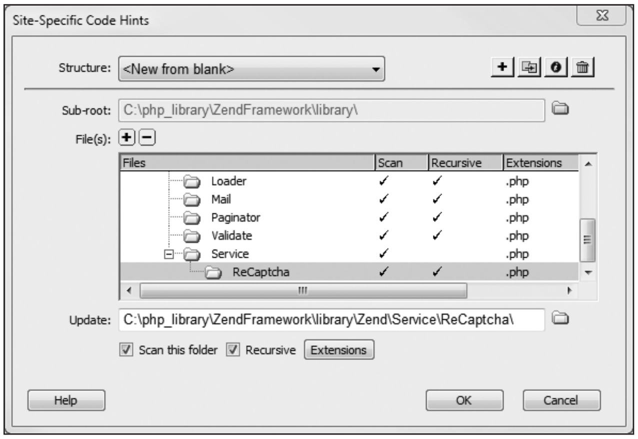
☐ Recursive

Extensions:

Current Extensions
.php

11 Click Add, and repeat steps 7–10 to add the following folders, all of which are subfolders of Zend: Auth, Captcha, Db, File, Loader, Mail, Paginator, and Validate. When setting up each folder, select the Recursive checkbox and add `.php` to the Extensions list.

You also need to add the Service folder and one of its subfolders, ReCaptcha. The Service folder contains many subfolders, so select the Recursive checkbox only for the ReCaptcha subfolder.



Site-Specific Code Hints

Structure: <New from blank>

Sub-root: C:\php_library\ZendFramework\library\

File(s):

Files	Scan	Recursive	Extensions
Loader	✓	✓	.php
Mail	✓	✓	.php
Paginator	✓	✓	.php
Validate	✓	✓	.php
Service	✓		.php
ReCaptcha	✓	✓	.php

Update: C:\php_library\ZendFramework\library\Zend\Service\ReCaptcha\

☑ Scan this folder ☑ Recursive

If you forget to select the Recursive checkbox or add the `.php` filename extension, you can change the options using the checkbox and button at the bottom of the Site-Specific Code Hints dialog box.

12 Save the configuration by clicking the Save Structure icon at the top right of the dialog box.

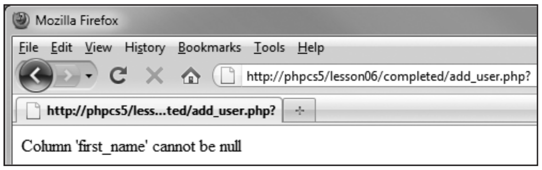
13 Type **Zend** in the Name text box (you can use any name except Custom, Drupal, Joomla, or WordPress), and click Save.

14 Click OK to close the Site-Specific Code Hints dialog box. This inserts a file called dw_php_codehinting.config in the site root. Dreamweaver uses this to scan the ZF folders and create code hints for the selected components.

TIP: The folders are scanned in reverse alphabetical order each time you launch Dreamweaver. This process shouldn't slow down the program, but if you select the whole framework, it can take several minutes before all code hints are ready for use.

Improving the Registration Form

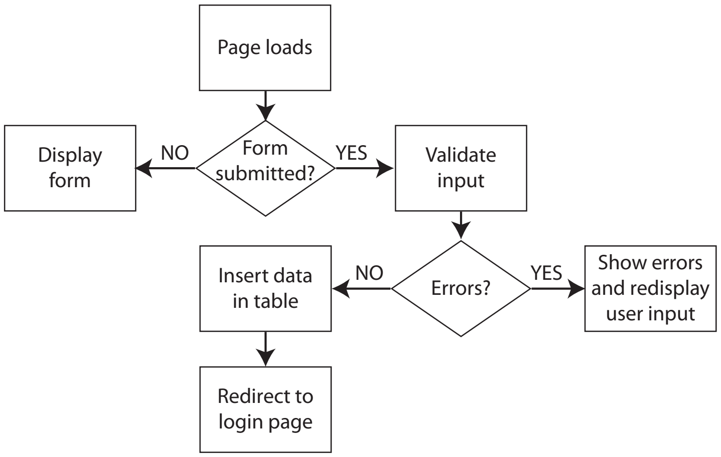
The registration form created in Lesson 6 has serious flaws. There's no control over the values entered in each input field. If you submit the form without filling in any of the fields, MySQL stops you, but the server behavior code leaves you with this unhelpful message and no way to get back to the form.


Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://phpcs5/lesson06/completed/add_user.php?
http://phpcs5/less...ted/add_user.php?

```
Column 'first_name' cannot be null
```

This is MySQL's way of saying that `first_name` is a required field, but you need a better way of conveying that message to the user.

What's more, a series of blank spaces is accepted as valid input, so you could end up with a completely blank record. There's also the problem of duplicate usernames, not to mention setting a minimum length for the password. The registration form needs to check all the user input before attempting to insert it into the database and to redisplay the form with error messages if validation fails, as shown in the following diagram.



Additionally, there's the question of forgotten passwords. In Lesson 8, you'll learn how to send users an email to reset their password, so you need to add extra columns to the `users` table—one to store email addresses and the other for a security token.

There's a lot to fix. Let's start by updating the `users` table.

Adding a unique index to the users table

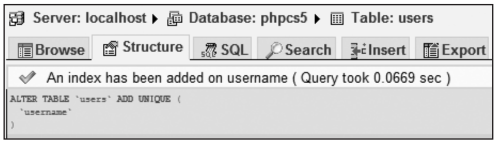
A *unique index* prevents duplicate values from being inserted in a database column. Adding an index to a column is quick and easy in phpMyAdmin.

1 Open phpMyAdmin, and select the `users` table in the `phpcs5` database.

2 If you have any records with duplicate usernames, click the Delete icon ✕ next to the record you want to delete. Leave at least one record in the table, because you need it for testing later.

3 Click the Structure tab at the top left of the screen to display the definition of the `users` table.

4 Click the Unique icon in the Action section of the `username` row. When the page reloads, you should see confirmation that an index has been added to username. The SQL command used to create the index is displayed immediately below. Check that it says `ADD UNIQUE`.



An index has been added on username (Query took 0.0669 sec)

```
ALTER TABLE `users` ADD UNIQUE (
  `username`
)
```

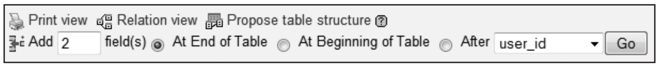
5 If you clicked the wrong icon and created a different type of index or a unique index on the wrong column, click the Details link at the bottom of the page to reveal the Indexes section, and delete the index you have just created; then repeat step 4 to add a unique index on `username`.

Leave phpMyAdmin open with the Structure tab selected to continue with the instructions in the next section.

Adding extra columns to the users table

It takes only a couple of minutes to add a column to a database table in phpMyAdmin. Changing the structure of a database is simple, but it should normally be done only in the development stage. Once you start filling the database with records, you risk losing data or having incomplete records.

1 With the Structure tab of the `users` table selected in phpMyAdmin, locate "Add field(s)" toward the bottom of the screen. Type **2** in the text field, leave the "At End of Table" radio button selected, and click Go.

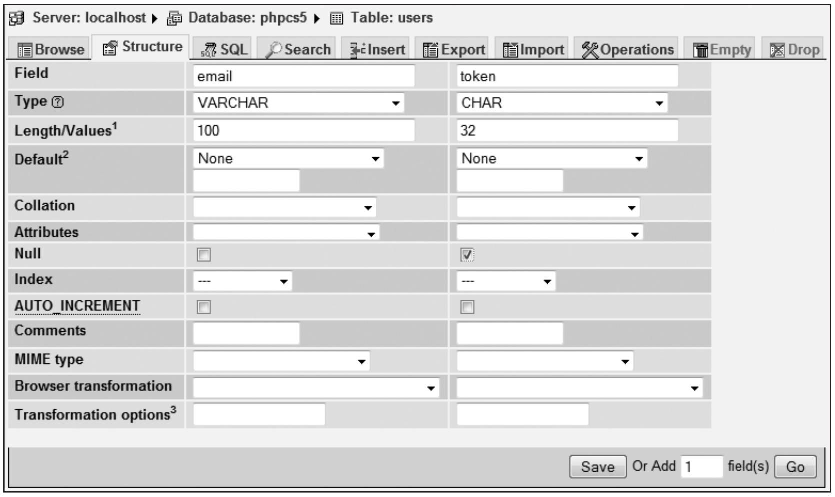

Print view
Relation view
Propose table structure
Add 2 field(s) [✓ At End of Table] At Beginning of Table After user_id
Go

NOTE: The other radio buttons let you specify where the new column(s) are to be inserted. If you select the After radio button, phpMyAdmin inserts the new column(s) in the middle of the table after the column chosen from the list.

This presents you with a matrix where you define the two new columns. Because there are only two, the options are listed vertically, which makes them easier to see.

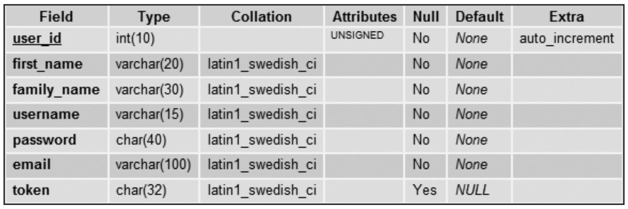
2 For the `email` column, type **email** in Field, set Type to `VARCHAR`, and Length/Values to **100**.

The token will be a randomly generated, fixed-length string. For the other column, type **token** in Field, set Type to `CHAR`, and Length/Values to **32**. Also select the Null checkbox to make this column optional.



Field	email	token
Type	VARCHAR	CHAR
Length/Values[1]	100	32
Default[2]	None	None
Collation		
Attributes		
Null	☐	✓
Index	---	---
AUTO_INCREMENT	☐	☐
Comments		
MIME type		
Browser transformation		
Transformation options[3]		

3 Click Save. The revised table structure should look like this:



Field	Type	Collation	Attributes	Null	Default	Extra
user_id	int(10)		UNSIGNED	No	*None*	auto_increment
first_name	varchar(20)	latin1_swedish_ci		No	*None*	
family_name	varchar(30)	latin1_swedish_ci		No	*None*	
username	varchar(15)	latin1_swedish_ci		No	*None*	
password	char(40)	latin1_swedish_ci		No	*None*	
email	varchar(100)	latin1_swedish_ci		No	*None*	
token	char(32)	latin1_swedish_ci		Yes	*NULL*	

CAUTION! If you click Go instead of Save, phpMyAdmin adds the options for another column. Give the column a dummy name, and select `INT` as Type. After you click Save, delete the unwanted column by clicking the Delete icon ✕ in the table structure.

There is no need to update the existing record(s) in the `users` table. They can be deleted after you have tested the script later in this lesson.

Loading ZF class files

Before you can use ZF classes and objects, you need to include the definition files into each page. With such a large framework, it would be cumbersome to include each file individually, so ZF provides an autoloader. This loads only those class definitions that are needed for the current script. For it to work, you need to add the library folder to your `include_path`, where PHP looks for include files.

1 Create a new PHP file, and save it as **library.php** in lesson07/workfiles/scripts.

2 Switch to Code view, and delete all the HTML code. You should have a completely blank file.

3 Add an opening PHP tag at the top of the file. Do *not* create a matching closing PHP tag.

4 On the next line, assign the absolute path to the library folder to `$library`.

The value depends on your operating system and where you saved ZF.

- On Windows, it should look similar to this:

```
$library = 'C:/php_library/ZendFramework/library';
```

You can use either forward slashes or backslashes in the path, but it's more common to use forward slashes.

- On Mac OS X, it should look something like this:

```
$library = '/Users/username/php_library/ZendFramework/library';
```

Note that the path begins with a forward slash. Replace **`username`** with your own Mac username.

NOTE: This path needs to be changed when you upload the site to your remote server, but it's in only one file, so it's not a major problem.

5 The value of `include_path` is specified in php.ini, but you don't always have access to php.ini on shared hosting, so you can use the `set_include_path()` function to change it on the fly. Add the following code on the next line:

```
set_include_path(get_include_path() . PATH_SEPARATOR . $library);
```

Rather than overwriting the existing value of `include_path`, you need to add `$library` to it. The existing value is retrieved by `get_include_path()`. Each path needs to be separated by a semicolon on Windows or a colon on Mac/Linux. To make the code portable between different operating systems, the constant `PATH_SEPARATOR` inserts the appropriate separator automatically. Everything is joined with the concatenation operator (a period or dot).

TIP: Press Ctrl+spacebar to invoke code hints to speed up the creation of this line of code and ensure its accuracy.

6 To use the autoloader, you need to include the class file for `Zend_Loader_Autoloader` like this:

```
require_once('Zend/Loader/Autoloader.php');
```

This is the only ZF file that you need to load explicitly. You don't need to use a fully qualified path to the Zend folder, because the code in the previous step added its parent folder, library, to the PHP `include_path`.

7 Now invoke the autoloader like this:

```
$loader = Zend_Loader_Autoloader::getInstance();
```

Technically speaking, you don't need to assign the result to a variable, but it's useful to do so to check that everything is working correctly.

8 To test your script so far, add the following:

```
if ($loader) {
  echo 'OK';
} else {
  echo 'We have a problem';
}
```

9 Save library.php, and click Live View to test the page. If everything is OK, you should see OK onscreen. If you see "We have a problem," read the error message(s). The most likely cause is a mistake in the path to the library folder. Also, check the spelling of all the functions and make sure `PATH_SEPARATOR` is all uppercase.

10 Once everything is working, remove the conditional statement that you added in step 8. You can also remove the `$loader` variable from step 7. The code in your page should look like this (the value of `$library` depends on your setup):

```
<?php
$library = 'C:/php_library/ZendFramework/library';
set_include_path(get_include_path() . PATH_SEPARATOR . $library);
require_once('Zend/Loader/Autoloader.php');
Zend_Loader_Autoloader::getInstance();
```

Do *not* add a closing PHP tag. See the sidebar, "Omitting the Closing PHP Tag," for an explanation.

Connecting to the database with Zend_Db

In ZF, all communication with a database is done through a `Zend_Db` object. This is similar to setting up a MySQL connection for Dreamweaver's server behaviors, but it has two significant advantages:

- A `Zend_Db` object doesn't connect to the database until it's needed. This puts less strain on the database than a Dreamweaver MySQL connection, which always connects, even if the script doesn't need it.

Omitting the Closing PHP Tag

When a script ends with PHP code, the closing PHP tag ?> is optional. In fact, the ZF coding standard actually forbids its use in pages that contain only PHP code. Leaving out the closing PHP tag prevents problems with the "headers already sent" error (see the sidebar "Why the Next Page Doesn't Always Load" in Lesson 6) and prevents a lot of hair tearing.

Dreamweaver CS5 supports the omission of the closing PHP tag. However, as you'll see in Lesson 8, leaving out the closing tag of an include file sometimes switches Design view into CSS quirks mode. If this hinders you, add a closing tag, but make sure it's not followed by any whitespace or newline characters.

The closing tag should always be used if HTML follows the PHP script in the *same* page. It's OK if the HTML is in a parent page. The PHP engine automatically switches back to HTML mode at the end of an include file.

- `Zend_Db` has several adapter subclasses that connect to different database systems. To connect to a different database, just create a `Zend_Db` object using the appropriate subclass. Normally, this involves a single line of code unless your SQL uses database-specific functions.

Since most pages require a database connection, it makes sense to instantiate the `Zend_Db` object in the same file that loads the ZF classes.

1 To connect to a database, you need to supply the location of the database server, the username and password of the account you want to use, and the name of the database. You pass these details as an associative array (see "Creating an associative array" in Lesson 3) to the `Zend_Db` constructor, using the array keys `'host'`, `'username'`, `'password'`, and `'dbname'`.

Create an array for the `cs5write` user account at the bottom of library.php like this:

```
$write = array('host'     => 'localhost',
               'username' => 'cs5write',
               'password' => 'Bow!e#CS5',
               'dbname'   => 'phpcs5');
```

TIP: It's not essential to indent an associative array and line up the => operators like this, but it makes your code easier to read and debug.

2 Create another array for the `cs5read` account directly below.

```
$read  = array('host'     => 'localhost',
               'username' => 'cs5read',
               'password' => '5T@rmaN',
               'dbname'   => 'phpcs5');
```

3 Your choice of `Zend_Db` adapter depends on the database you want to use and the PHP configuration of your remote server. If your remote server supports pdo_mysql, use this:

```
$dbWrite = new Zend_Db_Adapter_Pdo_Mysql($write);
```

If your remote server supports only mysqli, use this:

```
$dbWrite = new Zend_Db_Adapter_Mysqli($write);
```

TIP: You don't need to type out the whole class name. As explained in Lesson 1, Dreamweaver's code hints ignore underscores and recognize substrings within names. Typing **pdomy** takes you directly to `Zend_Db_Adapter_Pdo_Mysql`. Just press Enter/Return as soon as it's highlighted. Also, be careful when passing `$write` as the argument to the constructor. Because it's an array, Dreamweaver automatically adds an opening square bracket, which you need to remove.

4 Create another object for the `cs5read` account, using the appropriate adapter:

```
$dbRead = new Zend_Db_Adapter_Pdo_Mysql($read);
```

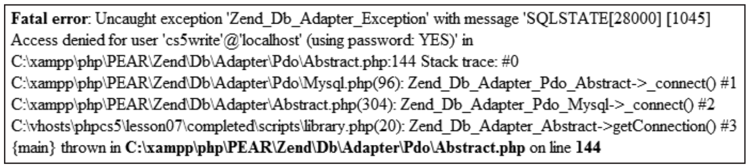
Or

```
$dbRead = new Zend_Db_Adapter_Mysqli($read);
```

5 Because a `Zend_Db` object doesn't connect to the database until it's needed, it's a good idea to make a test connection to ensure your code is OK. Add these conditional statements at the end of library.php:

```
if ($dbWrite->getConnection()) {
  echo 'Write OK<br />';
}
if ($dbRead->getConnection()) {
  echo 'Read OK';
}
```

When you type the -> after the object, code hints should show you the methods it can use. The `getConnection()` method has a self-explanatory name. If each connection is OK, the conditional statements display confirmation. If there's a problem, you'll see a fatal error similar to this:



```
Fatal error: Uncaught exception 'Zend_Db_Adapter_Exception' with message 'SQLSTATE[28000] [1045]
Access denied for user 'cs5write'@'localhost' (using password: YES)' in
C:\xampp\php\PEAR\Zend\Db\Adapter\Pdo\Abstract.php:144 Stack trace: #0
C:\xampp\php\PEAR\Zend\Db\Adapter\Pdo\Mysql.php(96): Zend_Db_Adapter_Pdo_Abstract->_connect() #1
C:\xampp\php\PEAR\Zend\Db\Adapter\Abstract.php(304): Zend_Db_Adapter_Pdo_Mysql->_connect() #2
C:\vhosts\phpcs5\lesson07\completed\scripts\library.php(20): Zend_Db_Adapter_Abstract->getConnection() #3
{main} thrown in C:\xampp\php\PEAR\Zend\Db\Adapter\Pdo\Abstract.php on line 144
```

Don't panic. The important information is in the second line, which says access was denied for `cs5write` and that a password was used. This normally means the password was wrong.

Another possible cause is choosing the wrong adapter class. It's easy to mix up `Zend_Db_Adapter_Pdo_Mssql` with `Zend_Db_Adapter_Pdo_Mysql`. The former is for Microsoft SQL Server. If you make this mistake, the error message is likely to tell you that the `mssql` driver is not installed. If it is installed, you might be trying to connect to the wrong database server.

Check your code, paying particular attention to spelling and case sensitivity.

6 After verifying that your connections are working, delete the code you added in step 5. It's not needed any more.

Leave library.php open to continue working with it in the next section.

NOTE: For details of the `Zend_Db` adapter classes for other databases, see http://framework.zend.com/manual/en/zend.db.adapter.html.

Handling exceptions with try and catch

ZF is an object-oriented framework. If an error occurs in any part of the script, it *throws an exception*. Unlike an ordinary PHP error, which displays the error message at the point in the script where it occurs, an exception can be handled in a different part of the script. If you look closely at the first line of the fearsome error message in the preceding screen shot, you'll see it refers to an "uncaught exception." When you throw something, it needs to be caught.

To prevent this sort of unsightly error message, you should always wrap object-oriented code in `try` and `catch` blocks like this:

```
try {
  // main script
} catch (Exception $e) {
  echo $e->getMessage();
}
```

The main script goes between the curly braces of the `try` block, where PHP tries to run the code. If all is well, the code is executed normally, and the `catch` block is ignored. If an exception is thrown, the script inside the `try` block is abandoned, and the `catch` block runs instead.

Objects can define many different types of exceptions, so you can have different `catch` blocks to handle each type separately. The `Exception` in the parentheses after `catch` indicates it's a generic `catch` block to handle all exceptions. The exception is assigned to the variable `$e` so you can access any messages it contains. At the moment, the `catch` block just uses `echo` and the `getMessage()` method to display the error message. When the script is ready to be deployed in a real site, you replace the code in the `catch` block with a more elegant way of handling the problem, such as displaying an error page.

You need to wrap most of the code in library.php in a `try` block, and add a `catch` block at the bottom of the page.

1. Position the insertion point at the end of the following line, and press Enter/Return to insert a blank line:

   ```
   require_once('Zend/Loader/Autoloader.php');
   ```

2. On the new line, type **try**, followed by an opening curly brace.
3. Select all the code on the following line to the bottom of the page, and click the Indent Code icon in the Coding toolbar to indent the code in the `try` block.
4. Add a new line at the bottom of the page, and insert the closing brace of the `try` block, together with a `catch` block like this:

   ```
   } catch (Exception $e) {
     echo $e->getMessage();
   }
   ```

5. Save library.php. You can compare your code with lesson07/completed/library.php.

Using Zend_Validate to check user input

The standard way of validating user input on the server is to create a series of conditional statements to test if a value meets certain criteria. For example, if you want to check whether a password contains between 8 and 15 characters, you can use the PHP function `strlen()`, which returns the length of a string, like this:

```
if (strlen($_POST['password']) >= 8 && strlen($_POST['password']) <= 15) {
  // it's valid
}
```

This works, but it doesn't check what characters are used in the password. Pressing the spacebar eight times passes this test. So, you need to add other conditional statements to make sure all criteria are met.

`Zend_Validate` works in a similar way but provides a set of commonly used validators. Each subclass has an easily recognizable name that makes your validation script much easier to read, and you don't need to become an expert in PHP functions to ensure that user input matches your requirements. **Table 7.1** lists the most commonly used subclasses. Each one is prefixed by `Zend_Validate_`, so `Alnum` becomes `Zend_Validate_Alnum`.

For example, to check that a string is 8–15 characters, use `Zend_Validate_StringLength` like this:

```
$val = new Zend_Validate_StringLength(8,15);
```

This instantiates a `Zend_Validate_StringLength` object, setting its minimum and maximum values to 8 and 15 respectively, and assigns it to `$val`.

To check whether `$_POST['password']` contains between 8 and 15 characters, pass it as an argument to the `isValid()` method like this:

```
$val->isValid($_POST['password'])
```

If `$_POST['password']` contains 8–15 characters, this returns `TRUE`. Otherwise, it returns `FALSE`.

Normally, if a validation test fails, you want to generate an error message. Do this by using a conditional statement with the logical Not operator (see "Using the logical Not operator" in Lesson 3) like this:

```
$val = new Zend_Validate_StringLength(8,15);
if (!$val->isValid($_POST['password'])) {
  $errors['password'] = 'Password should be 8-15 characters';
}
```

Adding the logical Not operator looks for a value that is *not* valid, so the error message is assigned to `$errors['password']` only if `$_POST['password']` is *not* 8–15 characters.

This validation test is fine as far as it goes, but it has the same problem as the earlier example: It checks only the number of characters. Pressing the spacebar 8–15 times still passes validation. You need to combine validators. One way is to use a series of conditional statements, but ZF offers another solution—chaining validators.

Table 7.1 **Commonly Used Validation Classes**

Class	Description
`Alnum`	Checks that the value contains only alphabetic and number characters. Whitespace characters are permitted if TRUE is passed as an argument to the constructor.
`Alpha`	Same as `Alnum` except numbers are not permitted.
`Between`	Accepts a value between minimum and maximum limits. Constructor requires two arguments, which can be numbers or strings, to set the limits. By setting an optional third argument to TRUE, the value cannot be equal to either the maximum or minimum.
`CreditCard`	Checks whether a value falls within the ranges of possible credit card numbers for most leading credit card issuers. Does *not* check whether the number is genuine.
`Date`	Checks not only that a date is in the `'YYYY-MM-DD '` format, but also that it's a valid date. For example, `'2010-2-30'` fails because it's not a real date, although it's in the right format.
`Digits`	Accepts only digits. The decimal point and thousands separator are rejected.
`EmailAddress`	Validates an email address. Has the option to check whether the hostname actually accepts email, but this slows down performance. On Windows, this option requires PHP 5.3 or later.
`Float`	Accepts a floating point number. The maximum value is platform-dependent.
`GreaterThan`	Checks that a value is greater than a minimum. Constructor takes a single argument to set the minimum value.
`Identical`	Checks that a value is identical to the value passed as an argument to the constructor.
`Int`	Accepts an integer.
`LessThan`	Checks that a value is less than a maximum. Constructor takes a single argument to set the maximum value.
`NotEmpty`	Checks that a value is not empty. Various options can be set to configure what is regarded as an empty value, offering greater flexibility than the PHP `empty()` function.
`PostCode`	Checks that a value conforms to the pattern for a postal or zip code. The pattern is determined by passing a locale string to the constructor, for example, `'en_US'` for the United States or `'en_GB'` for the UK.
`Regex`	Validates against a regular expression passed as an argument to the constructor.
`StringLength`	Checks the length of a string. The constructor accepts one, two, or three arguments. The first sets the minimum length, the second optionally sets the maximum length, and the third optionally specifies the encoding. Alternatively, these values can be presented as an associative array using the keys `'min'`, `'max'`, and `'encoding'`.

Chaining validators to set multiple criteria

To test for more than one criterion, create a generic `Zend_Validate` object, and use its `addValidator()` method to add each new test. You can instantiate each validator separately, and then pass it as an argument to `addValidator()` like this:

```
$val = new Zend_Validate();
$val1 = new Zend_Validate_StringLength(8, 15);
$val2 = new Zend_Validate_Alnum();
$val->addValidator($val1);
$val->addValidator($val2);
```

However, it's simpler to instantiate each validator directly as the argument to `addValidator()` like this:

```
$val = new Zend_Validate();
$val->addValidator(new Zend_Validate_StringLength(8, 15));
$val->addValidator(new Zend_Validate_Alnum());
```

You can even chain the `addValidator()` methods one after the other like this:

```
$val = new Zend_Validate();
$val->addValidator(new Zend_Validate_StringLength(8, 15))
    ->addValidator(new Zend_Validate_Alnum());
```

Notice that there is no semicolon at the end of the second line, and the second -> operator isn't prefixed by the `$val` object. Indenting it like this makes the code easier to read, but you could place it immediately after the closing parenthesis at the end of the first `addValidator()` method.

NOTE: Chaining methods like this will be familiar to readers with jQuery experience. Unfortunately, Dreamweaver CS5's code hints don't support chaining methods, so the code in this book always uses separate statements to apply methods to ZF objects.

All three sets of code perform the same task: `$val` tests for a string 8–15 characters long that contains only letters and numbers, with no spaces.

Armed with this knowledge, you can validate the input of the registration form.

Building the validation script (1)

The user registration form from Lesson 6 has been modified to add a text input field for the email address and some hints for the user. The style sheet has also been changed to make room for error messages.

1 Copy add_user.php from lesson07/start to lesson07/workfiles.

2 It's more efficient to use an external file for the validation code so you can reuse the code for other projects. Choose File > New, and create a new PHP page. Save it as **user_registration.php** in lesson07/workfiles/scripts.

3 In the file you just created, switch to Code view, delete the HTML code inserted by Dreamweaver, and add an opening PHP tag at the top of the page. This page will contain only PHP, so it shouldn't have a closing PHP tag.

4 After the opening PHP tag, initialize an array to store error messages:

```
$errors = array();
```

5 When the form is first loaded, there's nothing to process, so the `$_POST` array is empty. An empty array is treated as `FALSE` (see "What PHP regards as false" in Lesson 3), so you can use this to ensure that the validation script is run only when the form is submitted. Add a conditional statement like this:

```
if ($_POST) {
  // run the validation script
}
```

6 The validation script needs access to the ZF files. Include library.php by adding it between the curly braces of the conditional statement:

```
require_once('library.php');
```

7 Add `try` and `catch` blocks inside the conditional statement created in the previous step:

```
if ($_POST) {
  // run the validation script
  require_once('library.php');
  try {
    // main script goes here
  } catch (Exception $e) {
    echo $e->getMessage();
  }
}
```

8 The first input field you need to validate is `first_name`. Personal names are alphabetic, so `Zend_Validate_Alpha` seems like a good choice. Add the following code inside the `try` block:

```
try {
  // main script goes here
  $val = new Zend_Validate_Alpha(TRUE);
  if (!$val->isValid($_POST['first_name'])) {
    $errors['first_name'] = 'Required field, no numbers';
  }
} catch (Exception $e) {
```

By passing `TRUE` as an argument, this permits spaces.

9 Before going any further, it's a good idea to test the script so far. Save user_registration.php, and switch to add_user.php.

Include user_registration.php by inserting space above the `DOCTYPE` declaration and adding the following code:

```
<?php
require_once('scripts/user_registration.php');
?>
```

10 To display error messages next to each input field, you need to add a pair of `<span>` tags with a PHP conditional statement in between.

Locate the following line in Code view:

```
<input type="text" name="first_name" id="first_name" />
```

TIP: It's a good idea to work in Vertical Split view (choose View > Split Vertically and click Split in the Document toolbar). Select the input field in Design view to highlight the `<input>` tag in Code view.

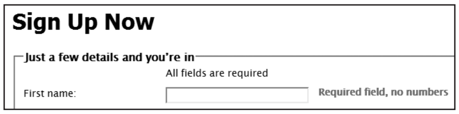
11 Add the following code after the `<input>` tag:

```
  <input type="text" name="first_name" id="first_name" />
  <span>
  <?php
  if ($_POST && isset($errors['first_name'])) {
    echo $errors['first_name'];
  }
  ?>
  </span>
</p>
```

The conditional statement begins by checking `$_POST`. If the form has been submitted, it equates to `TRUE`, so the next test is applied. The `isset()` function checks the existence of a variable. `$errors['first_name']` is created only if the validation test fails, so `$errors['first_name']` is displayed if the form has been submitted and the `first_name` field failed validation.

The `<span>` tags remain empty if there isn't an error, so it might seem more logical to include them inside the conditional statement. They have been left outside to act as a hook for a custom server behavior that you'll create later in this lesson to insert error messages for the other fields.

12 Save add_user.php, and click Live View or press F12/Opt+F12 to test it. Start by leaving the "First name" field blank. Submit the form, and remember to hold down the Ctrl/Cmd key if you're in Live View. If all your code is OK, you should see an error message next to the "First name" field.


Sign Up Now

Just a few details and you're in

All fields are required

First name: Required field, no numbers

13 Now try typing your own name and resubmitting the form. The error message disappears.

14 Type some numbers and resubmit. The error message reappears.

15 Click inside the field, and press the spacebar several times before resubmitting the form. The error message disappears. You still have the problem of an empty field.

Unfortunately, the `NotEmpty` validation class doesn't have an option to handle this. Also, personal names sometimes include a hyphen or apostrophe. The best solution is to use a *regular expression*—a pattern for matching text.

16 Turn off Live View, if necessary, and switch back to user_registration.php. Change the validator from `Alpha` to `Regex` like this:

```
$val = new Zend_Validate_Regex('/^[a-z]+[-\'a-z ]+$/i');
```

This regular expression—or regex, for short—makes sure the value begins with at least one letter and is followed by at least one more letter, hyphen, apostrophe, or space.

This is fine for English. If you need to accept accented letters or names written in a different script, such as Japanese or Chinese, use the following:

```
$val = new Zend_Validate_Regex('/^\p{L}+[-\'\p{L} ]+$/u');
```

This line performs the same task, but also accepts Unicode letter characters.

TIP: Regular expressions are used widely for matching text in PHP and other programming languages. Learn how to build your own regexes by following my tutorial series in the Adobe Developer Connection at www.adobe.com/devnet/dreamweaver/articles/regular_expressions_pt1.html.

17 Save user_registration.php, and test the "First name" field again. It now accepts names with spaces, hyphens, and apostrophes but rejects numbers and values that don't begin with a letter.

The second regex accepts names like Françoise, Дмитрий, and 太郎.

You can compare your code with lesson07/completed/add_user01.php and lesson07/completed/scripts/user_registration01.php.

Building the validation script (2)

The rest of the script follows a similar pattern: You need a validator for each input field and need to add a message to the `$errors` array if the value fails the test. Sometimes a validator can be reused, but if it's no longer appropriate, you can overwrite it by assigning a new one to the same variable.

1 The `surname` input field can use the same validator as `first_name`, so add the following code immediately after the `first_name` test in user_registration.php:

```
  if (!$val->isValid($_POST['first_name'])) {
    $errors['first_name'] = 'Required field, no numbers';
  }
  if (!$val->isValid($_POST['surname'])) {
    $errors['surname'] = 'Required field, no numbers';
  }
} catch (Exception $e) {
```

2 The next input field to validate is `username`. A username should consist of letters and numbers only, and should be 6–15 characters long. This requires two tests, so you need to create a `Zend_Validate` object, and chain the validators. You don't need the existing validator, so you can overwrite it.

Add the following code immediately after the code you entered in the previous step (and still inside the `try` block):

```
$val = new Zend_Validate();
$length = new Zend_Validate_StringLength(6,15);
$val->addValidator($length);
$val->addValidator(new Zend_Validate_Alnum());
if (!$val->isValid($_POST['username'])) {
  $errors['username'] = 'Use 6-15 letters or numbers only';
}
```

This starts by creating a generic `Zend_Validate` object ready for chaining. Next, a `StringLength` validator—with a minimum of 6 characters and a maximum of 15—is created and assigned to `$length`.

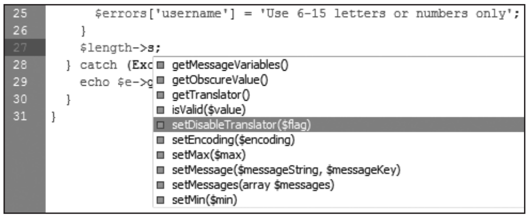
In the third line, the `addValidator()` method chains `$length` to `$val`.

Then the `Alnum` validator is chained to it. By not passing an argument to `Alnum`, no whitespaces are allowed.

Why use different ways of chaining the validators? Surely the `StringLength` validator could have been passed directly as an argument to `addValidator()` in the same way as `Alnum`, right? It could, but the password field needs to be a minimum of 8 characters.

Assigning the `StringLength` validator to its own variable lets you change the minimum value ready for reuse.

3 On the line immediately after the code you just inserted, type **$length->**. As soon as you type the -> operator, Dreamweaver code hints display the methods available to a `StringLength` validator. Type **s**. The code hints display a number of methods that begin with "set."



```
      $errors['username'] = 'Use 6-15 letters or numbers only';
    }
    $length->s;
  } catch (Exc
    echo $e->g
  }
}
```

4 Use your arrow keys to scroll down to `setMin($min)` and press Enter/Return or double-click. Set the value to **8**, and type a closing parenthesis and semicolon. The finished line should look like this:

```
$length->setMin(8);
```

This resets the minimum number of characters required by the `StringLength` validator to 8. The maximum remains unchanged at 15.

TIP: Many ZF classes have methods that begin with "get" and "set" to find out or change the values of an object's properties.

5 Now that you have changed the minimum required by the `StringLength` validator, you can create the validation test for the `password` input field. It's almost exactly the same as for `username`. Add this immediately after the line you just entered:

```
$val = new Zend_Validate();
$val->addValidator($length);
$val->addValidator(new Zend_Validate_Alnum());
if (!$val->isValid($_POST['password'])) {
  $errors['username'] = 'Use 8-15 letters or numbers only';
}
```

This allows any combination of letters and numbers. For a more robust password, use the `Regex` validator. There's a regex for a strong password that requires a mixture of uppercase and lowercase characters and numbers at http://imar.spaanjaars.com/QuickDocId.aspx?quickdoc=297. In the comments on the same page, there's an even stronger one that requires at least one special character.

6 To check that both password values are identical, use the `Identical` validator. This code goes immediately after the code in the preceding step:

```
$val = new Zend_Validate_Identical($_POST['password']);
if (!$val->isValid($_POST['conf_password'])) {
  $errors['conf_password'] = "Passwords don't match";
}
```

You want to check that `$_POST['conf_password']` is identical to `$_POST['password']`, so `$_POST['password']` is passed as the argument to the `Identical` validator.

CAUTION! The error message contains an apostrophe, so the string needs to be enclosed in double quotation marks.

7 The final test is for the email. Add this after code in the previous step:

```
$val = new Zend_Validate_EmailAddress();
if (!$val->isValid($_POST['email'])) {
  $errors['email'] = 'Use a valid email address';
}
```

8 Save user_registration.php. You'll come back to it later to add the code that inserts the user's details in the database. If you want to check your code so far, compare it with lesson07/completed/scripts/user_registration02.php.

Preserving input when validation fails

Ever submitted an online form only to be told there's an error, and all your input has been wiped out? Not much fun is it? Good validation and form design preserves the user's input if there's an error. It's quite simple to do. The input is stored in the `$_POST` or `$_GET` array, depending on the method used to submit the form (most of the time, it's `POST`). All that's necessary is to assign the appropriate element of the `$_POST` or `$_GET` array to the `value` attribute of the input field.

1 In add_user.php, locate the first `<input>` tag in Code view. It looks like this:

```
<input type="text" name="first_name" id="first_name" />
```

2 The `<input>` tag doesn't have a `value` attribute, so you need to add one, and use PHP to assign its content like this:

```
<input type="text" name="first_name" id="first_name"
value="<?php if ($_POST && $errors) {
  echo htmlentities($_POST['first_name'], ENT_COMPAT, 'UTF-8');
}?>" />
```

The PHP code block inside the quotation marks of the `value` attribute is controlled by a conditional statement that checks the `$_POST` and `$errors` arrays. The `$_POST` array is empty unless a form has been submitted, so the code inside the curly braces is ignored when the page first loads.

The `$errors` array is declared at the top of user_registration.php, so it always exists, but elements get added to it only if the validation script finds any problems with the user input. Consequently, `$errors` will equate to `TRUE` only if the form has been submitted and at least one error has been found.

If both tests equate to `TRUE`, the code passes `$_POST['first_name']` to a function called `htmlentities()` and uses `echo` to display the result. You could use `echo` on its own, but displaying raw user input in a web page is a security risk. The `htmlentities()` function sanitizes the input by converting characters that have a special meaning in HTML into HTML entities. For example, the angle brackets in `<script>` are converted to `&lt;` and `&gt;`, which prevent an unauthorized script from running in your web page.

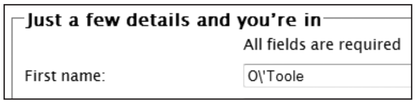
Often, `htmlentities()` takes just one argument—the string you want to convert. The second and third arguments have been added because `htmlentities()` uses Latin1 (iso-8859-1) as its default encoding. `ENT_COMPAT` is a constant that specifies preserving single quotation marks and converting only double ones. The third argument specifies UTF-8 (Unicode) as the encoding. This preserves accented or nonalphabetic characters.

3 Save add_user.php, type anything in the "First name" field, and submit the form. The validation script detects errors in the other fields, so `$_POST` and `$errors` are no longer empty arrays. The value you entered is redisplayed.

You can compare your code with lesson07/completed/add_user02.php.

Dealing with unwanted backslashes

In add_user.php, test the "First name" field with a name that contains an apostrophe, such as **O'Toole**. If magic quotes (see Lesson 2) are turned on, the apostrophe is preceded by a backslash like this when it's redisplayed:


Just a few details and you're in

All fields are required

First name: O\'Toole

If you see a backslash in front of the apostrophe, add the following code after the `catch` block at the bottom of scripts/library.php (you can copy and paste it from lesson07/completed/scripts/library_magic_quotes.php):

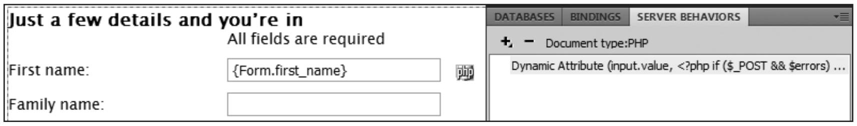
```
if (get_magic_quotes_gpc()) {
  $process = array(&$_GET, &$_POST, &$_COOKIE);
  while (list($key, $val) = each($process)) {
    foreach ($val as $k => $v) {
      unset($process[$key][$k]);
      if (is_array($v)) {
        $process[$key][stripslashes($k)] = $v;
        $process[] = &$process[$key][stripslashes($k)];
      } else {
        $process[$key][stripslashes($k)] = stripslashes($v);
      }
    }
  }
  unset($process);
}
```

When you test the page again, the backslash should have been removed.

NOTE: This script comes from the PHP manual at http://docs.php.net/manual/en/security.magicquotes.disabling.php and is the least efficient way of dealing with magic quotes. You need to add the script to library.php only if there is a backslash in front of the apostrophe and you cannot use any other method of disabling magic quotes (see "Deciding to use or not to use 'magic quotes'" in Lesson 2).

Creating your own server behaviors

Now you need to fix the redisplay of user input and error messages for the remaining input fields. Doing it all by hand is tedious. But take a look at what's happened in add_user.php. In Design view, Dreamweaver has added what looks like a dynamic text object in the "First name" input field. The Server Behaviors panel also lists a Dynamic Attribute.

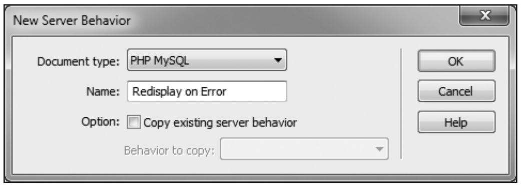

Just a few details and you're in
All fields are required
First name: {Form.first_name}
Family name:

Dynamic Attribute (input.value, <?php if ($_POST && $errors) ...

Does this mean Dreamweaver has a server behavior that you can you use here? No, but it does have the Server Behavior Builder, which lets you create your own.

1 With add_user.php open in the Document window, open the Server Behaviors panel by clicking its tab or choosing Window > Server Behaviors (Ctrl+F9/Cmd+F9). Click the plus button at the top left of the panel, and choose New Server Behavior.

2 In the New Server Behavior dialog box, make sure "Document type" is set to PHP MySQL, and type **Redisplay on Error** in the Name field. Leave the checkbox deselected, and click OK.

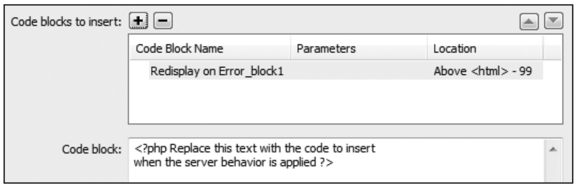


Document type: PHP MySQL
Name: Redisplay on Error
Option: Copy existing server behavior
Behavior to copy:

NOTE: Selecting the "Copy existing server behavior" checkbox lets you choose a server behavior on which to base a new one. However, you can do this only with server behaviors you have created yourself. You cannot use one of Dreamweaver's built-in server behaviors.

This opens a large dialog box where you define the new server behavior.

3 Click the plus button labeled "Code blocks to insert" to open the Create a New Code Block dialog box. Accept the name suggested by Dreamweaver, and click OK. This lists the code block in the top pane and inserts some placeholder text in the "Code block" section.



Code Block Name	Parameters	Location
Redisplay on Error_block1		Above <html> - 99

Code block:

```
<?php Replace this text with the code to insert
when the server behavior is applied ?>
```

4 Replace the placeholder text with the PHP code that you added to the `<input>` tag's `value` attribute in step 2 of "Preserving input when validation fails." Here it is again:

```
<?php if ($_POST && $errors) {
  echo htmlentities($_POST['first_name'], ENT_COMPAT, 'UTF-8');
}?>
```

5 If you leave the code like this, the server behavior would always use `$_POST['first_name']`. To make it editable, you need to replace `first_name` with a parameter.

Delete `first_name` but not the surrounding quotation marks. Make sure your insertion point is between the quotation marks of `$_POST['']`, and click the Insert Parameter in Code Block button.

6 In the "Parameter name" field, type **Field Name**, and click OK. The code in the "Code block" section should now look like this:

```
<?php if ($_POST && $errors) {
  echo htmlentities($_POST['@@Field Name@@'], ENT_COMPAT, 'UTF-8');
}?>
```

The `@@` surrounding the parameter name tell Dreamweaver to replace the value when you use the server behavior.

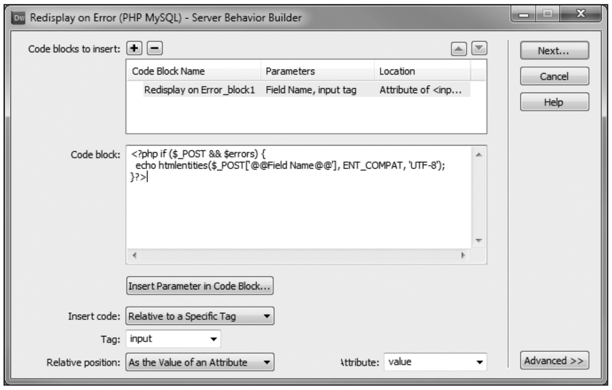
7 You now need to tell the server behavior where to insert the code when you use it. You want to use it in the `value` attribute of an `<input>` tag.

Set "Insert code" to Relative to a Specific Tag. An option called Tag appears.

8 Select "input" from the Tag menu.

9 Set "Relative position" to "As the Value of an Attribute." This opens up yet another option labeled Attribute.

10 Select "value" from the Attribute menu. The settings in the Server Behavior Builder dialog box should now look like this:


Redisplay on Error (PHP MySQL) - Server Behavior Builder

Code blocks to insert:

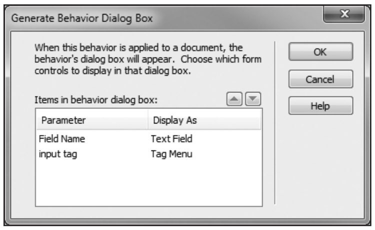
Code Block Name	Parameters	Location
Redisplay on Error_block1	Field Name, input tag	Attribute of <inp...

Code block:

```
<?php if ($_POST && $errors) {
  echo htmlentities($_POST['@@Field Name@@'], ENT_COMPAT, 'UTF-8');
}?>
```

Insert code: Relative to a Specific Tag
Tag: input
Relative position: As the Value of an Attribute
Attribute: value

11 Click Next at the top right of the dialog box to open the dialog box that defines the options that will appear in the new server behavior's dialog box.


Generate Behavior Dialog Box

When this behavior is applied to a document, the behavior's dialog box will appear. Choose which form controls to display in that dialog box.

Items in behavior dialog box:

Parameter	Display As
Field Name	Text Field
input tag	Tag Menu

12 The values suggested by Dreamweaver are fine. Just click OK to complete the creation of the Redisplay on Error server behavior.

13 Create another custom server behavior to display the error messages. The process is the same, so the following instructions provide only the main points.

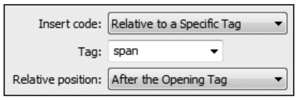
Click the plus button in the Server Behaviors panel, and choose New Server Behavior. Call it **Display Error Message**.

14 Create a new code block, and accept the default name. Replace the placeholder text with the following code:

```
<?php if ($_POST && isset($errors['@@Field@@'])) {
  echo $errors['@@Field@@'];
} ?>
```

This is the same as the code used to display the error message next to the "First name" field except both instances of `first_name` have been replaced by the parameter `@@Field@@`.

15 The error message is displayed inside a `<span>` tag, so use the following settings at the bottom of the panel:


Insert code: Relative to a Specific Tag
Tag: span
Relative position: After the Opening Tag

16 Click Next, accept the default settings in the next dialog box, and click OK.

Finishing the registration form

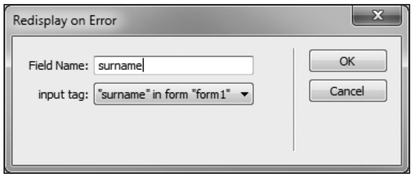
The custom server behaviors make it easy to preserve user input and display error messages. As long as you use an array called `$errors` to store error messages, they can be used in any page, not just this one.

TIP: You could make the server behaviors more flexible by using a parameter for the `$errors` array. However, consistency in coding is a virtue, and using a fixed variable keeps the code simple.

1 With add_user.php open in Design view, select the "Family name" field in the registration form.

2 Click the plus button in the Server Behaviors panel, and choose "Redisplay on Error" to open the dialog box for your new server behavior.

3 Type **surname** in the Field Name field. The correct value should already be selected for "input tag."



Field Name: surname
input tag: "surname" in form "form1"

4 Click OK. Dreamweaver inserts a dynamic text object in the field. It also lists "Redisplay on Error" in the Server Behaviors panel. Dreamweaver should also recognize the code in the "First name" field and list that as a "Redisplay on Error" server behavior.

NOTE: Don't worry if the hand-coded version isn't recognized as a "Redisplay on Error" server behavior. Differences in spacing and new lines could affect Dreamweaver's ability to recognize it as the same.

5 Repeat steps 1–4 with the Username and "Email address" fields, typing **username** and **email** respectively in Field Name.

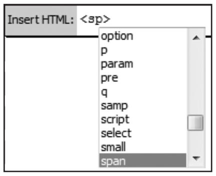
6 You can't apply the server behavior to the password fields. There's no point in doing so anyway, because password fields don't display user input.

7 Save add_user.php, and test the new server behavior by typing values in all fields except the password fields and submitting the form. The lack of passwords causes validation to fail and redisplays the values you entered.

✱ **NOTE:** If your page doesn't work as expected, compare your code with add_user03.php in lesson07/completed. The most likely problem is a mistake in the "Code block" section of the Server Behavior Builder or in the parameter. You can edit a custom server behavior by clicking the plus button in the Server Behaviors panel and choosing Edit Server Behavior. To rebuild a server behavior from scratch, select its name in the Edit Server Behavior dialog box, and click Remove.

8 The error messages are displayed in `<span>` tags. In Design view, position the insertion point immediately to the right of the "Family name" input field, right-click, and choose Insert HTML.

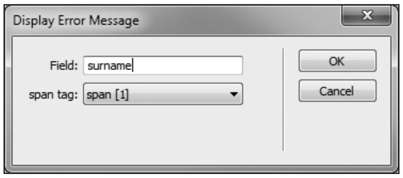
This displays a small window for you to add an HTML tag. Type **sp** to highlight `span`, and press Enter/Return to insert an empty pair of `<span>` tags.



9 Open Split view to make sure the insertion point is between the opening and closing `<span>` tags. If it isn't, move it there by clicking between them.

10 Click the plus button in the Server Behaviors panel, and choose Display Error Message to open the dialog box for the other new server behavior.

11 Type **surname** in Field. The value of "span tag" should be "span [1]." The span doesn't have an `id` attribute, so Dreamweaver uses an array to identify it, counting from zero. So, this is the second `<span>` in the page.

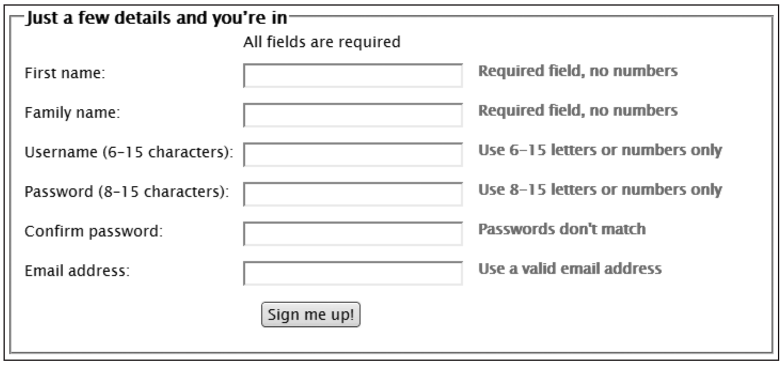

Display Error Message

Field: surname

span tag: span [1]

12 Repeat steps 8–11 for the remaining input fields. As long as the insertion point is between the opening and closing `<span>` tags, the Display Error Message dialog box selects the correct value for "span tag." The value you type in Field should match the `name` attribute of each input field, namely: **username**, **password**, **conf_password**, and **email**.

13 Save add_user.php, and test the page by typing a few letters in the “Confirm password” field only. When you submit the form, confirm that error messages are displayed next to each field.



Just a few details and you're in

All fields are required

First name:		Required field, no numbers
Family name:		Required field, no numbers
Username (6-15 characters):		Use 6-15 letters or numbers only
Password (8-15 characters):		Use 8-15 letters or numbers only
Confirm password:		Passwords don't match
Email address:		Use a valid email address

This completes add_user.php. Compare your code with lesson07/completed/add_user.php if the page doesn’t work as expected.

Using a variable in a SELECT query

The final part of the form input that needs to be validated before you can register the user in the database is the username. At the beginning of this lesson, you added a unique index to the `username` column, so the database rejects any attempt to insert the same value more than once. To avoid this, you need to check the database and display an error message if the username already exists.

`Zend_Db` offers several different ways of querying a database. In this case, the simplest way is to write your own SQL query. All you’re interested in is whether the username exists in the database, so selecting `user_id` is sufficient. To find the `user_id` for the username “hitchhiker,” the basic SQL query looks like this:

```
SELECT user_id FROM users WHERE username = 'hitchhiker'
```

SQL is designed to emulate human language, so the meaning should be obvious. By convention, the SQL commands are written in uppercase. The names of columns and tables are not enclosed in quotation marks, but string values are. So, this selects the values in the `user_id` column from the `users` table, where the value in the `username` column equals “hitchhiker.”

For this validation script, you want to match the value that comes from `$_POST['username']`, but you have no idea what this contains. It could be an attempt to hack into your database.

So, you need to sanitize the value that's inserted into the SQL. One way of doing this with `Zend_Db` is to use the `quoteInto()` method, which takes two arguments:

- The first argument is the SQL statement with a question mark used as a placeholder for the variable containing the user input.
- The second argument is the variable you want to use.

This sanitizes the value and wraps it in the necessary quotation marks.

To execute a `SELECT` query, you pass the SQL query to the `fetchAll()` method, which returns an array containing all the results.

Checking for duplicate usernames

Now that you know the basics of querying the database, you can fix the validation script so that it checks for a duplicate username.

1 The section in user_registration.php that validates the length and characters in the username ends with the following conditional statement (it should be around lines 19–21):

```
if (!$val->isValid($_POST['username'])) {
  $errors['username'] = 'Use 6-15 letters or numbers only';
}
```

If the username fails this validation test, there's no need to check the database; if the username passes, you need to make sure it's not a duplicate. This is an either/or situation, so it calls for an `else` block to be added to the original conditional statement like this:

```
if (!$val->isValid($_POST['username'])) {
  $errors['username'] = 'Use 6-15 letters or numbers only';
} else {
  // check the database for duplicate username
}
```

All the code in the following steps goes inside the `else` block.

2 Use the `quoteInto()` method to build the SQL statement with a question mark placeholder, and pass the variable to it as the second argument:

```
$sql = $dbRead->quoteInto('SELECT user_id FROM users WHERE username = ?',
➥ $_POST['username']);
```

`$dbRead` is one of the `Zend_Db` objects you created in library.php earlier in this lesson to connect to the database.

3 Execute the query by passing `$sql` to `fetchAll()`, and capture the result like this:

```
$result = $dbRead->fetchAll($sql);
```

4 The `fetchAll()` method returns an array of results. PHP treats an array that contains any elements as `TRUE`. If a match is found, the username is a duplicate, so you can create an error message like this:

```
if ($result) {
  $errors['username'] = $_POST['username'] . ' is already in use';
}
```

If no match is found, `$result` is empty, which PHP treats as `FALSE`, so the code inside the braces is ignored.

5 Save user_registration.php, and test add_user.php by entering a username that already exists in the database. If your database table is empty, you can add a record by selecting the `users` table in phpMyAdmin and clicking the Insert tab at the top of the page. When you submit the form with a duplicate username, you'll see the error message.

You can compare your code with lesson07/completed/scripts/user_registration03.php.

Inserting the user details with Zend_Db

Inserting a record in a database with `Zend_Db` is very easy. You don't need any SQL. Just create an associative array using the column names as the keys, and pass the table name and array as arguments to the `insert()` method.

You can now finish the user registration process:

1 If the user input passes all the validation tests, the `$errors` array is empty. You can use this to control whether to insert the data in the `users` table. Add this conditional statement at the end of the validation sequence, just above the `catch` block:

```
  if (!$val->isValid($_POST['email'])) {
    $errors['email'] = 'Use a valid email address';
  }
  if (!$errors) {
    // insert the details in the database
  }
} catch (Exception $e) {
```

PHP implicitly treats an empty array as `FALSE`. The logical Not operator (an exclamation point) reverses a Boolean value, so this effectively means "if there are *no* errors." If the `$errors` array is empty, the condition equates to `TRUE`, and the details will be inserted into the database.

All the remaining code goes inside this conditional statement.

2 Create an associative array of the column names and values. You don't need an array element for `user_id`, because MySQL automatically inserts the next available number for the primary key. The array should look like this:

```
$data = array('first_name'  => $_POST['first_name'],
              'family_name' => $_POST['surname'],
              'username'    => $_POST['username'],
              'password'    => sha1($_POST['password']));
```

`Zend_Db` automatically sanitizes the values before inserting them into the database. The password is the only value that receives special treatment. It's passed to the `sha1()` function for encryption.

3 To insert the data, you need to use the `Zend_Db` object with read/write privileges. Add this line after the array you have just created:

```
$dbWrite->insert('users', $data);
```

The first argument to the `insert()` method is a string containing the name of the table you want to use, and the second argument is the data array.

4 That's all there is to inserting the data. After the data is inserted, redirect the user to the login page with the `header()` function like this:

```
header('Location: login.php');
```

NOTE: Strictly speaking, you should use a full URL when redirecting a page, but all current browsers accept a relative URL. Using this shortcut is acceptable for local testing, but you should replace this with a fully qualified URL when deploying your pages on a live site. See http://docs.php.net/manual/en/function.header.php.

5 Save user_registration.php, and copy lesson07/start/login.php to your lesson07/workfiles folder.

6 Test add_user.php to enter a new user in the database. If there are no errors, you will be taken to login.php. You can compare your code with lesson07/completed/scripts/user_registration.php.

Authenticating User Credentials with Zend_Auth

The end result of the improved registration form that you just created is the same as Dreamweaver's built-in Insert Record server behavior—it inserts user details in the database. Dreamweaver's User Authentication server behaviors can still be used to control access to pages. However, ZF makes it easy to create scripts to authenticate user credentials. Using external files for the scripts that control access and log out users simplifies their inclusion in any page.

Controlling user access with Zend_Auth

The `Zend_Auth` class creates a *singleton* object, which means that only one instance can exist at a time. In practical terms, the only difference is that you use `getInstance()` instead of the `new` keyword like this:

```
$auth = Zend_Auth::getInstance();
```

`Zend_Auth` is capable of using a variety of authentication methods. When the user credentials are stored in a database, authentication is done by an adapter called `Zend_Auth_Adapter_DbTable`, which takes the following five arguments:

- A `Zend_Db` object that connects to the database
- The name of the table that contains the user credentials
- The name of the column that contains the username
- The name of the password column
- The encryption, if any, to be applied to the password

The fifth argument uses a question mark as a placeholder for the password. The `users` table uses `sha1()` encryption, so you create an adapter like this:

```
$adapter = new Zend_Auth_Adapter_DbTable($dbRead, 'users', 'username',
➥ 'password', 'sha1(?)');
```

When a user logs in, pass the username and password to the adapter like this:

```
$adapter->setIdentity($_POST['username']);
$adapter->setCredential($_POST['password']);
```

To check whether the user's credentials are valid, pass the adapter to the `Zend_Auth` object's `authenticate()` method:

```
$result = $auth->authenticate($adapter);
```

Finally, if the login succeeds, store the user's details like this:

```
if ($result->isValid()) {
  $storage = $auth->getStorage();
  $storage->write($adapter->getResultRowObject(
    array('username', 'first_name', 'family_name')));
}
```

The array passed as an argument to the `getResultRowObject()` method is a list of the database fields you want to store. Dreamweaver's built-in Log In User server behavior stores only the username and access level, if any. `Zend_Auth` gives you access to as much information about the user as you want—provided, of course, it's stored in the database.

NOTE: The `getStorage()` and `write()` methods store the user's details as a `Zend_Auth_Storage_Session` object, which is essentially just another way of storing session variables.

You restrict access to pages by using the `hasIdentity()` method, which returns `TRUE` if the user has logged in.

To access details of a logged-in user, use the `getIdentity()` method like this:

```
$auth = Zend_Auth::getInstance();
if ($auth->hasIdentity()) {
  $identity = $auth->getIdentity();
}
```

The details can then be accessed as properties of `$identity`, for example:

```
$identity->first_name
```

To log out a visitor, use the `clearIdentity()` method.

Creating the login script

The Log In User server behavior that you used in Lesson 6 sends the user to another page on failure or displays the login form again without explanation. This time you'll display an error message so the user knows what's happened.

1 Create a new PHP page, and save it as **user_authentication.php** in lesson07/workfiles/scripts. The page will contain only PHP code, so delete the HTML inserted by Dreamweaver, and add an opening PHP tag.

2 To control the error message, create a `FALSE` Boolean variable:

```
$failed = FALSE;
```

If the login fails, this will be reset to `TRUE` and display the message.

3 The login script should run only if the form is submitted. Add a conditional statement to include library.php if the `$_POST` array equates to `TRUE`:

```
if ($_POST) {
  require_once('library.php');
  // check the user's credentials
}
```

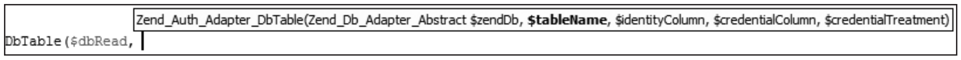
4 The script uses objects, so add `try` and `catch` blocks inside the conditional statement, and get an instance of `Zend_Auth` in the `try` block.

```
try {
  $auth = Zend_Auth::getInstance();
} catch (Exception $e) {
  echo $e->getMessage();
}
```

TIP: A quick way to select the code hint for Zend_Auth is to press Ctrl+spacebar, and type **dau**. This selects `Zend_Auth` immediately. Press Enter/Return to insert it, and then type two colons to bring up a code hint for `getInstance()`.

The rest of the script goes inside the `try` block.

5 Create a `Zend_Auth_Adapter_DbTable` object. A quick way to select its code hint is by typing **hada** after the `new` keyword. This takes five arguments, but Dreamweaver helps you by highlighting the current argument in bold.



```
Zend_Auth_Adapter_DbTable(Zend_Db_Adapter_Abstract $zendDb, $tableName, $identityColumn, $credentialColumn, $credentialTreatment)
DbTable($dbRead, 
```

The code for this line should look like this:

```
$adapter = new Zend_Auth_Adapter_DbTable($dbRead, 'users', 'username',
➥ 'password', 'sha1(?)');
```

6 Set the identity and credential values for the adapter. They come from the `username` and `password` values in the `$_POST` array:

```
$adapter->setIdentity($_POST['username']);
$adapter->setCredential($_POST['password']);
```

7 Pass the adapter to the `authenticate()` method of the `Zend_Auth` object, and save the result in a variable:

```
$result = $auth->authenticate($adapter);
```

8 Use the `isValid()` method to determine whether the user's credentials are correct. If they are, store the details, and use `header()` to redirect the user to members_only.php. If the login attempt fails, reset `$failed` to `TRUE`. The code looks like this:

```
if ($result->isValid()) {
  $storage = $auth->getStorage();
  $storage->write($adapter->getResultRowObject(array(
    'username', 'first_name', 'family_name')));
  header('Location: members_only.php');
```

```
    exit;
  } else {
    $failed = TRUE;
  }
```

CAUTION! Relative links in include files are treated as relative to the page that includes them, not to the location of the include file. Although this script is in the scripts folder, it will be included in login.php, which is in the same folder as members_only.php. So, the relative link that redirects the user on success needs to be relative to login.php, not to user_authentication.php.

Testing the login script

To use the login script, include it above the `DOCTYPE` declaration of the login page, and add a conditional statement to show an error message if the login fails.

1 Copy login.php and members_only.php from lesson07/start to lesson07/workfiles.

2 Include user_authentication.php in login.php by adding the following above the `DOCTYPE` declaration:

```
<?php
require_once('scripts/user_authentication.php');
?>
```

3 In Design view, insert a new paragraph after the Members Only heading, and type **Login failed. Check username and password.**

With the insertion point anywhere in the paragraph, choose `warning` from the Class menu in the Property inspector. This turns the text bold red.

4 Switch to Code view, and wrap the paragraph you just created in a conditional statement like this:

```
<?php if ($failed) { ?>
<p class="warning">Login failed. Please check username and password.</p>
<?php } ?>
```

The script in user_authentication.php sets `$failed` to `FALSE` and changes it to `TRUE` only if a login attempt fails, so this prevents the error message from being shown unless the user's credentials are rejected.

TIP: Some developers feel obliged to convert HTML in conditional statements to strings and use `echo` to display it. It's not necessary, and makes the code harder to write.

5 Save login.php, and test it in Live View or a browser. When the page first loads, the error message is not displayed. Try to log in using an invalid username or password. The error message is displayed.

6 Click “Sign in” without typing anything in either field. You get the following message: “A value for the identity was not provided prior to authentication with Zend_Auth_Adapter_DbTable.”

This comes from the `catch` block. Failing to enter anything in the Username field has caused `Zend_Auth` to throw an exception. This is more elegant than the lengthy error message you get without `try` and `catch`, but it’s not something you want to show visitors to your site.

7 Switch back to user_authentication.php, and add another conditional statement inside the existing one near the top of the script:

```
if ($_POST) {
  if (empty($_POST['username']) || empty($_POST['password'])) {
    $failed = TRUE;
  } else {
    require_once('library.php');
```

This uses `empty()` to test if `$_POST['username']` or `$_POST['password']` contains a value. If either is empty, `$failed` is reset to `TRUE`. The authentication script is now inside an `else` block that is executed only if `$_POST['username']` and `$_POST['password']` both have values.

8 A red marker on the last line of the script reminds you that you need to add a closing curly brace to match the opening one of the `else` block. Technically speaking, the missing brace goes between the last two braces at the bottom of the script, but you can put it on the final line, and the red marker goes away.

9 Save user_authentication.php, and test the login form again without filling in the fields. This time you get your custom error message in red.

10 Test the login form with a registered username and password. When you click “Sign in,” you are taken to members_only.php.

You can compare your code with lesson07/completed/login.php and lesson07/completed/scripts/user_authentication01.php.

Password-protecting pages

Building the login script was the complex part. To restrict access to a page, you use the `hasIdentity()` method, which returns `TRUE` or `FALSE`. If the user is logged in, you retrieve his or her details with `getIdentity()`. Otherwise, you redirect the user to the login page.

1 Create a PHP page, and save it as **restrict_access.php** in lesson07/workfiles/scripts. Delete the HTML inserted by Dreamweaver.

2 The script is so short; here it is in its entirety:

```
<?php
require_once('library.php');
try {
  $auth = Zend_Auth::getInstance();
  if ($auth->hasIdentity()) {
    $identity = $auth->getIdentity();
  } else {
    header('Location: login.php');
    exit;
  }
} catch (Exception $e) {
  echo $e->getMessage();
}
```

This includes library.php and creates an instance of `Zend_Auth`. The conditional statement checks whether the user is logged in. If it succeeds, the user's details are stored in `$identity`. Otherwise, the page is redirected to login.php, and `exit` ensures that the script comes to an immediate halt, preventing the protected page from being displayed.

3 To password-protect members_only.php, just include this script above the `DOCTYPE` declaration like this:

```
<?php
require_once('scripts/restrict_access.php');
?>
```

Assuming you're already logged in, you need to log out before you can test this. For that, you need a logout button and script, which is coming up next.

Creating a logout system

`Zend_Auth` makes logging out easy with the `clearIdentity()` method. You can add the necessary code to user_authentication.php, and use a link to login.php to log out the user. A query string at the end of the link triggers the logout.

1 Create a new paragraph in members_only.php, and type **Log Out.**

2 Select the text, and link to login.php. Add **?logout** to the end of the link. You can do this in the Link field of the Property inspector or in Code view. Save members_only.php.

3 Switch to user_authentication.php, and add the following code at the bottom of the page:

```
if (isset($_GET['logout'])) {
  require_once('library.php');
  try {
    $auth = Zend_Auth::getInstance();
    $auth->clearIdentity();
  } catch (Exception $e) {
      echo $e->getMessage();
  }
}
```

Values passed through a query string are in the `$_GET` array. This uses `isset()` to see if `$_GET['logout']` exists. If it does, the ZF files are included, an instance of `Zend_Auth` is created, and the `clearIdentity()` method performs the logout. That's all there is to it. The session variables storing the user's details are deleted automatically.

NOTE: If you're wondering why the script doesn't use the earlier instance of `Zend_Auth`, it's created only when the login form is submitted. You could shorten the code slightly by including the ZF files and creating the `Zend_Auth` instance outside the conditional statements. However, neither is needed when the login form first loads, so this is more efficient, albeit at the expense of a few extra lines of code.

4 Save user_authentication.php, and load members_only.php in a browser or Live View. If the page displays, click the "Log out" link. You will be taken to the login form.

5 Log in, and click the "Log out" link again.

6 Now try to access members_only.php directly. You will be denied access and taken to the login form.

Displaying a logged-in user's details

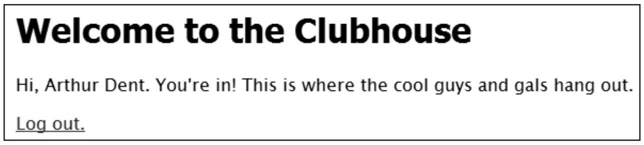
The code in restrict_access.php calls the `getIdentity()` method and stores the user's username, first name, and family name as properties of `$identity`. This makes it possible to greet a user by name after logging in.

1 In Code view, amend the beginning of the first paragraph in members_only.php like this:

```
<p>Hi, <?php echo "$identity->first_name $identity->family_name"; ?>.
➥ You're in!
```

The column names are treated as properties of `$identity`, allowing you to access the information they contain about the user with the -> operator.

2 Save members_only.php, and log back in to be greeted by name.



Welcome to the Clubhouse

Hi, Arthur Dent. You're in! This is where the cool guys and gals hang out.

Log out.

What You Have Learned

In this lesson, you have:

- Installed the Zend Framework (pages 214–215)
- Set up ZF site-specific code hints (pages 215–218)
- Altered a database table to add a unique index and extra columns (pages 219–220)
- Created a library file to load ZF classes and connect to the database (pages 221–227)
- Validated user input on the server with Zend_Validate (pages 227–236)
- Redisplayed user input when errors are detected in server-side validation (pages 236–237)
- Removed unwanted backslashes inserted by "magic quotes" (pages 237–238)
- Used Dreamweaver's Server Behavior Builder (pages 238–241)
- Displayed error messages when input fails validation (pages 242–244)
- Used a variable in a SELECT query to check for duplicate usernames (pages 244–246)
- Inserted user input into a database with Zend_Db (pages 246–247)
- Created a login system with Zend_Auth (pages 247–252)
- Password-protected pages with Zend_Auth (page 253)
- Created a logout system (pages 253–254)
- Displayed a logged-in user's details (pages 254–255)