PHP

By W. Jason Gilmore

ABOUT THIS REFCARD

PHP is the world's most popular server-side Web scripting language, sporting a syntax simple enough to attract novice programmers yet powerful enough to run some of the world's most popular websites, among them Yahoo!, Facebook, GameSpy, and Vimeo.

This reference card was created to help you quickly navigate some of PHP's most commonplace features, including object-oriented programming, array and string manipulation, regular expressions, and MySQL integration.

CONFIGURATION

PHP's behavior can be configured at a variety of levels:

Global Configuration

The php.ini file is PHP's configuration file, containing more than 200 directives capable of tweaking nearly every aspect of the language's behavior. This file is parsed every time PHP is invoked, which for the server module version occurs only when the web server starts, and every time for the CGI version.

Host- and Directory-specific Configuration

If you lack access to the php.ini file, you may be able to change desired directives within Apache's httpd.conf or .htaccess files. For instance, to force the display of all PHP errors for solely your development domain (for instance http://dev.wjgilmore.com), add the following to a .htaccess file:

```
php_flag display_errors on
```

Hot Tip Each directive is assigned one of three permission levels (PHP_INI_ALL, PHP_INI_PER_DIR, PHP_INI_SYSTEM) which determines where it can be set. Be sure to consult the PHP documentation before tweaking settings outside of the php.ini file. See http://www.php.net/ini for a complete list of directives.

Script-specific Configuration

Occasionally you'll want to tweak directives on a per-script basis. For instance to change PHP's maximum allowable execution time for a script tasked with uploading large files, you could call the `ini_set()` function from within your PHP script like so:

```
ini_set('max_execution_time', 60);
```

Changing the PHP File Extension

PHP's default file extension is .php, however you can change it to whatever you please by adding the desired extension to the `AddType` directive within Apache's httpd.conf file. For instance to configure Apache to recognize .dzone as a supported PHP file extension:

```
AddType application/x-httpd-php  .php  .dzone
```

POPULAR PEAR PACKAGES

The PHP Extension Application Repository (PEAR) is the de facto service for distributing reusable PHP components. Over 500 packages are available for download from http://pear.php.net/, including these popular solutions:

PEAR Packages	Description
Auth	Facilitates authentication against IMAP, LDAP, plaintext files, most modern databases, RADIUS, and other authentication solutions.
Config	Aids in the management of application configuration data
HTML_QuickForm2	Streamlines the creation, processing, and validation of HTML forms.
HTML_Table	Simplifies the generation of dynamic HTML tables
HTTP_Upload	Assists in the management of files uploaded through an HTML form.
Mail	Facilitates transmission of e-mail through a website by supporting multiple mailer backends (including PHP's native mail() function, Sendmail, and SMTP)
MDB2	A database abstraction layer supporting numerous databases, including MySQL, PostgreSQL, Oracle, and MS SQL.
Net_UserAgent_Detect	Provides information regarding the user's browser and operating system.
PHPDocumentor	Automates the code documentation creation and management process
PHPUnit	Aids in the creation, execution and analysis of application tests
XML_RPC	Supports creation of PHP-driven XML-RPC clients and servers.

POPULAR FRAMEWORKS

Web frameworks help the programmer to embrace best practices, simultaneously decreasing errors and eliminating redundant code. If you haven't yet settled upon a framework, consider checking out one or several of the following popular solutions:

Popular Frameworks, continued

Framework	Source
CakePHP	http://www.cakephp.org/
CodeIgniter	http://www.codeigniter.com/
eZ Components	http://ez.no/ezcomponents
Prado	http://www.pradosoft.com/
symfony	http://www.symfony-project.org/
Zend Framework	http://framework.zend.com/

OBJECT-ORIENTED PHP

Creating a Class

A class defines the behavior and characteristics of an entity you'd like to represent in an application. A sample class follows:

```
class RadioStation {
   private $_id;
   private $_name;
   private $_frequency;
   private $_band;
   private $_audioStream;

   public function setBand($band) {
      $this->_band = $band;
   }

   public function getBand() {
   return $this->_band;
   }
   ...
}
```

Object Instantiation

To create an instance of a class (known as an object), you call the class name like you would a function, preceding it with the new keyword:

```
$wtvn = new RadioStation();
```

Class Constructors

Constructors are useful for performing initialization tasks at class instantiation time, thereby saving you the hassle of calling additional class methods. Constructors are declared using the `__construct()` method, like so:

```
function __construct($id="") {
   // If specific station ID is requested, retrieve it
   from the database
   if (isset($id))
            $this->find($id);
}
```

Class Destructors

Custom class destructors can perform tasks when the object is destroyed. You can create a destructor using the `__destruct()` method:

```
function __destruct() {
   printf("The radio station %s has been destroyed!",
   $this->name);
}
```

Attribute and Method Visibility

PHP supports three levels of attribute and method visibility:

Attribute and Method Visibility	Description
Public	Public attributes and methods can be accessed anywhere
Private	Private attributes and methods are only accessible within the class that defines them
Protected	Protected attributes and methods are available to the class and its subclasses.

Object-Oriented PHP, continued

Class Constants

Class constants are defined with the `const` keyword, and can be referenced through the scope resolution operator (`::`). For instance, to define a constant identifying the RadioStation class' minimum supported PHP version:

```
const MIN_PHP_VER = '5.3';
```

You can then reference it outside the class like so:

```
echo RadioStation::MIN_PHP_VER;
```

Extending Classes

Class hierarchies can be created using the `extends` keyword. For instance, an application tasked with cataloging all major media outlets might first define a `MediaOutlet` class which defines some broad characteristics, and then child classes such as `RadioStation` and `TVStation` would inherit from it:

```
class MediaOutlet {
   protected $owner;
   protected $residentCountry;

   public function setOwner($owner) {
      ...
   }
}
class RadioStation extends MediaOutlet {
   ...
}
```

If you wanted to prevent child classes (in this case, `RadioStation`) from overriding a parent method, prefix it with the `final` keyword. For instance:

```
final public function setOwner($owner) {
   ...
}
```

Class Abstraction

The aforementioned `MediaOutlet` class would be more accurately defined as an *abstract class*, because it would never be explicitly instantiated (instead, one would instantiate derived classes such as `RadioStation`, `TVStation`, `Newspaper`, etc.). Abstract classes are declared using the `abstract` keyword:

```
abstract class MediaOutlet {
   ...
}
```

You can choose to override any methods found within an abstract class, which would then be inherited by its child classes, or alternatively you can declare them as abstract, requiring these methods be defined by any child.

Creating Interfaces

An interface helps developers rigorously enforce application specifications, and is similar to an abstract class, but contains solely the required method signatures. Any class implementing the interface must also implement all defined interface methods.

Interfaces are defined using the `interface` keyword and their names are typically prefixed with a capital `I`:

```
interface IRadioStation {
   public function setBand($band);
   public function getBand();
}

class RadioStation implements IRadioStation {
   ...
}
```

WORKING WITH ARRAYS

Creating an Array

The following four examples all create an array named `$stations` consisting of three elements:

```
$stations = array (
  "WTVN",
  "WBNS",
  "WYTS");
```

```
$stations << "WTVN";
$stations << "WBNS";
$stations << "WYTS";
```

```
$stations - array();
$count = array_push($stations, "WTVN", "WBNS", "WYTS");
```

You can create an array consisting of a character- or numerically-based range using the `range()` function:

```
// $teenListenerDemographic =
// array(13,14,15,16,17,18,19)
$teenListenerDemographic = range(13,19);
```

Retrieving Array Contents

Indexed arrays such as those created so far can be accessed according to their numerical offset (beginning with a zero-based offset). For instance to retrieve the second value in the `$stations` array:

```
$callSignal = $stations[1];
```

Perhaps the most flexible way to enumerate array contents is through the `foreach` statement:

```
foreach($stations AS $station)
   printf("%s<br />", $station);
```

Associative Arrays

Associative arrays give developers the opportunity to assign meaningful context to both the array value and its corresponding key:

```
$stations = array(
   "WTVN" => "610",
   "WBNS" => "1460",
   "WYTS" => "1230 "
);
```

You can then obtain a value (in this case the station/band) by referencing its call signal:

```
// $channel = "610"
$channel = $stations["WTVN"];
```

The foreach statement proves equally useful for navigating associative arrays:

```
foreach($stations AS $key => value)
   printf("%s => %s<br />", $key, $value);
```

Multidimensional Arrays

Multidimensional arrays are useful for representing more complex data structures:

```
$stations = array(
   "AM" =>
   array("WTVN" => "610",
      "WBNS" => "1460",
      "WYTS" => "1230"),
   "FM" =>
   array("WLVQ" => "96.3",
      "WNCI" => "97.9")
);
```

Multidimensional Arrays, continued

Referencing an element isn't unlike the methods used for indexed and associative arrays; it's just a tad more verbose:

```
$channel = $stations["FM"]["WTVN"];
```

Determining Array Size

The number of elements found in an array can be determined using the `count()` function:

```
// Outputs "3 stations are being tracked"
printf("%d stations are being tracked",
count($stations));
```

Sorting Arrays

PHP offers a powerful assortment of functions (more than 70) capable of sorting arrays in a variety of ways. Most of these functions accept an optional parameter which can change the sorting behavior. Four values are supported, including `SORT_REGULAR` for comparing elements without implicit typecasting, `SORT_NUMERIC` for comparing elements numerically, `SORT_STRING` for comparing elements as strings, and `SORT_LOCALE_STRING`, for sorting elements according to the defined locale.

Description	Function
Sort an array while maintaining the key association	bool asort(array &$array [, int $sort_flags])
Reverse sort an associative array while maintaining key association	bool arsort(array &$array [, int $sort_flags])
Sort an associative array by key, maintaining index association	bool ksort(array &$array [, int $sort_flags])
Reverse sort an associative array by key, maintaining index association	bool krsort(array &$array [, int $sort_flags])
Sort an array case-insensitively in an order logically presumed by humans	bool natcasesort($array &array)
Sort an array in an order logically presumed by humans	bool natsort(array &$array)
Sort an array in reverse order	bool rsort(array &$array [, int $sort_flags])
Sort an array according to the specifications of a user-defined function	bool usort(array &$array, callback $comparison_function)
Sort an array according to the specifications of a user-defined function, maintaining index association	bool uasort(array &$array, callback $comparison_function)
Key sort an array according to the specifications of a user-defined function	bool uksort(array &$array, callback $comparison_function)

Consult the PHP manual for a complete listing: http://www.php.net/array.

STRING PARSING

PHP supports over 100 functions identified as specific to string parsing and manipulation. Following are the most commonly used tasks.

Description	Function

Converting an array to a string	`$stations = array("WTVN","WBNS","WYTS");`
	`$stations = implode(",", $stations)`
	`// $stations = "WTVN,WBNS,WYTS"`

Converting a string to an array	`$stations = "WTVN,WBNS,WYTS";`
	`$stations = explode(",", $stations);`
	`// $stations[0]="WTVN", $stations[1]="WBNS",`
	`$stations[2]="WYTS"`

Counting words in a string	`$sentence = "Columbus is home to numerous radio stations";`
	`$words = str_word_count($sentence);`
	`// $words = 7`
	See also: `count_chars()`

String Parsing, continued

Description	Function

`Converting a string to uppercase`	`$callsign = strtoupper("wtvn");`
	`// $callsign = "WTVN"`
	See also: `lcwords(), strtolower(), ucfirst(), ucwords()`

Strip HTML and PHP tags from a string	`$input = "You won the <a href="http://www.example.com">lottery!</a>."`
	`$clean = strip_tags($input);`
	`// $clean = "You won the lottery!"`
	See also: `htmlentities(), htmlspecialchars()`

Replace all occurrences of a substring	`$phrase = "Big rockers listen to rock radio";`
	`$phrase = str_replace("rock", "talk", $phrase);`
	`// $phrase = "Big talkers listen to talk radio"`
	See also: `substr_replace(), strireplace(), strtr()`

Return part of a string as specified by an offset	`$description = "WFAN: Sports Radio 66";`
	`$callsign = substr($description, 0, 4);`
	See also: `strrchr()`

Compare two strings case-insensitively	`if (strcasecmp("WTVN", "wtvn") == 0)`
	`   echo "The strings are equal in a case-insensitive context."`
	See also: `strncasecmp()`

Convert newline characters to the HTML <br /> tag	`$stations = "WTVN: 610\nWLW: 700\nWYTS: 1230";`
	`$html = nl2br($stations);`
	`// $html = "WTVN: 610<br />WLW: 700<br />WYTS: 1230"`
	See also: `htmlentities(), htmlspecialchars()`

REGULAR EXPRESSIONS

PHP's regular expression features borrow heavily from both the Perl and POSIX formats, and in fact are formally identified as such.

Perl-compatible (PCRE) Regular Expression Functions

PHP supports eight PCRE-specific functions, including these commonly used solutions:

Function	Description
array preg_grep(str $pattern, array $subject [, int $flags])	Searches $subject for $pattern, returning an array of matches. The optional $flags parameter can be set to PREG_GREP_INVERT, causing an array consisting of unmatched elements to be returned.
int preg_match(str $pattern, str $subject [, array &$matches [, int $flags [, int $offset]]])	Determines whether $pattern exists in $subject. If $matches is defined, a similarly named variable will be returned containing the matches. If $flags is set to PREG_OFFSET_CAPTURE, the string offset value will also be returned for each match. See preg_match_all() for a variation of this function.
mixed preg_replace(mixed $pattern, mixed $replacement, mixed $subject [, int $limit [, int &$count]])	Searches $subject for $pattern, replacing any instances with $replacement. See preg_replace_callback() for a variation of this function.

Common PCRE Pattern Modifiers

Modifier	Description
g	Perform a global search
i	Perform a case-insensitive search
m	Treat the string as multiple lines (
s	Ignore newline characters
x	Ignore white space and comments
u	Stop at the first match (ungreedy search)

Metacharacters

\A	Match only beginning of string
\b	Match a word boundary
\B	Match anything but word boundary
\d	Match a digit character
\D	Match a non-digit character
\s	Match a whitespace character
\S	Match a non-whitespace character
[]	Enclose a character class
()	Enclose a character grouping or define backreference
$	Match end of line
^	Match beginning of line
.	Match any character except for newline
\	Quote the next metacharacter
\w	Match any string containing underscore and alphanumeric characters
\W	Match a string containing anything but underscore and alphanumericl characters

POSIX Regular Expression Functions

PHP supports seven functions as defined by the POSIX 1003.2 specification, including these commonly used solutions:

int ereg(str $pattern, str $string [, array &$regs])	Search $string for a $pattern. You can optionally include the $regs parameter, which will cause an array of the same name to be returned containing each match. See eregi() for case-insensitive counterpart.
string ereg_replace(str $pattern, str $replacement, str $string)	Replace any patterns found in string with replacement. See eregi_replace() for case-insensitive counterpart.
array split(str $pattern, str $string [, int $limit])	Split $string into an array, dividing it according to $pattern. See spliti() for case-insensitive counterpart.

POSIX Regular Expression Syntax

[0-9]	Any decimal digit from 0 - 9
[a-z]	Any character from lowercase a through lowercase z
[A-Z]	Any character from uppercase A through uppercase Z
[A-Za-z]	Any character from upper case A through lowercase z
p+	Any string containing at least one p
p*	Any string containing zero or more p's
p?	Any string containing zero or one p
p{N}	Any string containing sequence of two p's
p{N,M}	Any string containing sequence of between N and M p's
p{2,}	Any string containing sequence of at least two p's
p$	Any string with p at the end of it
^p	Any string with p at the beginning of it
[^a-zA-Z]	Any string not containing characters a-z through A-Z
p.p	Any string containing p followed by any character, followed by another p

Regular Expression Examples

Validating a Phone Number

Presumes the required format is XXX-XXX-XXXX.

```
// PCRE
if (preg_match('/^[2-9]{1}\d{2}-\d{3}-\d{4}$/', '614-
599-2599'))
   echo "Valid number!";
// POSIX
if (ereg('^[2-9]{1}[0-9]{2}-[0-9]{3}-[0-9]{4}$', '614-
999-2599'))
   echo "Valid number!";
```

Validating a Username

Presumes username is between 6 and 10 alphabetical and numerical characters.

```
// PCRE
if (preg_match('/^[a-z0-9]{6,10}$/i', '800gilmore'))
   echo "Valid username!";
// POSIX
if (eregi('^[a-z0-9]{6,10}$', '800gilmore'))
   echo "Valid username!";
```

Turn URLs into hyperlinks

```
// PCRE
$text = "Go to http://www.wjgilmore.com.";
$html = preg_replace('/\s(\w+:\/\/)(\S+\.?)(\w+)/',
        ' <a href="\\1\\2\\3">\\1\\2\\3</a>', $text);
// POSIX
$text = "Go to http://www.wjgilmore.com. ";
$html= ereg_replace('[a-zA-Z]+://(([.]?[a-zA-
Z0-9_/-])*)', '<a href="\\0">\\0</a>', $string);
// $html = "Go to <a href=" http://www.wjgilmore.
com">http://www.wjgilmore.com."
```

TELLING TIME WITH PHP

The Date Function

The `date()` f unction is perhaps one of PHP's most commonly used functions, capable of retrieving nearly every temporal attribute of a specific timestamp.

```
string date(string $format [, $int $timestamp])
```

a	Lowercase Ante meridiem and Post meridiem
A	Uppercase Ante meridiem and Post meridiem
B	Swatch Internet Time
c	ISO 8601 date
e	Timezone identifier
g	12-hour hour format without leading zeros
G	24-hour hour format with leading zeros
h	12-hour hour format with leading zeros
H	24-hour hour format with leading zeros
i	Minutes with leading zeros
I	Specifies whether date is in daylight savings time
O	Difference to Greenwich time (GMT) in hours
P	Difference to Greenwhich time (GMT) with colon between hours and minutes
r	RFC 2822 date
s	Seconds, with leading zeros
T	Timezone abbreviation
u	Milliseconds
U	Seconds since Unix Epoch
z	Timezone offset in seconds

Day Parameters

d	Day of month, two digits with leading zeros
D	Three letter textual representation of day
j	Day of month without leading zeros
l	Textual representation of day
N	ISO-8601 numeric representation
S	Two character English ordinal suffix for day of month
w	Numeric representation of day of week
z	Numerical offset of day of year

Week Parameters

W ISO-8601	week number of year

Telling Time with PHP, continued

Month Parameters

F	Full text representation of month
m	Numeric representation of month
M	Three letter textual representation of month
n	Numeric representation of month, without leading zeros
t	Number of days in given month

Year Parameters

L	Whether date is a leap year
o	ISO-8601 year number
Y	Full numeric representation of year
y	Two digit representation of year

Date Function Examples

July 29, 2008	print date('F j, Y');
7/29/08	print date('m/j/y');
Today is Tuesday, July 29 10:45:21am	printf("Today is %s", date('l, F j h:i:sa'));
There are 31 days in July.	printf("There are %d days in %s.", date('t'), date('F'));

Setting the Timezone

You can set the timezone for all scripts by setting the date. timezone configuration directive within the php.ini file, or on a per-script basis using the `date_default_timezone_set()` function.

Other Useful Functions

Function	Description
int mktime([int $hour [, int $min [, int $sec [, int $month [, int $day [, int $year [, int $is_dst]]]]]]])	Returns the Unix timestamp for a given date
int time()	Returns current timestamp
string setlocale(int $category, string $locale)	Sets the script locale
int strtotime(string $time [, int $now])	Converts English textual date/time description into a Unix timestamp
bool checkdate(int $month, int $day, int $year)	Validates the date composed by the `$month`, `$day`, and `$year` arguments.
array getdate([int $timestamp])	Retrieves a timestamp as an associative array. Associative keys include seconds, minutes, hours, mday (day of the month), wday (day of week), mon (month), year, yday (day of the year), weekday, month, and 0 (seconds since UNIX Epoch)

PHP 5.1.0 introduced an object-oriented DateTime class. See http://www.php.net/DateTime for more information.

Date-related Examples

Output "December 25 falls on a Thursday"	$date = date('l', mktime(0,0,0,12,25,2008)); printf("December 25 falls on a %s", $date);
Output "Next month is August."	printf("Next month is %s", date('F', strtotime('+1 month')));
Output "Last Friday fell on July 25, 2008"	$date = date('F d, Y', strtotime('Last Friday')); printf("Last Friday fell on %s", $date);

Output "Oggi è martedì"	setlocale(LC_ALL, "it_IT");
	printf("Oggi &egrave; %s", strftime("%A"));

Retrieve a page's last-modified date	echo date('l, F j h:i:sa', filemtime($_SERVER["SCRIPT_NAME"]));
Calculate the difference between two dates	$date1 = strtotime("2008-08-14"); $date2 = strtotime("2008-07-11"); $diff = $date2 - $date1; printf("Difference in days: %s", $diff / 60 / 60 / 24);

MYSQL INTEGRATION

Although PHP supports several popular databases, MySQL remains by far the most common database solution. PHP's MySQL support has evolved considerably in recent years, with the MySQLi (MySQL Improved) extension being the current recommended solution. Here are the most commonly used methods.

Hot Tip The PHP 5.3 release includes a new MySQL driver known as mysqlnd (MySQL Native Driver). This driver eliminates the need for a previously required special licensing exception (FLOSS), and eliminates the need to have MySQL installed on the same machine as PHP. It has already been integrated with the mysql and mysqli extensions, with PDO support in the works.

Connecting to MySQL

The mysqli extension provides a number of ways to connect to MySQL, but the easiest involves just passing the connection data along when instantiating the `mysqli` class:

```
mysqli new mysqli([string host [, string user [, string
pswd
   [string dbname [int port [string socket]]]]]]);
```

Here's an example:

```
$mysqli = new mysqli("localhost", "webuser", "secret",
"corporate");
```

Handling Connection Errors

In case of connection error you can retrieve both the error number and error string using the `errno()` and `error()` methods. Example:

```
if ($mysqli->errno) {
   printf("Unable to connect: %s", $mysqli->error);
   exit();
}
```

Sending a Query to the Database

Once the connection has been established, you can begin querying the database. Queries are sent using the `query()` method:

```
mixed query(string $query [, int $resultmode])
```

Setting the optional $resultmode parameter to `MYSQLI_USE_RESULT` will cause `query()` to return the result as an unbuffered set.

Example:

```
$result = $mysqli->query("SELECT callsign FROM
stations");
```

Sending INSERT, UPDATE, and DELETE queries works identically. For instance, sending an UPDATE query works like this:

```
 $result = $mysqli->query("UPDATE stations SET station
= '610' WHERE callsign = 'WTVN'");
```

Retrieving Data

Data can be parsed from the result set using a number of data structures, including via associative and indexed arrays, and objects.

Retrieving data as an associative array:

```
while ($row = $result->fetch_array(MYSQLI_ASSOC) {
   printf("%S", $row["callsign"]);
}
```

Retrieving data as an indexed array:

```
while ($row = $result->fetch_row() {
   printf("%S", $row[0]);
}
```

Retrieving data as an object:

```
while ($row = $result->fetch_object() {
   printf("%S", $row->callsign);
}
```

Determining the Number of Rows Affected and Retrieved

To determine the number of affected rows after sending an INSERT, UPDATE, or DELETE query, use the affected_rows property.

Example:

```
$result = $mysqli->query("UPDATE stations SET station =
'610' WHERE callsign = 'WTVN'");
printf("Rows affected: %d", $result->rows_affected);
```

To determine how many rows were returned when using a SELECT query, use the num_rows property:

```
$result = $mysqli->query("SELECT * FROM stations WHERE
state ='Ohio');

printf("Rows affected: %d", $result->num_rows);
```

Working with Prepared Statements

Prepared statements both optimize query performance and decrease the possibility of SQL injection attacks by separating the query data from the logic, first passing the query to MySQL for preparation, binding variables to the query columns, and finally passing the data to MySQL for query execution.

To prepare a query, create the query, and then initialize a statement object using the `stmt_init()` method:

```
$query = "INSERT INTO stations VALUES(?, ?)";

$stmt = $mysqli->stmt_init();
```

Next the query is prepared by passing it to MySQL using the `prepare()` method:

```
$stmt->prepare($query);
```

Next, bind the parameters using the `bind_param()` method:

```
$stmt->bind_param('ss', "WTVN", "610");
```

Finally, execute the prepared statement using the `execute()` method:

```
$stmt->execute();
```

You can also use prepared statements to retrieve results. The general process used to execute the previous INSERT query is identical to that required for executing a SELECT query, except that the `bind_param()` method is not required, and you bind results following a call to the `execute()` method. An example follows:

```
$query = "SELECT callsign, frequency FROM stations
          ORDER BY callsign";
$stmt = $mysqli->stmt_init();
$stmt->prepare($query);
$stmt->execute();
$stmt->bind_result($callsign, $frequency);
while ($stmt->fetch())
   printf("%s: %s<br />", $callsign, $frequency);
```

Transactions

By default the MySQLi extension will render each query "permanent" upon successful execution, actually changing the database's contents when INSERT, UPDATE, and DELETE queries are processed. However the success of some tasks depend upon the successful execution of several queries, and until all have

Transactions, continued

successfully executed, no changes to the database should actually occur. ATM transactions and online credit card processing are common examples requiring several queries. Using transactions, you can change the MySQLi extension's behavior, committing a series of queries as you see fit.

To begin a transaction, start by disabling the autocommit feature:

```
$mysqli->autocommit(FALSE);
```

Execute the various queries as you see fit, and if everything proceeds as you expect, execute the `commit()` method:

```
$mysqli->commit();
```

Otherwise, if a problem occurs, execute the rollback() method:

```
$mysqli->rollback();
```

USEFUL ONLINE RESOURCES

Resource	Source
PHP Zone	http://php.dzone.com
The PHP Website	http://www.php.net
Zend Developer Zone	http://devzone.zend.com/
PlanetPHP	http://www.planet-php.net/
PHPDeveloper.org	http://phpdeveloper.org/
Developer.com	http://www.developer.com/
ONLamp PHP Devcenter	http://www.onlamp.com/php/

ABOUT THE AUTHOR

W. Jason Gilmore

Jason Gilmore is founder of W.J. Gilmore, LLC, providing web development, consulting, and technical writing services to clientele ranging from publicly traded corporations to small startups. Jason is a prolific contributor to a number of leading publications such as Developer.com, Linux Magazine, and TechTarget, with almost 100 articles to his credit. He's cofounder of the CodeMash conference (http://www.codemash.org/), a non-profit organization charged with organizing the annual namesake event.

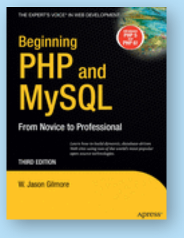
Publications

- Beginning PHP and MySQL
- Beginning PHP and PostgreSQL 8 with Robert H. Treat
- Beginning PHP and Oracle

Website

http://www.wjgilmore.com

RECOMMENDED BOOK



Beginning PHP and MySQL is the definitive book on the PHP language and MySQL database. Readers are treated to comprehensive introductions of both technologies, and in addition to in-depth instruction regarding using these two technologies in unison to build dynamic web sites.

BUY NOW
books.dzone.com/books/phpsql

DZone

DZone communities deliver over 3.5 million pages per month to more than 1.5 million software developers, architects and designers. DZone offers something for every developer, including news, tutorials, blogs, cheatsheets, feature articles, source code and more. **"DZone is a developer's dream,"** says PC Magazine.

DZone, Inc.
1251 NW Maynard
Cary, NC 27513

888.678.0399
919.678.0300

Refcardz Feedback Welcome
refcardz@dzone.com

Sponsorship Opportunities
sales@dzone.com

ISBN-13: 978-1-934238-27-1
ISBN-10: 1-934238-27-9
50795
9 781934 238271
$7.95

Version 1.0