

# Regular Expressions

This appendix introduces regular expressions in a quick and practical manner. This introduction is only the beginning, however, as the subject is so extensive that entire books have been written on it. A good example is *Regular Expression Recipes* by Nathan A. Good (Apress, 2004). Regular expressions (regexes) are a method for precisely matching strings that fit certain selection criteria. They are an old concept from theoretical computer science and are based on finite state machines.

There are many varieties of regular expressions, and they are different in many respects. The two most frequently encountered regular expression engines are Posix regular expressions and Perl compatible regular expressions (PCRE). PHP uses the latter. Actually, it can use both, but the Posix variant is deprecated in PHP 5.3 and later. The following are the main PHP functions that implement the PCRE regular expression engine:

- `preg_match`
- `preg_replace`
- `preg_split`
- `preg_grep`

There are other functions that are part of the regular expressions machinery, but these four functions are the most frequently used ones. The “preg” in each stands for “Perl regular expression,” as opposed to Posix regular expressions or “extended regular expressions.” Yes, there used to be “ereg” versions of the regular expression functions, but they are deprecated as of PHP 5.3.0.

This appendix has two parts: the first explains the PCRE regular expression syntax; the second shows examples of using them in PHP scripts.

## Regular Expression Syntax

The basic elements of regular expressions are meta-characters. When meta-characters are escaped by the backslash character (“\”), they lose their special meaning. Table A-1 is a list of the meta-characters.

**Table A-1.** *Meta-Characters and Their Meanings*

Expression	Meaning
.	Matches any single character.
*	Matches zero or more occurrences of the character expression preceding it.
?	Matches 0 or 1 occurrences of the character expression preceding it. It also makes regular expressions non-greedy. (An example of greedy and non-greedy regular expressions will come later.) It is also used for setting internal options.
/	Regular expression delimiter. It marks the beginning and the end of the regular expression.
+	Matches 1 or more occurrences of the character expression preceding it.
[ ]	Character classes: [a-z] are all lowercase letters. [Ab9] would match any of the characters 'A', 'b' or '9'. It is also possible to negate the character class with the "^" at the beginning. [^a-z] would match any character except lowercase letters.
^	Beginning of a line.
\$	End of a line.
( )	Match groupings. That will be explained in detail later.
	This is the "or" expression, separating two sub-expressions.
{ }	Quantifiers. \d{3} means "3 digits", \s{1,5} means "one, two, three, four or five space characters", Z{1,} means "one or more letters Z." That is synonymous with Z+.

In addition to meta-characters, there are also some special character classes, which are listed in Table A-2.

**Table A-2.** *Special Character Classes*

Class Symbol	Meaning
\d,\D	Lowercase symbol "d" matches a digit. Uppercase symbol "D" is a negation, and matches a non-digit character.
\s,\S	Lowercase "s" matches a space character or a tab character. Uppercase is a negation that matches any non-space character.
\w,\W	Lowercase "w" matches a "word character", that is a letter or a digit. As with the previous example, "W" is a negation of "w" and matches "any non-word character."

Regular expression `.*` will match any character. Regular expression `^.*3` will match characters from the beginning of the line until the last digit “3” on the line. This behavior can be changed; we’ll cover this later in the section about greediness. For now, let’s see some more examples of regular expressions.

## Regular Expression Examples

First, let’s see dates. Today, it is Saturday, April 30, 2011. The first pattern to match the date in this format would look like this:

```
/[A-Z][a-z]{2,},\s[A-Z][a-z]{2,}\s\d{1,2},\s\d{4}/
```

The meaning of that is “A capital letter, followed by at least two lowercase letters and a comma, followed by space, uppercase letter, at least two lowercase letters, space, 1 or 2 digits, comma, space and, finally, precisely four digits for a year.” Listing A-1 is a small PHP snippet to test regular expressions:

### *Listing A-1. Testing Regular Expressions*

```
<?php
$expr = '/[A-Z][a-z]{2,},\s[A-Z][a-z]{2,}\s\d{1,2},\s\d{4}/';
$item = 'Saturday, April 30, 2011.';
if (preg_match($expr, $item)) {
    print "Matches\n";
} else {
    print "Doesn't match.\n";
}
?>
```

Note that the `$item` variable in Listing A-2 has a dot at the end, while our regular expression finishes with `\d{4}` for the year and doesn’t match the dot at the end. If we didn’t want that, we could have “anchored” the regular expression to the end of line, by writing it like this: `/[A-Z][a-z]{2,},\s[A-Z][a-z]{2,}\s\d{1,2},\s\d{4}$/`. The newly added dollar sign at the end of the expression means “end of the line,” which means that the regular expression will not match if there are any trailing characters on the line after the year. Similarly, we could have anchored the regular expression to the beginning of the line by using the meta-character “`^`”. The regular expression that would match the entire line, regardless of the content, is `/^.*$/`.

Now, let’s take a look at a different date format, YYYY-MM-DD. The task is to parse the date and extract components.

---

■ **Note** This can easily be done with the `date` function and it is a pretty good illustration of the internal workings of some PHP functions.

---

We need not only to verify that the line contains the valid date; we also need to extract the year, month, and date. In order to do that, we’ll need to match groupings or sub-expressions. Match groupings can be thought of as sub-expressions, ordered by the sequence number. The regular expression that would enable us to perform the task at hand looks like this:

```
/(\d{4})-(\d{2})-(\d{2})/
```

The parentheses are used to match groupings. Those groupings are sub-expressions, and can be thought of as separate variables. Listing A-2 shows how to do that with the built-in `preg_match` function.

**Listing A-2.** *Matching Groupings with the Built-In `preg_match` Function*

```
<?php
$expr = '/(\d{4})-(\d{2})-(\d{2})/';
$item = 'Event date: 2011-05-01';
$matches=array();
if (preg_match($expr, $item,$matches)) {
    foreach(range(0,count($matches)-1) as $i) {
        printf("%d:-->%s\n",$i,$matches[$i]);
    }
    list($year,$month,$day)=array_splice($matches,1,3);
    print "Year:$year Month:$month Day:$day\n";
} else {
    print "Doesn't match.\n";
}
?>
```

In this script, the function `preg_match` takes the third argument, the array `$matches`. Here is the output:

```
./regex2.php
0:-->2011-05-01
1:-->2011
2:-->05
3:-->01
Year:2011 Month:05 Day:01
```

The 0<sup>th</sup> element of the array `$matches` is the string that matches the entire expression. That is not the same as the entire input string. After that, each consecutive grouping is represented as an element of the array. Let's see another, more complex example. Let's parse a URL. Generally the form of a URL is the following:

```
http://hostname:port/loc?arg=value
```

Of course, any part of the expression may be missing. An expression to parse a URL of the form described above would look like this:

```
/^https?:\/\/[^\:\/]+?:?\d*\/[^\?]*.*\/
```

There are several new elements worth noticing in this expression. First is the `s?` part in the `^http[s]?:`. That matches either `http:` or `https:` at the beginning of the string. The caret character `^` anchors the expression to the beginning of the string. The `?` means “0 or 1 occurrences of the previous expression.” The previous expression was letter `s` and translates into “0 or 1 occurrences of the letter `s`.” Also, the slash characters `/` were prefixed with the backslash characters `\` to remove the special meaning from them.

PHP is extremely lenient when it comes to the regular expression delimiter; it allows changing it to any other delimiter. PHP would recognize brackets or the pipe characters `|`, so the expression would have been equally valid if written as `[^https?:\/\/[^\:\/]+?:?\d*\/[^\?]*.*]`, or even using the pipe character as a delimiter: `|^https?:\/\/[^\:\/]+?:?\d*\/[^\?]*.*|`. The general way of stripping the special meaning from the special characters is to prefix them with a backslash character. The procedure is also known as “escaping special characters.” Regular expressions are smart and can figure out the meaning of the

characters in the given context. Escaping the question mark in `[^?]*` was unnecessary because it is clear from the context that it denotes the class of characters different from the question mark. That doesn't apply to the delimiter character like `/`; we had to escape those. There is also the `[^\:/]+` part of the expression, which stands for "one or more characters different from the colon or slash." This regular expression can even help us with slightly more complex URL forms. See Listing A-3.

**Listing A-3.** *Regular Expressions with Complex URL Forms*

```
<?php
$expr = '[^https*://([^\:/]+)?\d*/[^\?]*.*]';
$item = 'https://myaccount.nytimes.com/auth/login?URI=http://';
if (preg_match($expr, $item)) {
    print "Matches\n";
} else {
    print "Doesn't match.\n";
}
?>
```

This is the login form for the *New York Times*. Let's now extract the **host**, **port**, **directory**, and the argument string, using groupings, just like we did in Listing A-2 (see Listing A-4).

**Listing A-4.** *Extracting the host, port, directory, and argument String*

```
<?php
$expr = '[^https*://([^\:/]+)?(\d*)/([^\?]*\??(.*))]';
$item = 'https://myaccount.nytimes.com/auth/login?URI=http://';
$matches = array();
if (preg_match($expr, $item, $matches)) {
    list($host, $port, $dir, $args) = array_splice($matches, 1, 4);
    print "Host=>$host\n";
    print "Port=>$port\n";
    print "Dir=>$dir\n";
    print "Arguments=>$args\n";
} else {
    print "Doesn't match.\n";
}
?>
```

When executed, this script will produce the following result:

```
./regex4.php
Host=>myaccount.nytimes.com
Port=>
Dir=>auth/login
Arguments=>URI=http://
```

## Internal Options

The value for the port wasn't specified in the URL, so there was nothing to extract. Everything else was extracted properly. Now, what would happen if the URL was written with capital letters, like this:

```
HTTPS://myaccount.nytimes.com/auth/login?URI=http://
```

That would not match, because our current regular expression specifies the lowercase characters, yet it is a completely valid URL that would be properly recognized by any browser. We need to ignore case in our regular expression if we want to allow for this possibility. That is achieved by setting the “ignore case” option within the regular expression. The regular expression would now look like this:

```
[(?i)^https?:\/\/([^\:\/]+):?(\\d*)\\/([^\?]*\\)??(.*)]
```

For any match after `(?i)` the case will be ignored. Regular expression `Mladen (?i)g` would match both strings `Mladen G` and `Mladen g`, but not `MLADEN G`.

Another frequently used option is `m` for “multiple lines.” Normally, regular expression parsing stops when the newline characters “`\n`” is encountered. It is possible to change that behavior by setting the `(?m)` option. In that case, parsing will not stop until the end of input is encountered. The dollar sign character will match newline characters, too, unless the “`D`” option is set. The “`D`” option means that the meta-character “`$`” will match only the end of input and not newline characters within the string.

Options may be grouped. Using `(?imD)` at the beginning of the expression would set up all three options: ignore case, multiple lines, and “dollar matches the end only.”

There is also an alternative, more traditional notation for the global options that allows the global modifiers to be specified after the last regular expression delimiter. Using that notation, our regular expression would look like this:

```
[^https?:\/\/([^\:\/]+):?(\\d*)\\/([^\?]*\\)??(.*)]i
```

The advantage of the new notation is that it may be specified anywhere in the expression and will only affect the part of the expression after the modifier, whereas specifying it after the last expression delimiter will inevitably affect the entire expression.

---

■ **Note** The full documentation of the global pattern modifiers is available here: [www.php.net/manual/en/reference.pcre.pattern.modifiers.php](http://www.php.net/manual/en/reference.pcre.pattern.modifiers.php)

---

## Greediness

Normally, regular expressions are greedy. That means that the parser will try to match as large a portion of the input string as possible. If regular expression `'(123)+'` was used on the input string `'123123123123123A'`, then everything before the letter `A` would be matched. The following little script tests this notion. The idea is to extract only the `img` tag from an HTML line and not any other tags. The first iteration of the script, which doesn’t work properly, will look like Listing A-5.

**Listing A-5.** *Insert Listing Caption Here.*

```
<?php
$expr = '/<img.*>/';
$item = '<a>text</a>';
$matches=array();
if (preg_match($expr, $item,$matches)) {
    printf( "Match:%s\n", $matches[0]);
}
```

```

} else {
    print "Doesn't match.\n";
}
?>

```

When executed, the result would look like this:

```

./regex5.php
Match:text</a>

```

---

■ **Note** Some browsers, most notably Google Chrome, will attempt to fix bad markup, so both greedy and non-greedy output will exclude the stray `</a>`.

---

We matched more than we wanted, because the pattern `".*>"` matched as many characters as possible until it reached the last `">"`, which is a part of the `</a>` tag, not of the `<img>` tag. Using the question mark will make the `"*"` and `"+"` quantifiers non-greedy; they will match the minimal number of characters, not the maximal number of characters. By modifying the regular expression into `'<img.*?>'`, the pattern matching would continue until the first `">"` character was encountered, producing the desired result:

```
Match:
```

Parsing HTML or XML is a typical situation in which non-greedy modifiers are utilized, precisely because of the need to match the tag boundaries.

## PHP Regular Expression Functions

So far, all we've done is to check whether the given string matches the specification, written in a convoluted form of PCRE regular expression, and extract elements from the string, based on the regular expression. There are other things that can be done with regular expressions, such as replacing strings or splitting them into arrays. This section is devoted to the other PHP functions that implement the regular expression mechanisms, in addition to the already familiar `preg_match` function. The most notable one among these is `preg_replace`.

### Replacing Strings: `preg_replace`

The `preg_replace` function uses the following syntax:

```
$result = preg_replace($pattern,$replacement,$input,$limit,$count);
```

Arguments `$pattern`, `$replacement`, and `$input` are self explanatory. The `$limit` argument limits the number of replacements, with -1 meaning no limit; -1 is the default. The final argument, `$count`, is populated after the replacements have been made with the number of replacements actually performed, if specified. It may all look simple enough, but there are further ramifications. First of all, the pattern and replacement can be arrays, as in the Listing A-6.

**Listing A-6.** *Insert Listing Caption Here.*

```
<?php
$cookie = <<<'EOT'
    Now what starts with the letter C?
    Cookie starts with C
    Let's think of other things that starts with C
    Uh ahh who cares about the other things

    C is for cookie that's good enough for me
    C is for cookie that's good enough for me
    C is for cookie that's good enough for me

    Ohh cookie cookie cookie starts with C
EOT;
$expression = array("/(?i)cookie/", "/C/");
$replacement = array("donut", "D");
$donut = preg_replace($expression, $replacement, $cookie);
print "$donut\n";
?>
```

When executed, this little script produces a result that probably wouldn't appeal to the Cookie Monster from *Sesame Street*:

```
./regex6.php
    Now what starts with the letter D?
    donut starts with D
    Let's think of other things that starts with D
    Uh ahh who cares about the other things

    D is for donut that's good enough for me
    D is for donut that's good enough for me
    D is for donut that's good enough for me

    Ohh donut donut donut starts with D
```

The important thing to notice is that both the pattern and replacement are arrays. The pattern and replacement array should have an equal number of elements. If there are fewer replacements than patterns, then the missing replacements will be replaced by null strings, thus effectively destroying the matches for the remaining strings specified in the patterns array.

The full strength of the regular expressions can be seen in Listing A-7. The script will produce SQL-ready truncate table commands from the list of provided table names. This is a rather common task. For the sake of brevity, the list of tables will already be an array, although it would typically be read from a file.



*Listing A-7. Insert Listing Caption Here.*

```
<?php
$tables = array("emp", "dept", "bonus", "salgrade");
foreach ($tables as $t) {
    $trunc = preg_replace("/^(\\w+)/", "truncate table $1;", $t);
    print "$trunc\\n";
}
```

When executed, the result would look like this:

```
./regex7.php
truncate table emp;
truncate table dept;
truncate table bonus;
truncate table salgrade;
```

The use of `preg_replace` shows several things. First, there is a grouping `(\\w+)` in the regular expression. We saw groupings in the previous section, when extracting date elements from a string in Listing A-2. That grouping also appears in the replacement argument, as “\$1”. The value of each sub-expression is captured in the variable `$n`, where `n` can range from 0 to 99. Of course, as is the case with `preg_match`, `$0` contains the entire matched expression and the subsequent variables contain the values of sub-expressions, numbered from left to right. Also, note the double quotes here. There is no danger of confusing variable `$1` for something else, as the variables of the form `$n`, `0<=n<=99` are reserved and cannot be used elsewhere in the script. PHP variable names must start with a letter or underscore, that is a part of the language specification.

## Other Regular Expression Functions

There are two more regular expression functions to discuss: `preg_split` and `preg_grep`. The first of those two functions, `preg_split`, is the more powerful relative of the `explode` function. The `explode` function will split the input string into array of elements, based on the provided delimiter string. In other words, if the input string is `$a="A,B,C,D"`, then the `explode` function, used with the string “,” as the separator, would produce the array with elements “A”, “B”, “C” and “D”. The question is how do we split the string if the separator is not of a fixed format and the string looks like `$a='A, B,C .D'`? Here we have space characters before and after the separating comma and we also have a dot as a separator, which makes using just the `explode` function impossible. The `preg_split` has no problems whatsoever. By using the following regular expression, this string will be flawlessly split into its components:

```
$result=preg_split('/\\s*[,.]\\s*/',$a);
```

The meaning of the regular expression is “0 or more spaces, followed by a character that is either a dot or a comma, followed by 0 or more spaces.” Of course, adding regular expression processing is more expensive than just comparing the strings, so one should not use `preg_split` if the regular `explode` function suffices, but it’s really nice to have it in the tool chest. The increased cost comes from the fact that regular expressions are rather complex beasts under the hood.

---

■ **Note** Regular expressions are not magic. They require caution and testing. If not careful, one can also get unexpected or bad results. Using a regular expression function instead of a more familiar built in function doesn't, by itself, guarantee the desired result.

---

The `preg_grep` function should be familiar to all those who know how to use the command line utility called `grep`. That's what function was named after. The syntax of the `preg_grep` function looks like this:

```
$results=preg_grep($pattern,$input);
```

The function `preg_grep` will evaluate regular expression `$pattern` for each element of the input array `$input` and store the matching output in the resulting array `results`. The result is an associative array with the offsets from the original array provided as keys. Listing A-8 shows an example, based on the file system `grep` utility:

*Listing A-8. Insert Listing Caption Here.*

```
<?php
$input = glob('/usr/share/pear/*');
$pattern = '/\..php$/';
$results = preg_grep($pattern, $input);
printf("Total files:%d PHP files:%d\n", count($input), count($results));
foreach ($results as $key => $val) {
    printf("%d ==> %s\n", $key, $val);
}
?>
```

The dot in the `.php` extension was escaped by a backslash character, because dot is a meta-character, so to strip its special meaning, it had to be prefixed by `"\"`. The result on the system used for writing this appendix looks like this:

```
./regex8.php
Total files:35 PHP files:12
4 ==> /usr/share/pear/DB.php
6 ==> /usr/share/pear/Date.php
8 ==> /usr/share/pear/File.php
12 ==> /usr/share/pear/Log.php
14 ==> /usr/share/pear/MDB2.php
16 ==> /usr/share/pear/Mail.php
19 ==> /usr/share/pear/OLE.php
22 ==> /usr/share/pear/PEAR.php
23 ==> /usr/share/pear/PEAR5.php
27 ==> /usr/share/pear/System.php
29 ==> /usr/share/pear/Var_Dump.php
32 ==> /usr/share/pear/pearcmd.php
```

The result may look differently on a different system, which has different PEAR modules installed. The `preg_grep` function can save us from checking regular expressions in a loop, and is quite useful.

There are several other regular expression functions that are used much less frequently than the functions described in this appendix. Those functions are very well documented and an interested reader can look them up in the online documentation, at [www.php.net](http://www.php.net).

# Index

## ■ A

Abstract class, 17

ACID requirements, 128

Adobe Flash Builder, 52-53

ADODB

- ADODB connection, 174

- Caching, 176-177

- characteristics, 173

- classes, 173

- CompleteTrans method, 175

- CSV file loading, 173-174

- execute method, 174-175

- Fetch Row and Field method, 176

- Linux distribution, 172

- system directory, unpacking, 173

- varieties, 173

ADODB set, 17

Agile development

- Bugzilla

  - bug/task listing, 266-267

  - home page, 266

  - level of granularity, 266

  - Mylyn (*see* Mylyn for eclipse)

  - open source, 266

    - project management, 267

  - definition, 263

  - navigator, 264

  - planning, 264

  - rally organizer

    - advantages, 265

    - course, 264

    - keystroke, 264

    - pit stops, 265

    - project leader, 264

  - roles, 264

  - team developers, 263

Ajax

- asynchronous *vs.* synchronous events, 357-358

- components, 347

- data sending, PHP script

  - Ajax request loading, HTML, 374

  - predator and prey animal, 373

  - sample output, 374

- High level JavaScript APIs, 367

- jQuery

  - concise version, 370

  - <p> element modification, 367-368

- \$.get and \$.post functions*, 371
- JSON data outputting, PHP array, 371–373
- plain text loading, XML file, 369–370
- negative points, 355
- simple graphic program
  - drawing grid, 377
  - Firefox extension Firebug, 383
  - HTML, 378, 380–381
  - load function, 382
  - PHP script, 381
  - save function, 382
  - table cells color manipulation, 375–377
- Traditional Web model, 355
- Web model, 356–357
- XMLHttpRequest object (*see* XMLHttpRequest object)
- Alternative PHP Cache (APC), 108
- Android development, 52
- Anonymous function
  - array\_reduce function, 97
  - create\_function solution, 98
- Application programming interface (API), 108
- \$argv array, 23
- Array Interface
  - auto-commit feature, 198
  - binding variable, 196
  - data file loading, 195
  - database infrastructure, 196–197

- dynamic space allocation, 195
- file reading and loading, 195–196
- loading time, 198
- PHP script writing, 198
- SAVE EXCEPTION option, 198
- \_\_autoload function, 6

## ■ B

- Basic authentication to connect, 60

## ■ C

- \_\_call method, 9, 10
- Clone operator, 11
- Closures. *See* Anonymous function
- Code refactoring
  - behavioral changes, 278
  - code length, 278
  - code repetition, 278
  - functionality modification, 277
  - larger legacy code
    - config.php, 287
    - location class, 287
    - travel class, 293, 296
    - travel function, 282–286
    - travelMath.php*, 292–293
    - travelView.php*, 288–291
  - small refactorings, 278
    - configuration options*, 279
    - Go for a Walk code*, 278
    - logical expression, readability*, 279

- logical function separation*, 280
  - PHP Script*, 281–282
  - Column constraints, 151
  - ColumnName() method, 160
  - ColumnType, 160
  - Comma separated values (CSV) files, 155
  - CompleteTrans method, 175
  - Constraints, 151–152
  - Continuous integration (CI), 277
    - build automation, 317
    - extra build steps, 314
    - Jenkins Server Setup, 318
      - Jenkins main menu, 321
      - Jenkins management, GUI, 319
      - Jenkins web site, 318
    - multistep task, 314
    - server, 315
    - static analysis, 316
    - version control, 315
  - CouchDB
    - Futon
      - couchClient, 146
      - database and collection creation, 143–144
      - document retrieval and updation, 148–149
      - getView method, 149–150
      - JavaScript function, 149
      - JSON object, 149
      - newly ingested document, 146–147
      - PHP script assessment, 145–146
      - records and browser, 147–148
      - revision field, 148
      - scott database, 144
      - views, 145
    - HTTP protocol, 143
    - vs.* JSON objects, 143
    - MVCC, 142
  - Cross site scripting (XSS), 248
    - Alert box, 249
    - document cookie, 249
    - interpolated markup, 250
    - PHP handling, 249, 250
    - prevention, 251
  - Cross-site request forgery (CSRF)
    - HTTP request, 251
    - prevention, 252
    - session fixation, 252–253
- ## ■ D
- Data security
    - attacks
      - CSRF, 251–253
      - same origin policy, 248
      - XSS, 248–250, 251
    - filter extension
      - filter\_has\_var, 257
      - filter sanitization, 257
      - filter\_var function, 254
      - flags, 255–256
      - multiple variable, 257

- superglobal variables, 255
- never trust data
  - Ajax, 247
  - \$\_COOKIES, 247
  - register\_globals, 244
  - \$\_SERVER, 247
  - \$\_SESSION, 247
  - whitelists and blacklists, 245–246
  - X-files, 243
- php.ini file
  - hardening process, 258–259
  - password algorithm, 260–261
  - server environment, 258
- SQL injection, 253–254
- Data validation
  - GET method, 111
  - JavaScript, 111–112
  - PCRE library
    - \$matches array, 116–118
    - preg\_match\_all function, 116
  - PHP validation
    - filter\_var function, 114–115
    - preg\_match function, 114
    - URL and e-mail validate filters, 115–116
  - POST method, 112
  - search function, 113
- Database resident connection pooling (DRCP), 207
- Deep copy, 11

- Desktop client, 61
- Device detection system
  - chrome browser, 33
  - get\_browser function, 32–33
  - regex generation, 35
  - regex usage, 35
  - user-agent, 32
  - WURFL (*see* Wireless Universal Resource File)
- \$(document).ready function call, 368
- Document upload process
  - enctype attribute, 118
  - \$\_FILES metadata, 118–119
  - HTTP POST, 119
  - \_uploaded\_file() function, 119
- DOMDocument
  - element and attribute value searching, 343–344
  - element finding, 342–343
  - features, 341
  - node types, 341
  - save functions, 342

## ■ E

- \$EMP array, 131
- empno attribute, 132
- empty() function, 9
- Enctype attribute, 118
- Exception handlers, 23

## Extensible Markup Language (XML)

### DOMDocument

- element and attribute value searching, 343

- element finding, 342–343

- features, 341

- node types, 341

- save functions, 342

primer, 323–324

Reader and Writer, 344–345

schema

- stricter schema, 324

- xhtml1-transitional schema, 325

simpleXML (*see* SimpleXML)

## ■ F

### Facebook

- account verification, 80–81

- application profile page, 85

- canvas page, 83

- credit card information, 81

- Graph API, 89

- integration canvas settings, 83

- logout callback, 87–88

- official SDK, 82

- photo albums, 90

- requesting additional permission, 88

- terms of service, 81

FetchArray method, 160

File systems, 202

Firefox 4.0, 32

Firefox extension Firebug, 383

Foreign key constraints, 151–152

Full-text search

- logical operators, 178

- open source text engine, 178

- phrase searches, 177

- proximity searches, 177

- Quorum searches, 178

- Sphinx (*see* Sphinx)

- Word searches, 177

## ■ G

\_\_get and \_\_set Method, 8, 9

get\_browser function, 32–33

GET method, 111–112

Get next method, 16

global temporary tables, 189

goto statement, 101–102

## ■ H

harrison\_bergeron.txt, 202

High Level JavaScript APIs, 367

## ■ I

IGNORE flag, 126

Image conversion and thumbnails

- GD library, 119

- getimagesize() function

- image name, 119–120

- image.jpg and thumb.png, 120–121

- \$metadata array, 120



- thumbnail creation, 121
- Integrated Development Environments (IDEs), 298
- Interface, 13
  - vs.* abstract class, 17
  - interface `interf2`, 13–14
- Iterator implementation
  - ADODB, 17
  - arrays, 15–16
  - database cursor, 16
  - loops, 14–15
  - rewind function, 16
  - Standard PHP Library, 16
- Internal object handling
  - clone operator, 11–12
  - cloning, 12
  - deep copy, 11
  - destructor, 11
  - object comparison, 12
  - shallow copy, 11
  - `$x=$y` execution, 10–11
- Internet resources
  - ConFoo, 388
  - devzone.zend.com, 387
  - information dissemination, 387
  - International PHP conference, 388
  - open source India, 388
  - OSCON, 388
  - PHP certification, 388–390
  - www.php.net, 385–386

- www.phparch.com, 387
- www.zend.com, 386
- ZendCon, 388
- ISO-88591 string, 126
- `__isset` method, 9

## ■ J, K

- JavaScript validation, 111, 113
  - AJAX request, 113
  - `select()` method, 112
- jQuery
  - background colors, 377
  - concise version, 370
  - `<p>` element modification, 367–368
  - `$.get` and `$.post` function, 370–371
  - JSON data outputting, PHP array, 371–373
  - load function, 370
  - plain text loading, XML file, 369–370
  - wrap function, 370
- JSON
  - data types, 348
  - PHP arrays
    - associative key, 353–354
    - encoding and decoding, 350
    - functions, 349
    - JSON representation output, 351
    - PHP encoding output, 352
    - PHP nested array, 350–351
- JSON arrays, 348–349

JSON document, 127

JSON objects, 348–349

## ■ L

Lambda function. *See* Anonymous function

Large object (LOB)

characteristics, 202

column creation

DISABLE STORAGE IN ROW option, 203

file name and content, 203–204

LOB descriptor, 204

LOB segment and locator, 203

OCI-Lob class, 205

OCI-Lob->read(), 205–206

RETURNING clause, 204

PHP application, 202

relational database, 202

table creation, 202–203

text file, 202

types, 202

Local temporary tables, 189

Logical operators, 178

## ■ M

Magic methods, 8

\_\_call method, 9–10

\_\_get and \_\_set Method, 8–9

\_\_isset method, 9

\_\_toString() method, 10

Mobile PHP

CSS, 51–52

device detection

chrome browser, 33–34

get\_browser function, 33

regex generation, 35

regex usage, 34–35

user-agent, 32

WURFL (*see* Wireless Universal Resource File)

emulator and SDK, 52–53

image resizing tool, 50–51

mobile variance, 31

QR codes, 53–54

WALL, 48–49

MongoDB

aggregation, 139–142

characteristics, 128–129

data types, 129

\$EMP array, 131

empno attribute, 132–133

PHP script, 129–130

queries, 133–134

comm attribute, 135

count() function, 134

deptno attribute, 134

find() method, 134

foreach loop method, 134

object id checking, 136–137

- operators, 134
- regular expressions, 135–136
- safety, 133
- scott database, 131
- updation
  - denormalization, 137
  - department information, 137
  - documents, scripts, 137–139
  - hiredate attribute, 137
  - MongoDB Collection Information Design, 137
- Mozilla, 32
- Multi-Language Integration, 125–126
- multiple version consistency control (MVCC), 142
- Mylyn for eclipse
  - benefits, 274–275
  - Buzilla server content, 268
  - context recording, 271
  - context tracking, 271
  - individual task, 268–269
  - jpg image, 271–272
  - preferences window, 272
  - query processing, 270
  - Task List view, 271–273
  - Task Repositories view, 268–269
  - task repository query, 270
  - team development, 270
  - UI Legend, 273
  - Zend Studio, 268
- MySQL database creation, 126

- MySQLi extension
  - COMMIT statement, 165
  - components, 161
  - CSV files, 162–165
  - emp.csv and dept.csv, 163
  - fetch\_field method, object retriving, 167–168
  - format string, 165
  - LOAD DATA command, 162–163
  - MYSQLI\_RESULT, 166
  - MYSQLI\_STMT properties, 165
  - name property, 167–168
  - PHP statement, 166
  - query mehtod, 167
  - report writing, 166–167
  - scott database, 162
  - statement bind method, 165
  - table description, 162
  - TIMESTAMP type, 163

## ■ N

- Namespaces. *See also* Simple XML
  - autoload function, 96–97
  - class context, 96
  - domestic.php, 94–95
  - multiple namespaces, animals.php, 95
  - predefined global namespace, 96
  - programming libraries, 93
  - sub-namespaces, 95–96
- NLS\_LANG variable, 210
- Non\_existing\_method, 10

Non-LOB columns, 203

NonNumericException handler, 24

NoSQL databases

CouchDB (*see* CouchDB)

goals, 127

JSON document, 127

MongoDB (*see* MongoDB)

SQLite (*see* SQLite)

Nowdoc

backslash characters, 98

*vs.* heredoc, 100–101

heredoc syntax, 98–99

NULL data type, 151

## ■ O

Object orientation

interface (*see* Interface)

internal object handling (*see* Internal object handling)

PHP Classes (*see* PHP Classes)

OCI8 extension

CSV file loading, 192–193

number of fields, 194

OCI8 calls, 195

oci\_fetch\_array, 195

OCI\_NO\_AUTO\_COMMIT argument, 193–194

PHP script, 194

query execution, 194

OCI-Lob->flush() method, 205

OCI-Lob->import, 205

Onreadystatechange callback function, 359

Open authentication (OAuth), 57–58

Operating systems (OS), 31

Oracle RDBMS

Array Interface (*see* Array Interface)

character sets and PHP, 209–211

characteristics, 189

connection pooling

advantages, 207, 209

configuration, 207–208

default pool arguments, 208

Java and server application, 207

oci\_new\_connection, 209

oci\_pconnect command, 209

Oracle 11, 207

Oracle 7, 207

parameter oci8.events, 208

production database, 207

LOB (*see* Large object (LOB))

OCI8 extention (*see* OCI8 extention)

options and capability, 191

PL/SQL, 190

PL/SQL, procedure and cursors, 198–201  
real application cluster, 190–191

row-level locking, 190

server attachment process, 189

SQL Server application, 190

- synonyms, 190
- tablespaces, 189
- Oracle\*Text, 202

## ■ P

*Parallel, asynchronous HTTP events*, 358

Perl Compatible Regular Expression (PCRE) library, 116, 391, 398–400

Personal identification number (PIN), 58

- getAccessToken, 64
- output, 65
- seven-digit number, 64
- Twitter API usage, 67–70
- twitter\_registration.php, 64–65
- validation, 65–66

Phar Extension, 105

- Alternative PHP Cache, 108

Phar archiver

- creation, 106
- pack argument, 106
- phar list, 107
- stub script, 107
- zip algorithm, 106

- PHP 5.3 distribution, 106

- unix-based system, 107–108

- wild.php and domestic.php, 106

PHP Classes

- autoloading, 1
- constructor method, 2
- 2-dimensional plane, 1–2
- encapsulation, 1

inheritance and overloading

- abstract classes, 5–6
- autoload function, 6
- class declarations, 7
- construct method, 4
- destructor method, 5
- directive, 7
- employee class, 3–4
- get\_raise method, 5
- give\_raise method, 5
- OO programming technique, 6
- parent functions, 4
- trailing, 7

magic method, 2 (*see also* Magic methods)

private members, 2

protected methods and members, 3

public members, 2

scope

- class context, 19
- get\_prop function, 19
- keyword self, 18
- static variables, 17–18

terminology, 1

PHP data objects (PDO)

- ACID requirements, 170
- application program interface, 169
- autocommit mode, 170
- CSV loading script, 169–170
- error-handling code, 170

- execute method, 170
- fixed query, 171
- getColumnMeta method, 171–172
- layers, 169
- LIMIT option, 170
- PDOException, 170
- query execution, 172
- setAttribute call, 170
- PHP Exception
  - \$argv array, 23
  - characteristics, 21
  - definition, 21
  - erroneous event, 22
  - NonNumericException handler, 23–25
  - SplFileObject, 25–26
  - synchronous events, 21
  - syntax
    - exception class, 23
    - exception handler, 23–24
    - throw statement, 23
- PHP interpreter, 173
- PHP library
  - e-mail
    - basic usage, 235
    - HTML formatted message, 235–236
    - SMS, 237–238
    - SMTP server, 236
  - Google Chart API
    - candlestick marker, 240
    - Candlestick-Style chart, 240–241
    - colored map, 238
    - country code, 239
    - dynamic graphs and charts, 238
    - gLineChart object, 241–242
    - map, 239
  - Google Map integration
    - boiler template, 231
    - download package, 231
    - Effile tower, 232
    - satellite imagery, 232
    - terrain, multiple marker, 234
    - traffic routes, 233
  - SimplePie
    - advantage, 215
    - basic usage, 214
    - browser output, 215
    - cache, 214–215
    - cURL extension, 214
    - favicon and custom formatted data, 215–216
    - favicon and story creator, 218
    - RSS and Atom feed consumption, 214
    - structure of element, 216–217
  - TCPDF
    - AddPage method, 219
    - barcode and gradient generation, 223–224
    - document generation, 218
    - image and HTML formatted text, 220–221

- language configuration and library file, 219
- line breaks, 222
- Write method, 219
- website data scraping
  - cURL usage, 226
  - DOMDocument, 225
  - file\_get\_contents, 226–227
  - information retrieval, 225
  - page scraping, 225
  - phpQuery, 229–231
  - remote content grabbing, 225
  - webpage link finding, 227–228
- PHP references
  - array member, 28
  - assignment by reference, 28
  - copying, 27–28
  - Function and Pass Arguments, 29
  - loops, 28
  - non-object types, 27
  - normal assignment, 28
  - object, 26–27
  - returning values, 30
  - Visibility Protection, 29–30
- Phrase searches, 177
- Placeholder, 157
- Posix regular expressions, 391
- POST method, 112
- Primary key constraints, 151

- Process global area (PGA), 207
- Proximity searches, 177

## ■ Q

- Quick Response (QR) codes, 53–54
- Quorum searches, 178

## ■ R

- Real application clusters (RAC), 190–191
- Really Simple Syndication (RSS), 334–335
- Regression tests, 302
- Regular expressions
  - array \$matches, 394
  - built-in preg\_match function, 394
  - characters classes, 392
  - greediness, 396
  - host, port, directory and argument string extraction, 395
  - internal options, 396
  - match characters, 393
  - meta-characters, 391–392
  - PHP functions, 391
  - PHP snippet, 393
  - preg\_split and preg\_grep function, 400–401
- Regex characters and matches, 121–123
- String methods
  - compile, exec and test*, 124
  - match, search, replace and split, 123
  - PCRE functions, 124–125

- sub-expressions, 393–394
- types, 391
- URL forms, 395
- Relational Database Management System (RDBMS), 128
- RGB array triplets, 349

## ■ S

- Sequential, synchronous HTTP events*, 358
- `set_exception_handler`, 23
- `setRequestHeader` function, 360
- Shallow copy, 11
- Share global area (SGA), 207
- shared-nothing architecture, 190
- Simple API for XML (SAX), 324
- SimplePie
  - advantage, 215
  - basic usage, 214
  - cache, 214
  - cURL extension, 214
  - favicon and custom formatted data, 215, 216
  - favicon and story creator, 218
  - RSS and Atom feed consumption, 214
  - structure of element, 216–217
- SimpleXML
  - namespaces, XPath
    - attribute value finding, 333
    - registered namespaces, 332, 333
    - unregistered namespaces, 331

- parsing
  - `attribute()` function, 328
  - attribute value matching, XPath, 331
  - children and parent element matching, XPath, 330–331
  - content finding, 327–328
  - element finding, XPath, 329–330
  - output, 328
  - PHP warning message, 326
  - string, 325–326
- Really Simple Syndication, 334–336
- XML generation, 336, 338–341

## Social media

- database
  - security, 74
  - Singleton design pattern, 75
  - `twitter_db_actions.php`, 75–76
  - `twitter_db_connect.php`, 74
  - updated callback script, 76–77

- data cache storage, 77

## Facebook

- account verification, 80–81
- application profile page, 85
- Apress page, 84
- canvas page, 83
- credit card information, 81
- Graph API, 89–90
- integration canvas settings, 83
- `login.php`, 85–86
- logout callback, 87–88



- official SDK, 82
  - permission request page, 86
  - photo albums, 90–91
  - requesting additional permission, 88
  - terms of service, 81
- OAuth, 57–58
  - My Access Token, 63–64
- Twitter
  - access token storage, 78
  - account, 60–62
  - callback script, 70–73
  - client authentication with PIN (*see* Personal identification number (PIN))
  - destroying a status, 79
  - friend update list, 78
  - friendship, creation/destroy, 80
  - GET, 78
  - private rest API, 60
  - public search API, 58–60
  - tweets and IDs, 79
  - twitter\_oauth\_signin.php, 78
- Twitter's OAuth login, 73–74
- Software developer kits (SDKs), 52
- Sphinx
  - associative array, 185–186
  - configuration file, 179–180
  - document\_id column, 180
  - food\_articles table, 178
  - index, 181
    - max\_matches option, 181
    - PHP script, 182–184
    - program indexer, 181
    - search program, 181–182
  - indexer and search process, 178
  - MySQL database, 187
  - operating system, 178
  - output, 184–185
  - PostgreSQL, 178
- SplFileObject class, 16, 25–26
- SplMinHeap class, 6
- SQLite
  - characteristics, 150
  - columnName() method, 160
  - columnType, 160
  - data dictionary, 153
  - data printing, 158–159
  - database structure, 154–155
  - \$DDL command, 155
  - emp and dept tables, 156–158
  - execute method, 158–160
  - fetchArray method, 160
  - insert statement, 157
  - NULL data type, 151
  - placeholders, 157
  - programming interface, 153
  - query method, 155
  - query optimizer, 152
  - relational databases, 152
  - SELECT statements, 152
  - statement class, 157–158

- tables and index creation, 155
- try block, 157
- two CSV files, 155–156
- views and constraints, 151–152
- sqlite\_master, 153
- Standard PHP Library (SPL)
  - class extention, 103–104
  - compare function, 103
  - FileSystemIterator class, 104
  - flags, 104
  - Globiterator class usage, 105
  - SplFileObject class, 104
  - SplMaxHeap script, 102–103
  - sub-directories, 104–105
- Strings, 348

## ■ T

- tecnick.com PDF (TCPDF)
  - AddPage method, 219
  - barcode and gradient generation, 223–224
  - document generation, 218
  - image and HTML formatted text, 220–221
  - language configuration and library file, 219
  - line breaks, 222
  - Write method, 219
- Test Driven Development (TDD), 313
- \_\_toString() method, 10
- Trailing, 7

- Try block, 23

## Twitter

- access token storage, 78
- account, 60–62
- callback script, 70–73
- destroying a status, 79
- friend update list, 78
- friendship, creation/destroy, 80
- GET, 78
- private rest API, 60
- public search API, 58, 60
- tweets and IDs, 79
- twitter\_oauth\_signin.php, 78
- Twitter application sign-up form, 60–61
- twitter\_get\_search.php program, 59

## ■ U, V

- Unique constraints, 151
- Unit testing, 277
  - code coverage, 302
  - object oriented Walk Class, 297
- PHPUnit
  - error verification, 308
  - file coverage, 303
  - refactoring, 309, 311–313
  - Sample Command line output, 303
  - Sample Zend Studio output, 303
  - TravelMath class, 304–306
  - TravelView class, 308
  - unexpexted errors, 307

- PHPUnit\_Framework\_TestCase, 299
- second test addition, 301
- side effects, 302
- successful test result, 301
- Test Driven Development, 313
- unit test failure, 300–301
- unit test skeleton, 299
- UTF-8 character, 126

## ■ W

- Web-based form design and management
  - data validation (*see* Data validation)
  - document uploading (*see* Document uploading process)
  - image conversion and thumbnails, 119–120
  - multi-language integration (*see* Multi-Language Integration)
  - regular expressions (*see* Regular expressions)
  - welcome.php program, 72–73
- Web browser, 61
- Wireless Abstraction Library (WALL), 48–50
- Wireless Universal Resource FiLe (WURFL)
  - ajax, css and image format, 42
  - audio and device capability, 43–44
  - available capability groups, 39–40
  - color-coded device capability, 41–42

- device stack output, user agent, 37–38
- getCapabilitiesNameForGroupmethod, 40–41
- output, emulated iPhone 4, 38–39
- server user agent modification, 38–39
- setup, 36–37
- tera-WURFL
  - code, user agent identification, 45
  - display and sound format, 46
  - image output, 47–48
  - setup, 44–45
- XML file, 36
- ZIP archived version, 36
- Word searches, 177

## ■ X, Y, Z

- XML. *See* Extensible Markup Language
- XMLHttpRequest object, 359
  - Ajax grabbed plain text, 363
  - Ajax grabbed specific XML, 365
  - basic XMLHttpRequest, 360
  - creation, JavaScript, 359–360
  - HTML grabbing, 366–367
  - page element modification, 361
  - XML file
    - animals.xml*, 363
    - contents grabbing, 362
  - output and parsing, 364