Using PHP with MySQL

PHP was created in 1995 as *Personal Home Page/Forms Interpreter* or PHP/FI. Within two years it morphed into something very like its present form. 75% of the world's websites use it. We're free to interpret its acronymic name as we please, say, *Pretty good Hypertext Processor*.

An old maxim says it takes 600 hours to learn your first programming language, just 100 to learn the next one. PHP is spectacularly quicker. Spanking new programmers have been known to get PHP web pages up in a few hours. Experienced programmers often get their first PHP web pages up in an hour or less.

This ultimately utilitarian scripting language has three main uses:

- *server-side scripting* requires the PHP parser, a web server and a web browser, is ideal for rapid generation of dynamic web pages, and is the focus of this chapter;
- *command-line scripting* requires only the PHP parser, and is ideal for scripts that run under Linux/Unix *cron* or the Windows Task Scheduler;
- *client-side GUI applications*, only with a very good knowledge of PHP and a GUI add-in tool like *GTK*.

The language is so flexible, a hobby industry of 'inappropriate' PHP uses has grown up; for example a Swedish programmer wrote a TCP/IP stack and web server in PHP. Such flexibility has its dark side: function call interfaces can be inconsistent, and language documentation can be incomplete. If that bothers you, get ready to be bothered.

PHP versions 4 and 5 support many RDBMSs including MySQL. Soon after PHP's first release, web developers discovered that generating HTML from PHP and database data is quick and efficient. A PHP-enabled server makes no demands on the client beyond ordinary HTML rendering. Nothing analogous to a Java engine or Javascript processor is required. Nearly all web servers run PHP. The combination of Linux, Apache, MySQL and PHP (*LAMP*) is a standard. So is *WAMP*: Windows, Apache, MySQL and PHP.

Whether a machine is running Linux/UNIX or Windows, if it is to serve up web pages containing embedded PHP scripts, the PHP interpreter must run inside the machine's web server. In this chapter we describe how to use PHP with MySQL under the Apache 2 web server in Linux and Windows, and under IIS in Windows.

Installation

The simplest way to set up a web server where MySQL databases can drive PHP and Perl scripts under Apache is to download *XAMPP* for your operating system and follow the instructions. It also installs *phpMyAdmin*. If this works for you, skip now to *PHP Basics*.

Installing Apache 2 and PHP under Linux

Installing Apache

Apache 2 is the only Apache version to consider for most purposes. Download the latest stable release from *http://httpd.apache.org/download.cgi*. which also has up-to-date instructions on how to verify the download. Move the file to */usr/local/src/lamp*, unpack it and make it, for example for Apache version 2.0.49 …

```
tar xsf httpd-2.0.49.tar.gz
cd /usr/local/src/lamp/httpd-2.0.49/
./configure --prefix=/usr/local/apache2/2.0.49 \
--enable-modules=all \
--enable-so
make
make install
ln -s /usr/local/apache2/2.0.49 /usr/local/apache2/current
```

To have Apache 2 start whenever the server boots, do this:

```
\cp /usr/local/apache2/current/bin/apachectl \
/etc/rc.d/init.d/httpd2
cd /etc/rc.d/rc3.d
ln -s ../init.d/httpd2 S20httpd2
ln -s ../init.d/httpd2 K20httpd2
```

Installing PHP

If your *Nix installation does not include PHP, download a recent stable version of PHP from *www.php.net/downloads.php*. As of PHP 5.4, the default package is the MySQL Native Driver *mysqlnd*, which includes the *mysql*, *mysqli* and *PDO* APIs. Unpack and build—it sounds simple enough, but combinations of *Nix, Apache and PHP yield dozens of variations, change with major releases of PHP and MySQL, and need individualised build and configuration sequences. Some information on this can be found at *www. php.net/manual/en/ install.unix.php*, but the INSTALL file in the download package is more useful; study it to choose the flavour that matches your *Nix and Apache installations, then follow the matching step-by-step.

Configuring Apache and PHP

Apache looks for Web pages to serve to clients in `DocumentRoot`. To change what `DocumentRoot` points to, use a text editor to edit it in *http.conf*, for example

```
DocumentRoot /usr/web
```

then set permissions in the Apache *htdocs* directory, for example:

```
chown -R nobody:nobody /usr/web/
```

In a text editor, look for `AddType` in */usr/local/apache2/current/conf/httpd.conf*, and add:

```
AddType application/x-httpd-php php
AddType application/x-httpd-php-source phps
```

Next, look for the `LoadModule` commands in *httpd.conf*, and add:

```
LoadModule php4_module modules/libphp4.so
```

If Perl is already installed, as it likely is, a `LoadModule` command for Perl will already exist. If Perl is not yet installed, it very likely soon will be, so add:

```
# LoadModule perl_module modules/mod_perl.so
```

which you can uncomment as soon as Perl is available.

Now look for a line beginning with `DirectoryIndex`, and ensure that it includes:

```
DirectoryIndex index.html index.html.var index.php
```

Start Apache:

```
/etc/rc.d/init.d/httpd2 start
```

Visit `http://localhost` with your browser and you should see the Apache home page. Now you can delete the sample files in */usr/local/apache2/current/htdocs/*. If you have an *index.html* to copy there, do so, or use your text editor to create a barebones *index.html*:

```
<html>
<head><title>default server page</title></head>
<body>Default Server Page</body>
</html>
```

To test PHP, use your text editor to make */usr/local/apache2/current/htdocs/phptest.php*:

```
<?php phpinfo(); ?>
```

and call it in your browser as `http://localhost/phpinfo.php`. It should display a long PHP status page giving a full readout of the PHP modules you have installed.

Installing Apache 2 and PHP under Windows

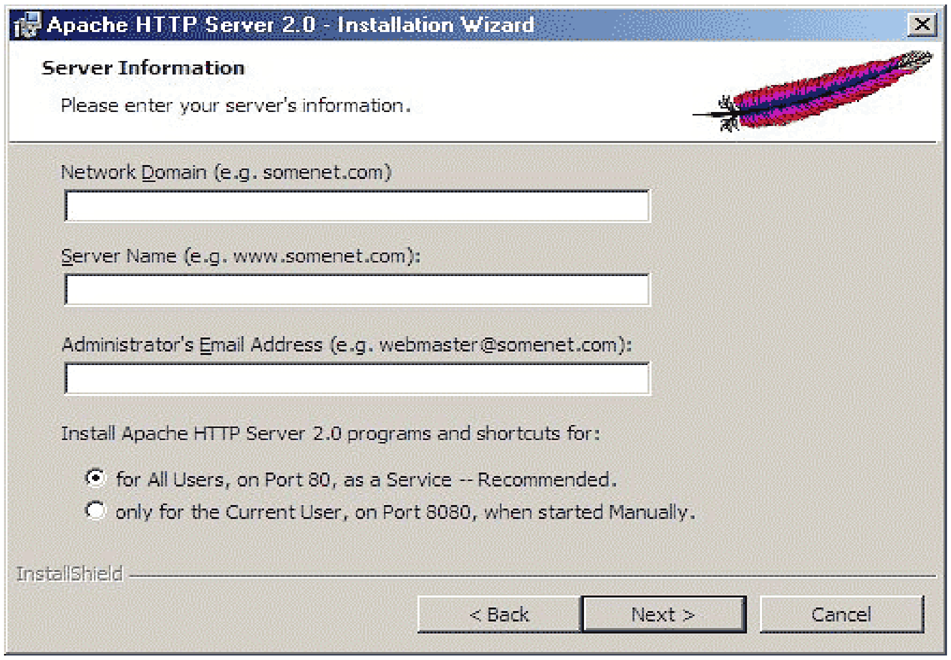
For a step-by-step installation guide with minimal verbiage see *here*. Apache 2 for Windows is reliable, full-featured and free, so it is the Windows web server of choice. Since version 4.3, PHP does not run on Windows 95, and no PHP version runs as a service under Windows 98, so you need Windows 2000 Professional or later.

Installing Apache

If you have Visual C++ 5.0 or later, or Microsoft Visual Studio, you can build Apache from source, but under Windows that may be the worst way to get started with Apache. The *build process under Windows* is complex even inside the Visual Studio IDE, and hair-raising from the command line. So, unless you just plain enjoy creating and debugging makefiles, you are better off getting *a Windows Installer binary distribution*, also known as an *.msi* file. By the time you know enough about Apache to know what customisations you need, you may be ready for the Apache build process.

If an earlier Apache version is running, bring it down and uninstall it before installing a new copy of Apache. Multiple versions of Apache 2 cannot coexist. Double-click the

Apache 2 *.msi* file. After agreeing to the licence, you will see a dialog box (*Fig 12-1*). Set Network Domain and Server Name to something appropriate; `localhost` works perfectly well, provided you are not installing Apache onto a remote computer. Enter your email address in Administrator's Email Address and select Port 80/service.


Apache HTTP Server 2.0 - Installation Wizard

Server Information

Please enter your server's information.

Network Domain (e.g. somenet.com)

Server Name (e.g. www.somenet.com):

Administrator's Email Address (e.g. webmaster@somenet.com):

Install Apache HTTP Server 2.0 programs and shortcuts for:

[✓ for All Users, on Port 80, as a Service -- Recommended.]

only for the Current User, on Port 8080, when started Manually.

Fig 12-1: Apache 2 Windows Installation Wizard

In the next dialog (not shown), change the default installation directory from *C:\Program Files\Apache Group* to *C:\Apache*, or something else that fits the 8.3 filename format, so you never have to write quotes around your Apache path. Here we'll refer to your choice as [APACHEDIR]. Typical installation will then provide a no-hassles working installation.

Soon the Installation Wizard reports that Apache 2 is installed. The only thing left to do is test it by opening the browser and typing `http://localhost` into your browser's Locator field, whereupon you should see the default Apache 2 webpage. Once this much is done, delete the sample files in *c:\Apache\Apache2\htdocs*. If you have an *index.html* to copy there, do so, or else create a barebones *index.html*:

```
<html>
<head><title>default server page</title></head>
<body>Default Server Page</body>
</html>
```

Preparing Apache 2 for PHP

The Apache configuration file is *[APACHEDIR]\Apache2\Conf\httpd.conf*. Edit it with any text editor, for example *TextPad*. To change the default `DocumentRoot` folder, where Apache looks for Web files to serve, set `DocumentRoot` in *httpd.conf*, for example

```
DocumentRoot "c:/web/htdocs"
```

There are two more things to do—tweak the Apache configuration file for PHP, and set up for error logging. Within your text editor, search for `DirectoryIndex`:

```
DirectoryIndex index.html index.html.var # index.php
```

To have Apache serve PHP pages, remove the comment character (#). You should also permit the use of *.htaccess* files in any directory: search for an `AllowOverride` setting and change it from `None` to `All`. Save the changes made so far, and leave the file open in the text editor—more changes are needed once PHP is installed.

If a setting triggers an error, the error *may* log to *[APACHEDIR]\apache2\logs\error.log*—but more likely will log only to the *Windows Event log* (Start | Settings | Control Panel | Administrative Tools | Event Viewer), which is not a convenient tool for debugging installation settings. It is much easier to test Apache 2 in a command line window, where error reports appear instantaneously. So open a command line window, navigate to *C:\Apache\Apache2\bin*, and make sure you can start and stop *Apache.exe* from there:

```
apache -k start
apache -k stop
```

Installing PHP

PHP binaries are no picnic to install in Windows, especially since the rules change from version to version, and often have been documented inaccurately. Variations abound:

First, Windows can run PHP programs in two main ways: via a CGI interface using *php.exe*, or internally *and faster* using PHP DLLs. So for many PHP releases there have been two Windows binaries, a smaller *.msi* package which installs the CGI executable *php.exe* and is missing the Apache DLLs (so you have to download them from *http://snaps.php.net*), and a more complete *.zip* package which includes those DLLs.

Second, installation under Apache differs greatly from installation under IIS.

Third, MySQL may be on the machine where PHP is, or somehere else on the LAN.

Fourth, there are three main PHP APIs for MySQL: *mysql*, *mysqli* (*i*="improved") and *PDO*. The main *mysqli* improvements over *mysql* are object orientation and multiple stored procedure calls, but *mysqli* O-O syntax is different and pickier, so refactoring code for it can be painful. *PDO* also supports O-O syntax, but its main draws are support for a dozen different RDBMSs, and named parameters.

Fifth, in late 2005 MySQL began releasing its own builds of *libmysql.dll*, *php_mysql.dll* and *php_mysqli.dll* as *Connector/PHP*. It strongly recommended Connector/PHP over the DLLs shipping with PHP. With PHP 5.3, all this suddenly stopped; *libmysql.dll* was replaced by the MySQL Native Driver *mysqlnd*, which includes the *mysql*, *mysqli* and *PDO* APIs. This became the default in PHP 5.4.

To be on the safe side, before proceeding *remove all traces of previous PHP installations*, including *php*.** from *%windir%* and *%windir%\system32*. Then:

For Apache: Take the package you want from *www.php.net/downloads.php* and unpack it to *%HOMEDRIVE%\php*, preserving paths.

For IIS: For PHP 4.4, download and run *www.iis-aid.com/iis_aid_php_installer*. For PHP 5, take the latest *zip package* and thread-safe *PECL binaries package* (PHP extensions) from *www.php.net/downloads.php*. Create a PHP installation folder, for example *c:\php*. Unpack the zip package there, preserving paths, then unpack the PECL package into the newly created PHP subdirectory *ext*, again preserving paths.

Here we call your PHP installation folder [PHPDIR]. The PHP manual is available for download in many languages and formats at *www.php.net/docs.php*.

Configuring Apache 2 to run PHP 4 or 5 with MySQL

1. Set PHP path: In My Computer | Properties | Advanced, ensure that [PHPDIR] is in the PATH. If you are installing PHP 4, move all DLLs from both *[PHPDIR]\dlls* and *[PHPDIR]\sapi* to *[PHPDIR]*. For PHP 5, that's not necessary—the DLLs are already in [PHPDIR].

2. Configure Apache 2 for PHP: In the text editor window where *Httpd.conf* is open, look for the section that contains a list of `AddType` commands, and add this one:

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

For PHP 4, in the `LoadModule` section of *Httpd.conf*, uncomment or add in:

```
LoadModule php4_module "c:/php/php4apache2.dll"
```

For PHP 5 it should be:

```
LoadModule php5_module "c:/php/php5apache2.dll"
```

3. Set up php.ini: Edit a copy of *[PHPDIR]\php.ini-recommended* as *php.ini* in your text editor, adjust `doc_root` and `session.save_path` if need be, pick a `timezone` from *www.php.net/manual/en/timezones.php*, and comment out one `extension_dir` setting:

```
doc_root = c:\apache\apache2\htdocs    ; MUST BE THE SAME AS IN Httpd.conf
session.save_path = c:/temp            ; DIRECTORY MUST EXIST
date.timezone = "America/Vancouver"
extension_dir = "c:\php\extensions"    ; FOR PHP 4 ONLY
extension_dir = "c:\php\ext"           ; FOR PHP 5 ONLY
```

Create *c:\temp* if it does not exist. The PHP manual says these path arguments need trailing backslashes, but since PHP 4.3 that is not the case.

4. **Apache 2, PHP and MySQL**: Up though PHP 5.2, if there is a copy of *libmysql.dll* in *[PHPDIR]*, copy it to *[APACHEDIR]\bin*, otherwise copy it from the MySQL *bin* folder; for PHP 5.3 and later, remove *libmysql.dll* from *[APACHEDIR]\ bin*. Ensure that *php_mysql.dll* and *php_mysqli.dll* are in `extension_dir` as set in Step 3. Uncomment:

```
extension=php_mysql.dll
extension=php_mysqli.dll
extension=php_pdo_mysql.dll
```

The default MySQL host is `localhost`. If MySQL is on another LAN machine, set the host:

```
mysql.default_host = "networkmysqlservername"
```

5. CGI mode: If you need to run PHP programs in CGI mode, add to *Httpd.conf*:

```
ScriptAlias /php/ "c:/php/"
Action application/x-httpd-php "/php/php.exe"
```

6. Test: Save all changes *and reboot the machine*. To test Apache, execute …

```
apache -k start
```

in [APACHEDIR]*apache2\bin*. If there is an error, Apache will report it immediately in this window. To ensure that Apache is serving PHP pages correctly, use your text editor to create a simple PHP page, called *phptest.php*, containing just this one line:

```
<? php phpinfo(); ?>
```

Save to the Apache `DocumentRoot` folder as *phptest.php*. To run it, browse `http://localhost/phptest.php`. If everything is working correctly, you will see a page with the PHP logo and a long list of settings. The environment variable `orig_script_name` is set to `/Php/Php.exe` if running through CGI, or to the script filename if Apache is running the PHP script directly. In either case, if PHP can connect to MySQL then the page shows a MySQL section.

Installing PHP to work with MySQL under IIS

For the latest on this subject click *here*. IIS is in *%HOMEDRIVE%\Inetpub*. If you downloaded and ran the installer from iis-aid.com, skip to Step 3. Otherwise …

1. Set environment variables: In My Computer | Properties | Advanced | Environment Variables | System variables, scroll down to `path`, click Edit, add the `[PHPDIR]` spec and click OK. Under System variables click New, add a variable called `PHPRC`, and set its content to your `[PHPDIR]` spec.

2. Install PHP application mapping: On Windows Server 2008 follow *this walk-through*, on Windows 7 follow *this one*, for earlier versions take David Wang's Visual Basic *script*, save it as *chglist.vbs*, and run this command:

```
chglist.vbs W3SVC/ScriptMaps "" ".php,[PHPDIR]\php5isapi.dll,5" /INSERT /COMMIT
```

substituting for `[PHPDIR]` your PHP installation path. In My Computer | Manage | Services and Applications | Internet Information Service | Web Sites | Properties | Home Directory | Configuration, use Add or Edit to ensure that there is an entry for which Extension is `.php` and Executable Path is `[HPDIR]\ext\php5isapi.dll` (eg *c:\php\ext\php5isapi.dll*).

For IIS 6.0, you must also configure a Web extension. In My Computer | Manage | Services and Applications | Internet Information Service | SERVER | Web Service Extensions, click on Add a new Web service extension, enter a name for Extension Name, set extension Status to Allowed, and in the Add File applet click on Browse, navigate to [PHPDIR], and select `php5isapi.dll`.

3. Set up php.ini to talk to MySQL: In [PHPDIR] make a copy of *php.ini-recommended*, name it *php.ini*, open it in a text editor, and touch up these entries:

```
extension_dir="c:\php\ext"   ; or whatever yours is
extension=php_mysql.dll
extension=php_mysqli.dll
extension=php_pdo_mysql.dll
```

The default MySQL host is `localhost`. If MySQL is on another machine on your LAN, set the default MySQL host:

```
mysql.default_host = "networkmysqlservername"
```

Ensure that *php_mysql.dll* and *php_mysqli.dll* are in `extension_dir` as set in Step 3.

4. Test: Reboot, or in My Computer | Manage | Services and Applications | Internet Information Service | All Tasks, click Restart. Write *Inetpub\wwwroot\test.php* that just says

```
<?php phpinfo(); ?>
```

then browse *http://localhost/test.php* and scroll down the page to see the MySQL entries.

phpMyAdmin

Once PHP is running correctly, navigate to *www.phpmyadmin.net/home_page/* and download the latest stable release of *phpMyAdmin* for your system. Make a *phpmyadmin* folder off *htdocs* (or *wwwroot* if you are running Windows/IIS), and unpack the archive there. Read *phpmyadmin/documentation.html*, then run *phpmyadmin/scripts/setup.php*. If your *phpMyAdmin* version is 2.11 or later—when *phpMyAdmin* introduced support for stored routines and views—`http://localhost/phpMyAdmin/index.php` has just become a very useful portal for administration of your MySQL server.

Now let's see how to write other such tools ourselves, in wondrously little time.

PHP basics

PHP web scripts run within the PHP parser, which in turn runs within a web server—often Apache or IIS. The PHP script itself remains invisible to the end-user, who sees only the images, PDF files, or HTML output created by the PHP script. A browser's *View Source* command shows not the PHP code, but the web code it has generated.

Much PHP syntax is borrowed from C, for example conditional statements, loop structures, string math and boolean operators, in-line variable assignments, and the statement terminator. But there is a crucial difference between PHP and C: *pace* Niklaus Wirth (who invented Pascal) and Dennis Ritchie (who invented C), *PHP has no variable declarations and no strict variable typing*. The data type of a variable is just the type of data stored in it—*integer*, *double*, *string*, *boolean*, *array*, *resource* or *object*—and can be changed by a new assignment. The value itself can be cast to another data type when it is assigned to another variable.

There is no date or time type, but there is a predefined DateTime class. To create a non-object variable, just assign it a value. Names of predefined variables and of those you create always begin with `$`. If the variable is predefined or created in the current session, PHP accepts the value assigned to it, otherwise PHP creates it and assigns the specified value. At execution time, the parser deduces from the context how the value is to be interpreted (for example as a string to be written to HTML, or as a numerical value to be computed).

A variable declared within a function is *local* to the function unless it is declared with the keyword `GLOBAL`, in which case it refers to the global variable of the declared name. Variables declared outside functions are *global* to the script. Any global variable `$name` can be referred to in any context as `$GLOBALS['name']`.

Strings: Current versions of PHP have *98 string functions*. In double-quoted strings—but *not* in single-quoted strings—PHP expands references to PHP variables; that's tremendously convenient for building dynamic query strings and for keeping the user informed. Concatenate strings with a period, *not* with the arithmetic addition (+) operator. There is an older, now-deprecated *POSIX-style regular expression module*, and a *Perl-compatible* REGEX module.

Arrays: Arrays can be created, and their elements accessed, by numeric index, for example `$data[0]`, or by hashed string key, for example `$_SESSION['db']`. The call

```
$ra = range( 1, 10);
```

creates an array with elements indexed from 1 to 10. Often string keys are more convenient and maintainable, for example after

```
$ra = array('Code'=>'FG','Description'=>'Frammelgrommit','MadeBy'=>'ABC Corp');
```

`$ra['Code']` returns `'FG'`. Keep in mind that string and numeric keys are independent of one another, so for example `$foo[0]` does *not* refer to the first element of an array created by the hash assignment `$foo['bar']= 'trythis'`. Arrays may be recursive, ragged, even one-based.

PHP has these predefined *superglobal* array variables:

- `$GLOBALS[]`: Contains a reference to every variable available within the script's global scope. The keys are the names of the global variables.
- `$_COOKIE[]`: Variables provided to the script via HTTP cookies.
- `$_ENV[]`: Variables provided to the script by the environment.
- `$_FILES[]`: Variables provided to the script via HTTP post file uploads.
- `$_GET[]`: Variables provided to the script via `HTTP GET`, usually submitted from HTML forms (which might be coded in PHP).
- `$_POST[]`: Variables provided to the script via `HTTP POST`, usually submitted from HTML forms (which might be coded in PHP).
- `$_REQUEST[]`: Variables provided to the script via any user input. If the script is running in a web browser, this array includes '`argc`' and '`argv`' keys, but when PHP runs from the command line, these keys can be found in `$_SERVER[]`.
- `$_SERVER[]`: Variables set by the web server or otherwise directly related to the script's execution environment.
- `$_SESSION[]`: Variables registered to a script's session. This is the only predefined array that accepts value assignments which persist across multiple PHP scripts in one session. For this to work, the scripts have to call the PHP function `session_start()`. More about this *shortly*.

There is a rich set of *array functions*, so it is easy to implement array-based hashes, stacks, queues and linked lists. So too for *math functions*, *statistical functions*, *datetime functions and methods*, and *file/directory functions*.

Thus PHP has a rich set of easy-to-use tools for handling data, especially text and lists, on web pages. There are also functions to encode URLs and connect to remote servers, and libraries for widely used databases, including of course MySQL.

Embed PHP code in HTML by demarcating it with these tags:

```
<?php
… // PHP code between standard PHP tags
?>
```

You can also use `<SCRIPT LANGUAGE="php"> … </SCRIPT>`. In *php.ini* you can turn on recognition of short tags and ASP-style tags with

```
short_open_tag=on   // <? … ?>
asp_tags=on         // <% … %>
```

but there is not a guarantee that the server will comply, so best practice is to stick with `<?php … ?>`. Whatever code the web server finds enclosed by `<?php … ?>`, it sends to the PHP parser. Here is a PHP implementation of Hello World:

```
<html><head><title>Hello World from PHP</title></head>
<body>
<?php
echo "<h1>Hello World from PHP!</h1>";
?>
</body></html>
```

That PHP code simply generates HTML for a static greeting. If you save this to the Apache *htdocs* location as *hello.php*, browsing `htpp://localhost/hello.php` will bring up the resulting HTML page. If you right-click on the web page in your browser, then select *View Source* from the context menu, you will see the HTML source code generated from your script by the PHP parser.

Like C, all PHP statements terminate with a semi-colon. If you've written some C or C++, you'll be right at home with PHP's logical operators, for example `||` for OR, `&&` for AND; otherwise they may take a little getting used to.

The simple and popular PHP API for MySQL, *mysql*, has been deprecated in favour of the more robust and secure *mysqli* API, so we have revised this chapter to use *mysqli*.

With the *mysqli* API, there is another choice to be made—procedural vs object-oriented code. The O-O approach can simplify big, complex projects, but it requires a very different way of thinking than you may be used to, and since our goal here is not to inculcate O-O habits but rather to smooth your path into PHP, procedural code is the more accessible starting point.

Available classes and functions are described in detail *here*. For this chapter we list the *mysqli* API functions you are likely to need in Table 12-1, where `$con` is a connection returned by `mysqli_connect()`, `$usr` and `$pwd` are MySQL usernames and passwords, `$db` is a database name, and `$res` is a result object returned by a query call.

Table 12-1: Commonly used PHP 5 mysqli API functions

Type	***Function***	***Action***
`int`	`mysqli_affected_rows($con)`	Return no. of rows changed by a query
`bool`	`mysqli_autocommit($con,$mode)`	Turn autocommit mode on or off
`bool`	`mysqli_change_user ($con,$usr,$pwd,$db)`	Change user of referenced connection
`string`	`mysqli_character_set_name($con)`	Return character set of `$con`
`bool`	`mysqli_close($con)`	Close referenced connection
`bool`	`mysqli_commit($con[,$flags[,$name]])`	Commit current transaction
`mysqli`	`mysqli_connect ($host,$usr,$pwd[,$db[,$socket]])`	Connect to a database
`int`	`mysqli_connect_errno($con)`	Return error no. from last connect call

Type	*Function*	*Action*
`string`	`mysqli_connect_error($con)`	Return error msg from last connect call
`bool`	`mysqli_data_seek($con,$offset)`	Seek to $offset in resultset
`bool`	`mysqli_dump_debug_info($con)`	Write debug info to server log
`int`	`mysqli_errno($con)`	Return no. of last mysqli error
`int`	`mysqli_error($con)`	Return message for last mysqli error
`mixed`	`mysqli_fetch_array ($res,$resulttype)`	Return query, resulttype= MYSQLI_ASSOC| MYSQLI_NUM|MYSQLI_BOTH (default)
	`mysqli_fetch_assoc($con)`	Return resultrow as associative array
`object`	`mysqli_fetch_field($res)`	Return info about resultset column
`array`	`mysqli_fetch_fields($res)`	Return array of resultset column info
	`mysqli_fetch_object ($res[,$class[,$params)`	Return current resultset object
`mixed`	`mysqli_fetch_row($res)`	Return next row in resultset, null if none
`int`	`mysqli_field_count($con)`	Return column count of last query
`bool`	`mysqli_field_seek($res,int $col)`	Set result cursor to this column
	`mysqli_free_result()`	Free memory occupied by resultset
`object`	`mysqli_get_charset($con)`	Return *charset object* for connection
`string`	`mysqli_get_client_info($con)`	Return client library version
`string`	`mysqli_get_client_stats($con)`	Return client per-process stats
`string`	`mysqli_get_client_version($con)`	Return client library version as integer
`array`	`mysqli_get_connection_stats($con)`	Return connection statistics
`string`	`mysqli_get_host_info($con)`	Return connection host and protocol
`int`	`mysqli_get_proto_info($con)`	Return MySQL protocol version of connection
`string`	`mysqli_get_server_info($con)`	Return version of connected server
`int`	`mysqli_get_server_version($con)`	Return version no. of connected server
`mysqli`	`mysqli_init()`	Prepare for mysqli_real_connect call
`string`	`mysqli_info($con)`	Return info about last query result
`mixed`	`mysqli_insert_id($con)`	Return auto_increment ID of row last inserted on this connection
`bool`	`mysqli_kill($con,$processid)`	Kill MySQL thread identified by mysqli_thread_id().
`bool`	`mysqli_more_results($con)`	Are there more results from mysqli_multi_query call?
`bool`	`mysqli_multi_query($con,$qrystr)`	Execute semicolon-delimited queries
`bool`	`mysqli_next_result($con)`	Prepare next mysqli_multi_query result
`int`	`mysqli_num_fields($res)`	Return no. of columns in result set
`int`	`mysqli_num_rows($res)`	Return no. of rows in a result set
`bool`	`mysqli_options($con,$option,$value)`	Set connection *options*
`bool`	`mysqli_ping($con)`	Ping connection

Type	*Function*	*Action*
mysqli_stmt	mysqli_prepare(*con,$qrystr)	Prepare a query for execution
mixed	mysqli_query($con,$qrystr[,$mode])	Execute a query, return a result. Mode= MYSQLI_{USE|STORE}_RESULT. Returns FALSE on failure.
bool	mysqli_real_connect ($con[,$host[,$usr[,$pwd[,$db[,$port [,$socket[,$flags]]]]]]])	Connect to MySQL server via a conneciton object prepared by mysqli_init().
string	mysqli_real_escape_string($con,$str)	Escape characters to make string suitable for query. First call mysqli_set_charset().
bool	mysqli_real_query($con,$qrystr)	Execute query for retrieval by mysqli_store_result, mysqli_next_result
int	mysqli_refresh($con,*$options*)	Flush tables/caches,reset replication
bool	mysqli_rollback ($con[,$flags[,$name]])	Roll back current teransaction
bool	mysqli_select_db($con,$db)	Select default database
bool	mysqli_set_charset ($con,$charsetname)	Set *character set*.
	mysqli_set_local_infile_default ($con)	Deactivate handler that was set with mysqli_set_local_infile_handler().
bool	mysqli_set_local_infile_handler ($con, $callable_read_func)	Set callback function for LOAD DATA INFILE command
string	mysqli_sqlstate($con)	Return SQLSTATE error of last query
bool	mysqli_ssl_set ($con,$key,$cert,$ca, $capath,$cipher)	Establish SSL connection, 5.3.3 or later
string	mysqli_stat()	Return uptime, threads, queries, open tables, flush tables, queries/sec
mysqli_stmt	mysqli_stmt_init($con)	Initialise statement object
mysqli_result	mysqli_store($con)	Transfer last query result for use by mysqli_data_seek().
int	mysqli_thread_id($con)	Return current thread ID
bool	mysqli_thread_safe($con)	Is client library thread-safe?
mysqli_result	mysqli_use_result($con)	Initiate result retrieval from last mysqli_real_query() call
int	mysqli_warning_count($con)	Return no. of warnings from last query

TOC ***Previous*** ***Next***