

# Set up a PHP and MySQL development environment

Skill Level: Introductory

[Brett McLaughlin](#)

Author and editor

O'Reilly and Associates

08 Jun 2005

This tutorial demonstrates how to set up the Apache Web server, the PHP interpreter, and a MySQL database as a development environment on a Windows XP machine.

## Section 1. Before you start

### About this tutorial

This tutorial walks you through the process of setting up your Windows XP machine as a PHP and MySQL development environment. You will start by downloading and installing the Apache Web server for handling HTTP requests (the type of request that you make every time you type a URL into a browser). You'll also learn how to configure the Apache server, so you can store your Web site documents anywhere you like. You'll even set up the default index files that Apache uses.

With a Web server setup, you'll then download and install the PHP scripting language. Beyond a simple download, you will have to configure Apache to pass requests through to the PHP engine. As with Apache, you will also learn how to locate your scripting files correctly, and secure them as well.

Once you have a Web server and a PHP scripting engine running, all that's left is adding a database. Typically, MySQL is the popular choice of database as it is cost-free and open source -- that is what I use in this tutorial. An alternative is IBM

Cloudscape, which is also a standards-driven relational database written in Java. It's designed for embedding in the many Java apps and servers that developers use today. Here, you'll learn how to install MySQL on the Windows platform and connect your PHP scripts to a MySQL database.

By the time you're done, you'll have a complete development environment, ready for testing and development.

This tutorial is written for Web developers. If you're comfortable working with PHP and databases, then this tutorial will get you up and running on Windows XP (whether or not it's connected to the Internet all the time). If you're just getting started with Web programming, though, this tutorial will provide a virtual playground for trying out PHP. Whatever your level of expertise, as long as you are willing to take things step by step, this tutorial will get you ready to go on Windows.

You will also do some general Windows configuration, including setting up your environment. If you're familiar with PATH and other Windows XP environment variables, you're already set; if this is new to you, you'll get a brief introduction in this tutorial.

## System requirements

All you need in terms of software and hardware is a machine running Windows XP (I use SP2, but I've run this same setup without SP2 for years). I'll walk you through all the downloading and installation, so you don't need any pre-existing software. You'll also need administrator access on your machine, so you might have trouble if you're working on a shared machine. Finally, you'll need Internet access during this tutorial to download software. Once you have the environment set up, however, you *won't* need Internet access.

---

## Section 2. Develop, test, deploy

### What is a development environment?

If you've ever done any serious development work, you're probably at least somewhat familiar with the term *development environment*. Depending on how sophisticated your company is, this might mean anything from a Sparc 10 high-powered UNIX box running 10 or 15 instances of Apache to a laptop with an IDE (integrated development environment). In fact, you'd probably get a different

definition of the term for every person you asked. If you haven't heard the term, you're probably better off -- you don't have all the misconceptions that many developers do. For the purposes of this tutorial, though, I propose a simple definition: A development environment is simply the environment you develop on. I know that sounds trite -- and that I'm not supposed to use the term I'm defining in its definition -- but it's actually quite useful.

First, I assume that you are a developer. You write some code, or you want to write some code, or you're being paid to write some code...in any case, you're coding. And, unless you work in binary on an abacus, you work on a computer. That computer, then, is your development environment. See, I told you this was simple!

More importantly, though, your development environment is *only* your computer. It's not, for example, a testing server you load your code onto to see if it works. It's not a server farm in a remote country that your entire company uses for super-secret deployments. It's not even the secondary machine you have in case your main computer dies. It's just the computer you write the code on.

The point here is that anytime you move code -- to another computer, to another operating system, to another zip code -- you change something. Maybe a patch is applied on your machine, but not on the target machine. Maybe one machine runs Apache and another runs iPlanet. Whatever the situation, two computers are never exactly alike. And when I talk about a development environment, I don't want to deal with changes. So it's simply the machine you work that acts, well, like it acts, and is not like some *other* machine, somewhere else.

## Think you don't need to do any testing?

You know it as well as I do: testing is boring. It's a pain; it's slow; it's unexciting. And, it will keep you employed. It's no great secret that developers who test their code are those same people who seem to be water-skiing on the weekends and buying new houses. That's because their code works when it gets deployed (see [What is deployment?](#) for more on deployment), and they're not worried about it breaking.

The secret to testing is in the changes that occur between one computer (your development environment) and another (the machine you want your code to run on). If you can figure out what those changes are, and account for them, life is going to get a lot better. For example, if your development environment is a PC running Windows XP, using Apache, PHP, and MySQL, and you intend to run your code on a Linux server running Apache, PHP, and PostgreSQL, you should watch out for the differences between Windows XP and Linux, and MySQL and PostgreSQL. Since Apache is more or less the same on Linux and Windows -- as is PHP -- you don't need to focus your attention on these applications.

In a perfect world, you have three entirely separate machines for coding:

- *Development environment*: The machine to write the code on is set up for your convenience with your favorite code editors and user preferences.
- *Testing machine*: The machine to test your code on is set up just like the deployment server in every respect. The only difference is that it is a private machine, and preferably separate from any other machines on the network that might interfere with it.
- *Deployment machine*: This is where the code ultimately runs. This machine only has the software it needs to run code and doesn't have user preferences or development tools (in fact, if done right, it won't even allow remote user logins).

Consider yourself lucky indeed if this is your setup. More often than not, you'll be forced to combine at least two of these roles on one machine. For example, you might develop on a PC laptop and test and deploy to a UNIX server. In this case, you might have a testing setup on a different port (like 8080 or 9900) so the outside world won't see your testing code.

For your purposes, don't worry too much about a testing environment until you get your development environment up and running. Just realize that you need to know exactly what software is on your development machine and how it differs from your target deployment server. It will help you isolate those spots you must focus on testing.

## What is deployment?

Once you have working code, you typically have to ensure it not only works on your machine (your development environment) but also on the machine it will run on. Because most of developers don't have servers sitting in their home office, this means taking your code from your development environment, which is now well-tested, and *deploying* it. This might involve FTPing the code to your Internet service provider (ISP), a server farm in a room much colder than the one you're sitting in, or even handing off a CD full of code to a system administrator. Whatever the case, this is the point where you deploy your code to another machine.

Deployment is a topic I won't get into much in this tutorial, but you need to understand that it is something you must deal with in application development. Even if you're just setting up a blog on your personal Web site, you still need to take the code on your development environment and put it somewhere for the world to see. At first glance, this might make the development environment seem less useful. Suddenly, all those changes from computer to computer creep in. However, this is precisely what makes the development environment useful. By the time you deploy, you've already written your code on a local machine and gotten all the software bugs out (HTML is written correctly and looks right on various browsers; PHP scripts are

all executable and secure; MySQL connections work as they should; and passwords are set). You've also tested your code on a machine similar to the deployment target and worked out any kinks in that process. So your rate of success when you actually deploy your code is high -- in fact, it's all often quite boring. You click the Upload, Commence FTP, or Burn buttons and sit back and relax.

Without a development environment, though, this is more like a roll of the dice. You upload code you haven't run (because you haven't set up your development environment correctly) and tested (how can you test something that never worked in the first place?). So how can you expect the code to work properly? Suddenly, you're testing out code on a live server as users, friends, and family get errors and weird messages. This is no way to develop code, so let's frown upon it (all together now) and get back to setting up your development environment.

## Seems a little formal, doesn't it?

If you want to put together a killer Web site and PHP provides you with a fun, comfortable scripting tool to do it, this might all seem like overkill. I understand that, and you certainly don't have to worry about having multiple servers and teams of testers and developers. However, even in a one-man production, you want your site to work when you load it to your ISP. If you keep at least these concepts in mind -- if not the actual details -- then you'll be able to find bugs that emerge when you upload your site. You might even find that you enjoy the process more when you know why those bugs emerge.

---

## Section 3. Set up the Apache Web server

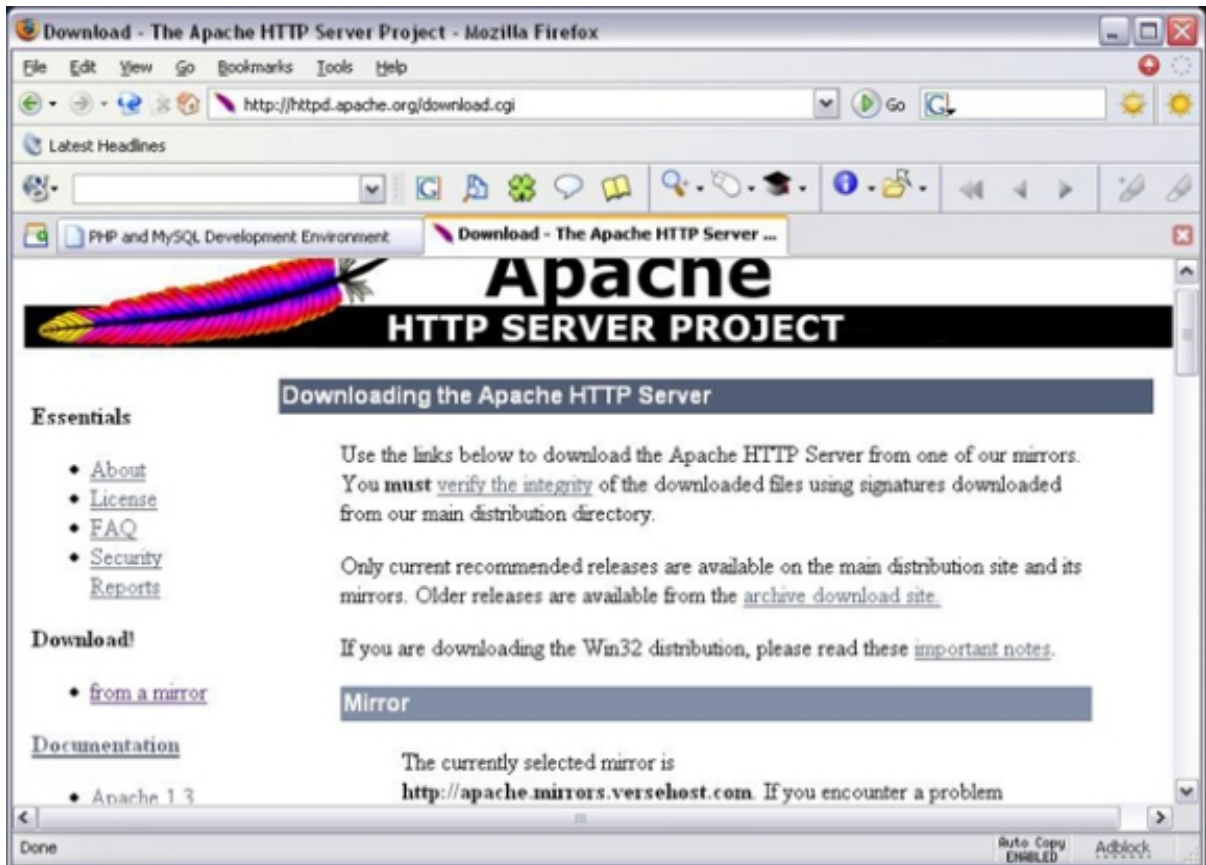
### Download Apache

Before you can do Web development, you need a Web server. And while plenty of commercial options exist, Apache is still the de facto standard when it comes to serving Web content. Furthermore, Apache is easy to configure. There isn't a fancy Web console, but I'll take plain old text files over a GUI any day of the week. Editing a text file is simple, and all you need is Notepad or Wordpad. Plus, Apache is blazing fast, and that's what a Web server should be, right? It should be simple, easy to use, and fast.

To start, go to the [Apache Web site](#). On the left, under Apache Projects, click on the **HTTP Server** link. This takes you to the main Apache [Web server site](#). From here,

click on the **Download from a mirror** link, also along the left side of the page. This takes you to a page like that shown in Figure 1.

**Figure 1. Download page for the Apache Web server**



Scroll down to the **Apache 2.0 section**; it will say something like "Apache 2.0.54 is the best available version." Now click on the link for the **Win32 binary**. As I write this, the filename is `apache_2.0.54-win32-x86-no_ssl.msi`. Save this file to disk and be sure you leave it somewhere you'll remember (there's nothing like going through all of this and then forgetting where you saved it). Since you will only use this file to launch the installation, you might want to place the file on your desktop during installation.

## Install the server

Double-click on the downloaded file to start the Apache installation process. You'll get a nice stylized logo along with a Welcome message to begin. Click on **Next** to go to the next screen.

You'll have to accept the Apache license to go any further. While many folks will tell you to skip right through these screens, you need to stop and read through this

license. Just think -- you can read it once and then skip through it on subsequent installations. For those of you who refuse my license admonitions (I know who you are), the Apache license is about as nonrestrictive as it gets. Basically, you can do whatever you want with the software. Change it, add to it, remove code from it; since you're not working with the source code, though, this doesn't really affect you. Accept the terms and click **Next** again.

The next screen gives you some options (finally). The network domain -- on a personal installation like this -- really doesn't matter that much. You can set it to the domain of your ISP (such as grandnetworks.com), or a domain you'd like to use on your home or office network (such as gilead.net), or anything else that fits the basic Internet format. The server name should be the same thing, preceded by your computer's host name. Again, on a personal Windows XP installation, this really isn't going to matter a lot (you access the installation with the *localhost* moniker, as you'll see shortly). Next, set the administrator's e-mail; I use something like `webmaster@gilead.net`. Finally, decide whether you want to install this for all users or just the current user. I tend to install programs for all users, just in case my wife ever decides to pick up programming as a hobby. (That's a joke, folks. Seriously, you should be laughing.) This option also sets the server up on the default HTTP Web server port, 80, which you want.

If this was a tutorial on setting up Apache on a production server, I'd really spend some time here. Setting up the domain and the server name are critical for ensuring that other machines can access your new server. Most development machines, though, are laptops or desktops running on local networks that provide IP addresses through DHCP (Dynamic Host Configuration Protocol). This means that you'll probably have a different IP address every time your machine starts up, and domain and host names become essentially useless, at least in terms of other machines accessing your Web server. It's just easier to set the values in this step arbitrarily and move on.

For the next step, simply select the **Typical** setup and move on. You'll lose some of the files needed for compiling in Apache modules, but that's probably a little beyond what you're trying to do anyway. The Typical setup will be plenty for your purposes. Next, select the folder to install Apache into; unless you have a good reason to change this, simply accept the default. It makes upgrades and reinstalls a lot easier. Finally, click Install. You'll see several progress indicators, and you might want to go grab a soft drink while you wait. Near the end of the process, you'll see a few command shells pop up (black windows that quickly appear and then disappear). You also might have to let any firewall or Internet security software know that it's OK for Apache to accept connections on incoming ports.

Once all this is done, click **Finish**. Delete the installation file (probably on your desktop, if you followed these instructions), and you're all done with installation. As a sanity check, you should have a little Apache icon running in your system tray now, as Figure 2 shows.

**Figure 2. Apache in the system tray**

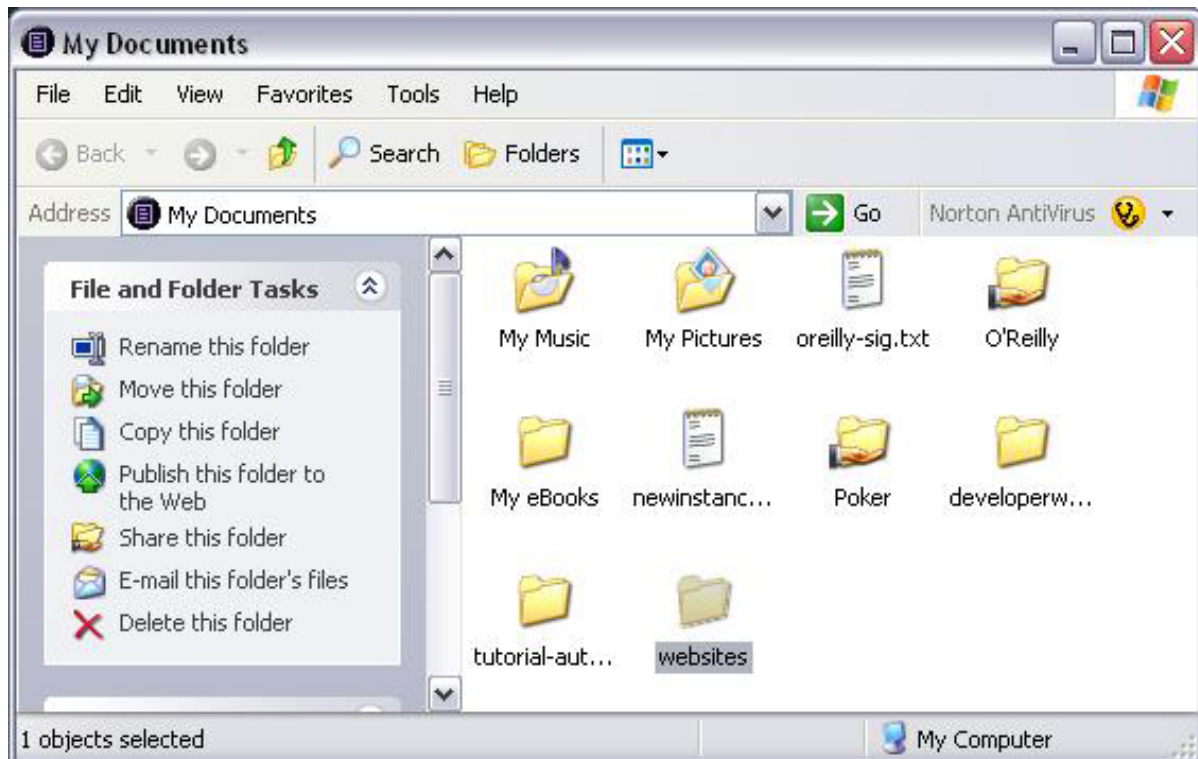
## Set the documents directory

By default, Apache sets up a directory for you to drop in HTML and image files so Web clients can access them. If you installed Apache in its default location, the Web document directory is `C:\Program Files\Apache Group\Apache2\htdocs`. While nothing is specifically wrong with this location, there's very little right about it, either. As an illustration, I typically back up my Windows My Documents directory. I store all my editing, writing, and programming files in that location, which makes backup and recovery a piece of cake. However, this practice would completely miss any Web site files you create for Apache. I suppose you could add `C:\Program Files\Apache Group\Apache2\htdocs` to the directories you back up, but that's a bit annoying. Even worse, when you upgrade Apache, new versions can wipe out or overwrite this directory.

I prefer to tell Apache to look for its Web files in a different location -- unsurprisingly, one that resides in the My Documents directory. This is an easy change to make.

First, create a sub-directory under My Documents; I use *websites* as that's pretty self-descriptive. Check out Figure 3 for a visual aid.

**Figure 3. A websites directory under My Documents**



Now you can tell Apache to look in this alternate location for Web files. Almost all of Apache's configuration is handled through a single text file, `httpd.conf`, located in the `C:\Program Files\Apache Group\Apache2\conf` directory. Open this file; it's long, but I'll walk you through what you need to do.

By default, Windows opens this file in Notepad. Press **Ctrl+F** to open the search dialog (if you're using a different editor, you can use that application's search command). In the search dialog, enter **DocumentRoot**. You'll see an entry that looks something like this:

```
#
# DocumentRoot: The directory out of which you will serve your
# documents. By default, all requests are taken from this directory, but
# symbolic links and aliases may be used to point to other locations.
#
DocumentRoot "C:/Program Files/Apache Group/Apache2/htdocs"
```

`DocumentRoot` is the key that Apache uses to represent its default Web directory. To have Apache look somewhere else, just replace the entry with your own directory, like this:

```
#
# DocumentRoot: The directory out of which you will serve your
# documents. By default, all requests are taken from this directory, but
# symbolic links and aliases may be used to point to other locations.
```

```
#  
DocumentRoot "C:/Documents and Settings/Brett McLaughlin/My Documents/websites"
```

Be sure to leave the path inside the quotes. That allows Apache to handle spaces in long Windows path names.

You also need to make this change in one more place; scroll down a few lines, and you'll see a comment like this:

```
#  
# This should be changed to whatever you set DocumentRoot to.  
#
```

Needless to say, follow the instructions and make this change:

```
#  
# This should be changed to whatever you set DocumentRoot to.  
#  
<Directory "C:/Documents and Settings/Brett McLaughlin/My Documents/websites">
```

You must restart the server before Apache recognizes these changes. Right-click on the Apache icon in your system tray (shown in Figure 2), and click **Open Apache Monitor**. Then along the right side of the monitor, click **Restart**. The status of the server is reflected in the Service Monitor. Now you just need to test things and see that they're working for yourself.

## Test your installation

Before you fire up your Web browser, I must review a concept with you. Bear with me; it won't be too bad, I promise. When you installed Apache, it set up shop on port 80 of your machine. That's great because when you enter a regular URL into your Web browser, the browser automatically tries to connect to port 80 on the machine requested. You can optionally add a port like this: `http://www.oreilly.com:8080`; but since you want to connect to the default port, you don't have to specify this extra information.

What that leaves, though, is the matter of what to type before the port number. Normally, you would enter something like `radar.oreilly.com` or `www.thepokergeek.net`. You probably entered something as a server name earlier -- perhaps `roland.gilead.net` -- but that doesn't mean that typing that server name into a browser will work. You must ensure that the name you type maps to the IP address of your machine. I will avoid diving too deeply into that quagmire and simply tell you a little secret: You can always access the machine you're on with the IP address

127.0.0.1. This is called the *loopback address* and essentially loops your requests back to your machine.

Of course, I hate typing, and 127.0.0.1 is a little long for frequent use. Instead, you can use a shortcut, the name *localhost*. This maps to your loopback address, as the following command output shows:

```
C:\Documents and Settings\Brett McLaughlin>ping localhost
Pinging Cuthbert [127.0.0.1] with 32 bytes of data:

Reply from 127.0.0.1: bytes=32 time<1ms TTL=128
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128

Ping statistics for 127.0.0.1:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 0ms, Maximum = 0ms, Average = 0ms

C:\Documents and Settings\Brett McLaughlin>
```

Notice that the name *localhost* maps to 127.0.0.1. Putting all of this port, IP address, and localhost business together, the address you want to type into your browser address bar is `http://localhost`. This requests a page from your local server on port 80. The result should look something like Figure 4, despite the apparent errors you'll see.

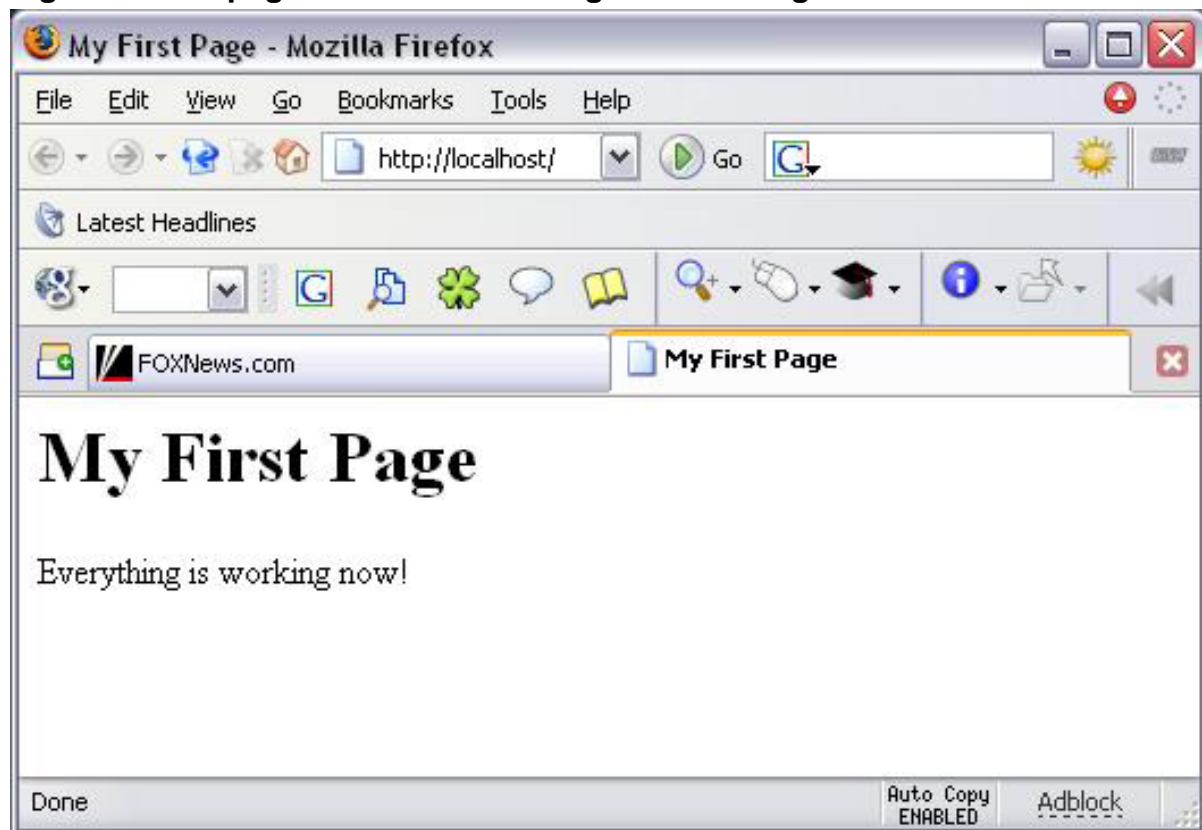
#### Figure 4. Apache reports a 403 Forbidden error



While this isn't a typical Web site, it *is* the desired result. Remember, you created a new directory (C:\Documents and Settings\Brett McLaughlin\My Documents\websites) and directed Apache to look in this directory for Web files. The problem is that there aren't any files in that directory. As a result, Apache reports that it won't show you the empty directory (a security measure). The easiest way to fix this problem is to navigate into this directory and create a simple file:

```
<html>
  <head>
    <title>My First Page</title>
  </head>
  <body>
    <h1>My First Page</h1>
    <p>Everything is working now!</p>
  </body>
</html>
```

Save this file as `index.html` under C:\Documents and Settings\Brett McLaughlin\My Documents\websites. Apache looks for files with this name by default, so now you should be able to reload your browser page (or reenter the URL if you closed your browser). This time, the response is a little more useful, as Figure 5 shows.

**Figure 5. This page indicates that things are working**

At this point, you have Apache installed and ready for use. Go have a snack and come back when you're rested -- you move on to PHP next.

---

## Section 4. Install the PHP interpreter

### Download PHP 5

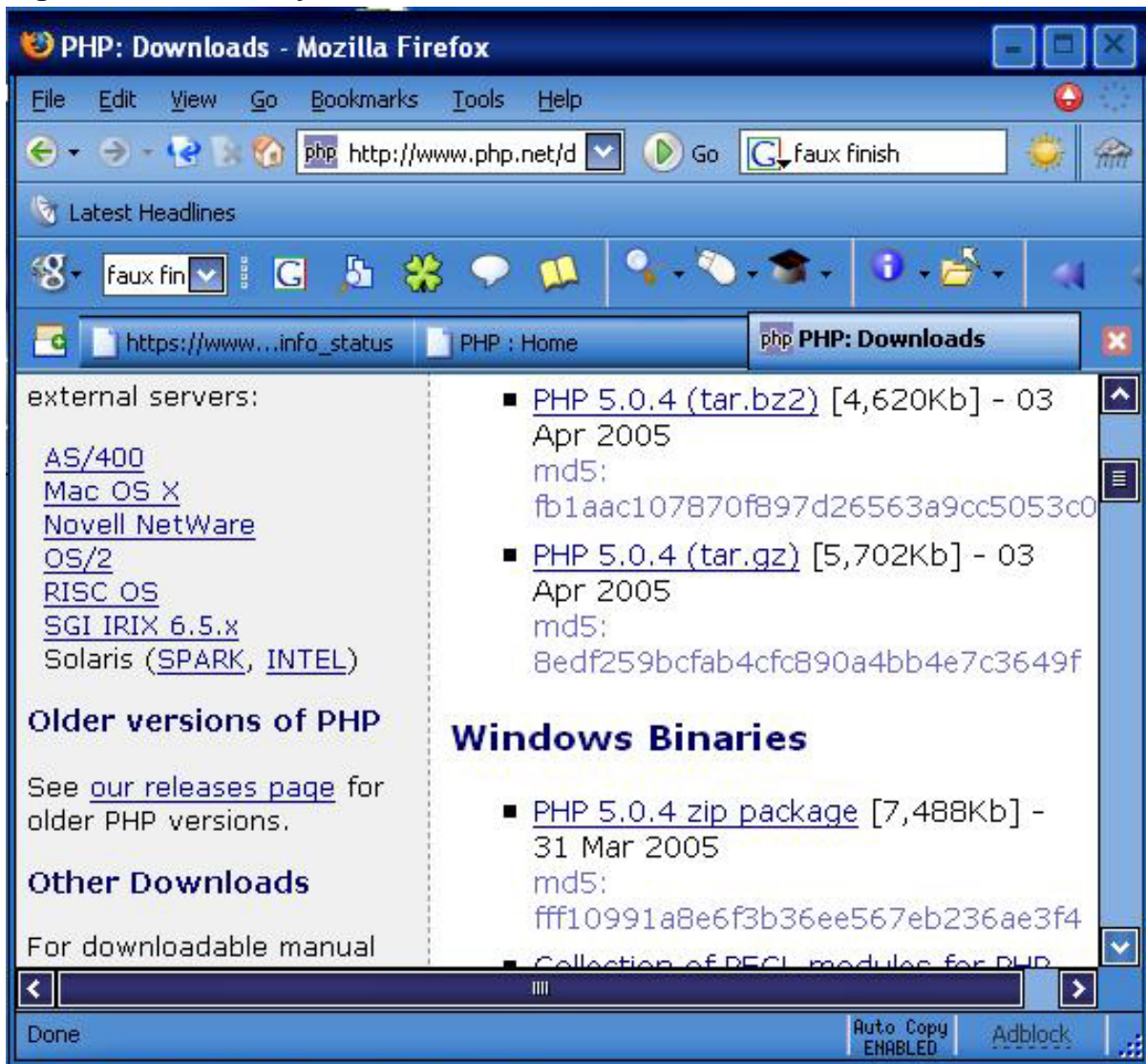
With Apache in place, you're ready to start something a little more interesting. PHP is -- in my opinion -- one of the best Web programming tools available. Sure, plenty of options exist along with a ton of applications. If you're building heavy-duty, server-side applications, Java might be the choice du jour; if you're a real hacker, Perl is great for getting down and dirty with a maximum amount of flexibility; JavaScript, AJAX, and Ruby (especially Ruby on Rails) are killer for highly interactive Web sites, like Google. But if you just want to get something basic done fast, I think PHP is the best option. It's easy to set up (something you're about to

see), integrates well with Apache, and once you get accustomed to the environment, you can go well beyond simple functionality. In short, it's just what it claims to be -- an easy-to-use, accessible, Web scripting language.

Of course, no matter how easy any language is to use, you must get your development environment up and running. This is where PHP shines. In just a few short pages, you'll have your scripts running on Apache. First, visit the PHP Web site, [www.php.net](http://www.php.net). Be sure not to visit php.com, which is something else entirely.

You can browse around the PHP Web site and get a wealth of information (including an introductory tutorial) later. Right now, you're here for the software, so look for the downloads link at the top of the page. Click on that link to go a page like that shown in Figure 6.

**Figure 6. Download your PHP software from the official Web site**



Look for the heading labeled Windows Binaries and find the link for the **PHP 5.0.4 zip package**; click the link and then choose the closest mirror to download the installer. (Do *not* download the PHP 5.0.4 Installer, as that does not include the MySQL extensions). Save the file to your local machine; since it's a temporary file, I just drop the zip file on my desktop. That's it; now it's on to the install.

## Install PHP on Windows

Once you have the zip file, double-click on it to open the zip file. Extract everything in the zip to `C:\PHP`.

Next? Well, you get to watch a progress indicator briefly, and you're done. Open `C:\PHP` and you should see something like Figure 7.

### **Figure 7. Installed PHP interpreter in C:\PHP**



The next step is to create a viable INI file, which tells PHP how to behave when it's called by other applications (like Apache). The best way to create an INI file is to copy one of the templates and rename it as `php.ini`. Look in `C:\PHP` and find the file named `php.ini-recommended`. Copy this file and paste it into the same directory; then rename it `php.ini`. That's all there is to it. You'll come back to this file much later, but for now, this is all you need to get up and running.

This process does *not* set up PHP to work with Apache. So, let's get back to some more text files and manual configuration.

## Connect PHP to Apache

Fire up your text editor and open Apache's `httpd.conf` file, as you did in earlier sections. Search for `ScriptAlias`, and you should see something like this:

```
#
# ScriptAlias: This controls which directories contain server scripts.
# ScriptAliases are essentially the same as Aliases, except that
# documents in the realname directory are treated as applications and
# run by the server when requested rather than as documents sent to the client.
# The same rules about trailing "/" apply to ScriptAlias directives as to
# Alias.
#
ScriptAlias /cgi-bin/ "C:/Program Files/Apache Group/Apache2/cgi-bin/"
```

This entry, for `/cgi-bin/`, tells Apache how to handle any URL that has the `cgi-bin` directory. For example, if you requested `http://www.newInstance.com/cgi-bin/mail-me`, Apache would not look for a file called `mail-me` in the `cgi-bin` directory. Instead, the `ScriptAlias` directive tells Apache to look in a different directory -- in this case, the `C:/Program Files/Apache Group/Apache2/cgi-bin/`. This is crucial, as you don't want to store scripts in Web-accessible directories. So you need to add an entry for PHP scripts. Here's what I did:

```
#
# ScriptAlias: This controls which directories contain server scripts.
# ScriptAliases are essentially the same as Aliases, except that
# documents in the realname directory are treated as applications and
# run by the server when requested rather than as documents sent to the client.
# The same rules about trailing "/" apply to ScriptAlias directives as to
# Alias.
#
ScriptAlias /cgi-bin/ "C:/Program Files/Apache Group/Apache2/cgi-bin/"
ScriptAlias /php/ "C:/PHP/"
```

You might think that `C:/PHP` isn't a great place to store scripts for your Web site. That's absolutely true; however, you're about to tell Apache to process all files ending in `.php` with the PHP interpreter, regardless of where they are located. So you're really not going to put anything in this directory that isn't already there; in other words, don't worry about this.

Next, you need to tell Apache that files ending in `.php` must be treated as an application; specifically, you want to assign a type to PHP files that you can then instruct Apache to process in a particular way. Just under your `ScriptAlias` entry, add this entry:

```
#
# ScriptAlias: This controls which directories contain server scripts.
```

```
# ScriptAliases are essentially the same as Aliases, except that
# documents in the realname directory are treated as applications and
# run by the server when requested rather than as documents sent to the client.
# The same rules about trailing "/" apply to ScriptAlias directives as to
# Alias.
#
ScriptAlias /cgi-bin/ "C:/Program Files/Apache Group/Apache2/cgi-bin/"
ScriptAlias /php/ "C:/PHP/"
AddType application/x-httpd-php .php
```

Now you just need to tell Apache what to do when it runs across this new type (linked up to files ending in `.php`). Basically, this just involves you telling Apache what program to run and hand off the PHP script to; of course, that magical program is the PHP interpreter you just installed. Add this directive right below the two you just put in place:

```
#
# ScriptAlias: This controls which directories contain server scripts.
# ScriptAliases are essentially the same as Aliases, except that
# documents in the realname directory are treated as applications and
# run by the server when requested rather than as documents sent to the client.
# The same rules about trailing "/" apply to ScriptAlias directives as to
# Alias.
#
ScriptAlias /cgi-bin/ "C:/Program Files/Apache Group/Apache2/cgi-bin/"
ScriptAlias /php/ "C:/PHP/"
AddType application/x-httpd-php .php
Action application/x-httpd-php "/php/php-cgi.exe"
```

Keep in mind that Apache already knows to translate `/php/` to `C:/PHP/`, so this actually assigns the PHP application type to the interpreter `C:/PHP/php-cgi.exe`. Of course, you can verify this file exists by browsing to the directory on your file system.

With all this in place, restart Apache and make sure you don't get any errors. If everything comes up OK when you try `http://localhost` as a simple test, you are in good shape.

## Test your PHP installation

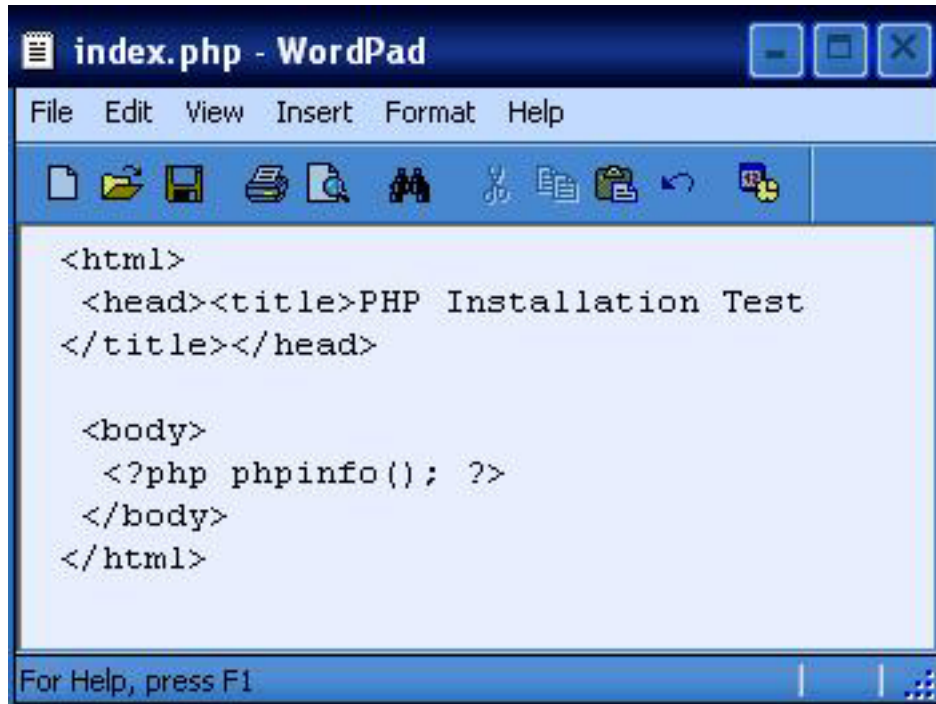
You don't have much left to do on the PHP front. First, make sure your PHP scripts work as they should. Crack open your text editor again and enter the following code:

```
<html>
  <head><title>PHP Installation Test</title></head>

  <body>
    <?php phpinfo(); ?>
  </body>
</html>
```

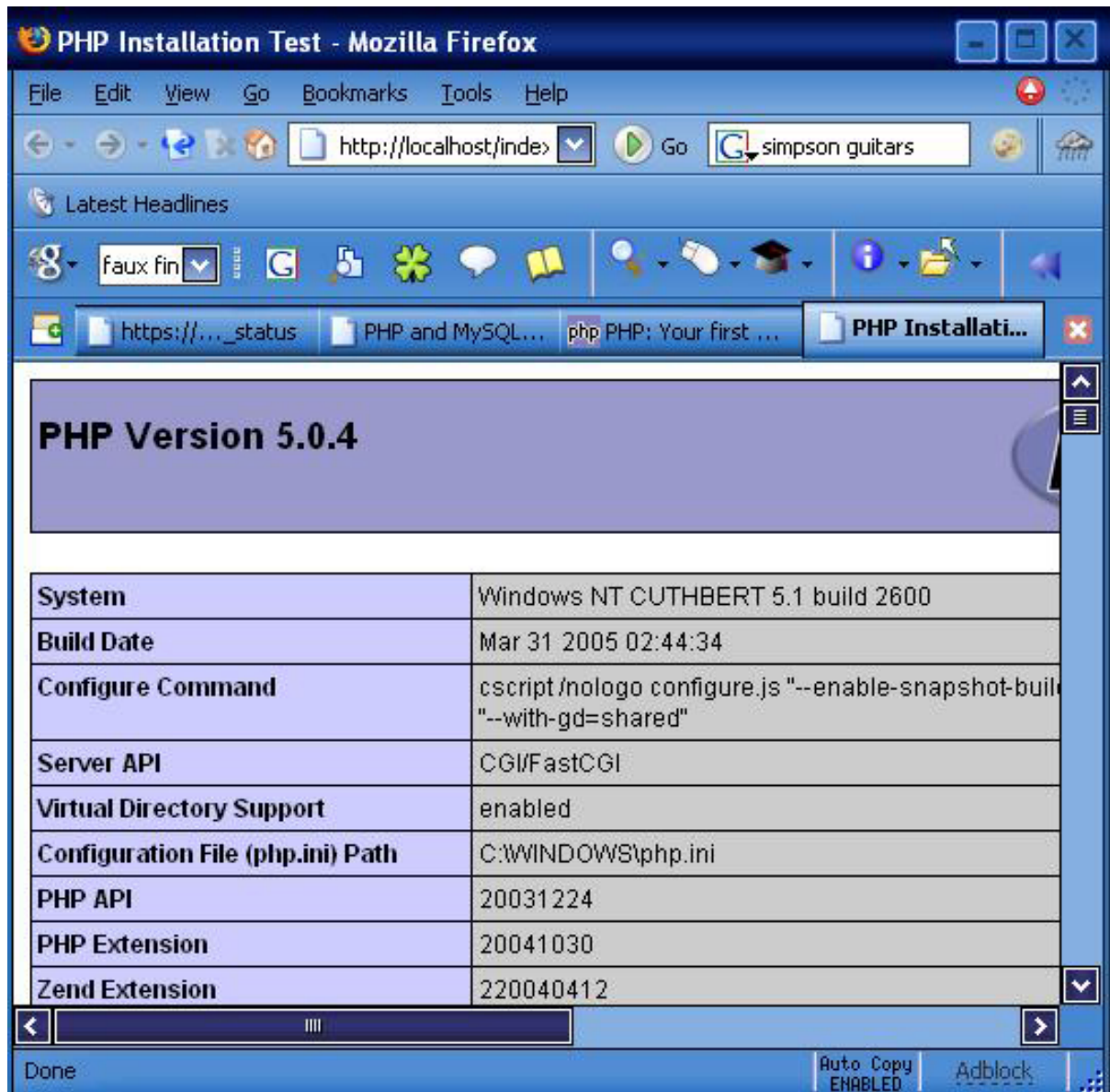
Save this file as `index.php` in the root directory of your Web site: `C:/Documents and Settings/Brett McLaughlin/My Documents/websites`. When I finished, I ended up with what you see in Figure 8.

**Figure 8. Editing a simple PHP script**



Now you can request this file as part of a normal URL: `http://localhost/index.php`. Fire up your favorite browser and see what you get. It should look something like Figure 9.

**Figure 9. The `phpinfo()` command outputs useful debugging information**



As should be obvious, the `phpinfo()` command gives you a ton of helpful information about your machine, the PHP interpreter, and everything else vaguely related to your development environment. And, in the process, it proves that your PHP installation works well.

## Allowing PHP scripts as index pages

You might want to make one last minor addition to your Apache and PHP installation. In some cases, you'll access your scripts as seen in the last section -- with a URL such as `http://localhost/my-script-name.php`. In other situations, though,

you might want your home pages to be PHP scripts. This is particularly useful if you work with includes and other scripting elements across your entire site and don't want users to have to enter a PHP-specific extension. In other words, you want `.php` to be a valid extension for default files, just as `.html` is. For example, file named `index.php` would be loaded by default, just as `index.html` is.

To make this change -- well, you guessed it -- open up Apache's `httpd.conf` file one more time. Search for `DirectoryIndex` this time, and you'll see something like this:

```
#
# DirectoryIndex: sets the file that Apache will serve if a directory
# is requested.
#
# The index.html.var file (a type-map) is used to deliver content-
# negotiated documents.  The MultiViews Option can be used for the
# same purpose, but it is much slower.
#
DirectoryIndex index.html index.html.var
```

To this list of files, add `index.php`:

```
#
# DirectoryIndex: sets the file that Apache will serve if a directory
# is requested.
#
# The index.html.var file (a type-map) is used to deliver content-
# negotiated documents.  The MultiViews Option can be used for the
# same purpose, but it is much slower.
#
DirectoryIndex index.html index.html.var index.php
```

Now restart Apache and enter the URL `http://localhost/z` (without the PHP file specified). You should see...well, exactly what you saw the last time you entered this URL: the My First Page page. Surprised? You have a file named `index.php`, so what's the deal? Well, the problem is that Apache looks for those files in the order they are specified. In this case, it finds `index.html` and stops looking before it ever finds `index.php`.

You might solve this by switching the order in `httpd.conf`, but that starts to really mess with normal Web server behavior. Instead of this (rather drastic) step, just rename `index.html` in your Web site root to `index.old`. Then go back and reload your browser (or reenter `http://localhost/`). Voila! Now that Apache doesn't find `index.html`, it looks for `index.html.var` (which isn't there) and then finally `index.php`. It loads this file, and you should see what you saw in Figure 9.

Pat yourself on the back. You have a working, fully-functional, mildly-customized Apache and PHP development environment.

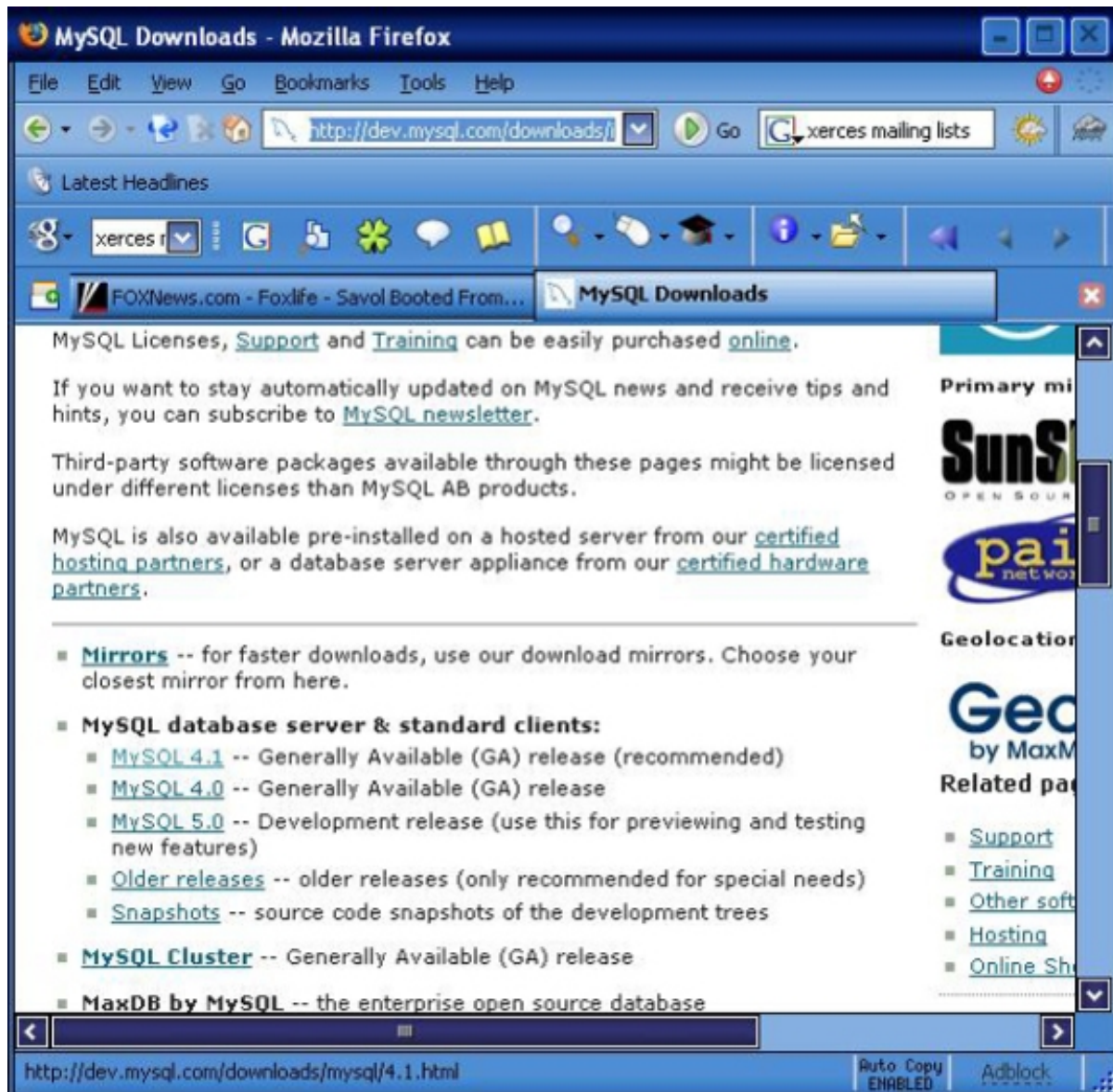
---

## Section 5. Install MySQL

### Download MySQL

At this point, downloading and installing software should be quite routine. For MySQL, go online and visit the [MySQL Web site](#). Along the top, you will see several tabs; click on the **Developer Zone** tab to open a submenu right below the tab. Click **Downloads**. Scroll down and select the **MySQL 4.1 GA** release (see Figure 10).

#### **Figure 10. Downloading the MySQL binary release**



This link takes you to the page for MySQL 4.1, and you can then select the download for your platform. Since you're using Windows for this tutorial, scroll down to Windows downloads and then look for Windows (x86). Look for the **Pick a mirror** link next to that line (be sure not to choose either Windows Essentials or Without installer). You'll be prompted to supply some information; if you have the time, fill out this form, as it helps the MySQL organization. In any case, you then just need to choose a mirror site and start the download. As I write this, the file is named `mysql-4.1.11-win32.zip`. Once the file downloads, you're ready for installation.

## Install MySQL

Open the downloaded zip file and extract the installer (`setup.exe`). Double-click on the expanded installer, and you should be up and running. Your first choice is the type of installation:

- **Typical:** MySQL is installed with the most common options. You basically let MySQL make the choices for you.
- **Complete:** The kitchen sink approach. MySQL is installed with everything possible thrown in.
- **Custom:** For control freaks like me, this lets you tweak every possible option.

Although I usually recommend the Custom option, Typical is the way to go for setting up a development environment. Confirm your choice on the next screen and select **Install**. Relax while the installation runs.

Next, you can log in or create a new MySQL.com account. I'll leave it to you whether you sign up, log in, or skip this altogether (I have an account). Once you're through this, you can finish up installation. The last option is to launch and configure MySQL right away, which you should choose to do.

## Configure your MySQL instance

Once installation is complete, MySQL launches the MySQL Server Instance Configuration Wizard. (You can access this program through **Start > Programs > MySQL > MySQL 4.1 > MySQL Server Instance Config Wizard**.) Select the Detailed Configuration option to get started with the wizard.

For the server type, select **Developer Machine**; just as the Wizard intimates, this is the right choice for a machine that you develop code on, and that must run several other applications (like Apache, for example). For the database, choose the **Multifunctional Database** option (again, this is the generic choice and works well because you'll be doing all sorts of different things with MySQL).

Next, set the directory for the InnoDB tablespace. If you read the fine print here, you'll realize you can just click **Next** and accept the default settings, which is what I recommend. The next screen prompts you for the number of connections. You should either select **Decision Support** (that assumes about 20 connections), or go with **Manual Setting** and select a lower number. Since I'm the only one working on my machine, and I only use MySQL for my Web applications, I selected Manual and set concurrent connections to 10 (which is probably still higher than I really need). I suggest you consider a similar setting.

You want the **Enable TCP/IP Networking** option on, and the default port of 3306 is fine. You should also accept the **Standard Character Set**, unless you have a solid

reason for going to UTF8 (I don't work with other languages in my applications, but if you do, go with UTF8).

The next screen has some important options. First, install MySQL as a Windows service and ensure that the **Launch the MySQL Server automatically** option is checked. This ensures MySQL starts up every time your machine reboots; Apache already does the same thing, so this makes a lot of sense. Additionally, check the **Include Bin Directory** option; this ensures that you can run the MySQL utilities without messing around with your PATH manually.

Next up is the root password; select a password. I urge you *not* to create an anonymous account. It's not a good idea in production, so why make it part of your development environment?

Finally, select Execute to set all these options on your database. Once the processing is done, you can select **Finish** to exit the instance configuration. At this point, your database is ready to roll.

## Create a new database

Now you need to create a new database on your MySQL installation. Since you've already set up the MySQL scripts in your PATH, this is easy. Open up a command prompt (Select **Start > Run** and type `cmd` in the Run box) and type this command:

```
mysqladmin -u root -p create DB-DEVEL
```

Enter the password you chose and click **Enter**. You won't get any feedback from MySQL (which is a bit annoying), unless something went wrong. Before checking your work, though, let me run through what you just did.

1. `mysqladmin` is the administration tool for MySQL. It lives in MySQL's `bin/` directory, which the configuration wizard added to your system's PATH, so you can access the program directly (as done here).
2. `-u root` tells MySQL to login as the root user, which you should have set a password for earlier. If you don't use this option, MySQL tries to log in as the `ODBC@localhost` user; that's useless because that user doesn't even exist!
3. `-p` instructs MySQL to prompt you for a password. You can enter the password on the command line, but then it's not hidden (`*****`), which is a bad practice. You'll use the `-p` flag for almost all your MySQL interaction.
4. `create DB-DEVEL` tells the command what to do; in this case, it

instructs MySQL to create a new database, named DB-DEVEL. I personally prefix my database names with "DB," just as a reminder to myself, and DEVEL stands for development; you can use a different database name if you like.

Since MySQL doesn't give you much feedback, you should ensure this command worked. To start the MySQL monitor, enter `mysql -u root -p` at a command line. Enter in your root password, and you'll see something like this:

```
C:\>mysql -u root -p
Enter password: *****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 7 to server version: 4.1.11-nt

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

This is where you'll do most of your database work, from creating tables to entering data. To ensure the database you created actually was created, enter `use DB-DEVEL`. `use` tells MySQL to switch to the named database. If everything is OK, you will see this output:

```
C:\>mysql -u root -p
Enter password: *****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 7 to server version: 4.1.11-nt

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> use DB-DEVEL
Database changed
mysql>
```

If the database doesn't exist, you'll get this error: `ERROR 1049 (42000): Unknown database 'DB-DEVEL'`. Back up a few steps and try again, until you get the database created. Then come back to this step, and you're ready to move on. Once you've verified that your database exists, type `exit` to leave the MySQL monitor.

## Add a table

With your database created, now you can add some simple tables and data to use in a test with PHP. Create a new text file and call it `create-sample-tables.sql`. The file contents should look like this:

```
CREATE TABLE users (
  user_id INT          NOT NULL,
```

```

username TEXT(8) NOT NULL,
firstname TEXT(35) NOT NULL,
lastname TEXT(50) NOT NULL,
PRIMARY KEY (user_id)
);

CREATE TABLE user_websites (
  website_id INT NOT NULL,
  user_id INT NOT NULL,
  website_url TEXT NOT NULL,
  PRIMARY KEY (website_id)
);

```

This creates a table called `users` and another table called `user_websites`. The rest is pretty self-explanatory. You can now feed this into MySQL like this:

```

C:\>mysql -u root -p DB-DEVEL < create-sample-tables.sql
Enter password: *****

```

The first part of this command should be familiar. I added `DB-DEVEL` to the end, which tells MySQL to start up and connect to that specific database (a shortcut to logging in and then typing `use DB-DEVEL;`). Then, the `<` symbol tells MySQL to accept as input the SQL file you just created. Unfortunately, MySQL still gives you no feedback, unless there's an error. In any case, take silence as assent.

Again, check your work. Log into the MySQL monitor with `mysql -u root -p DB-DEVEL` and enter `describe users;`. You should get a nice textual description of your table:

```

mysql> describe users;
+-----+-----+-----+-----+-----+-----+
| Field      | Type      | Null  | Key  | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| user_id    | int(11)   |       | PRI  | 0       |       |
| username   | tinytext  |       |      |         |       |
| firstname  | tinytext  |       |      |         |       |
| lastname   | tinytext  |       |      |         |       |
+-----+-----+-----+-----+-----+-----+
4 rows in set (0.00 sec)

```

You can do the same thing with the `user_websites` table:

```

mysql> describe user_websites;
+-----+-----+-----+-----+-----+-----+
| Field      | Type      | Null  | Key  | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| website_id | int(11)   |       | PRI  | 0       |       |
| user_id    | int(11)   |       |      | 0       |       |
| website_url | text      |       |      |         |       |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)

```

Finally, add some simple data in here. Create the following file and name it

insert-sample-data.sql:

```
INSERT INTO users VALUES (1, 'bmclaugh', 'Brett', 'McLaughlin');
INSERT INTO users VALUES (2, 'ggg10012', 'Gary', 'Greathouse');
INSERT INTO users VALUES (3, 'reckstei', 'Bob', 'Eckstein');
INSERT INTO users VALUES (4, 'mikeh', 'Mike', 'Hendrickson');

INSERT INTO user_websites VALUES (1, 1, 'http://www.newInstance.com');
INSERT INTO user_websites VALUES (2, 1, 'http://www.oreilly.com');
INSERT INTO user_websites VALUES (3, 4, 'http://www.oreilly.com');
```

Again, this should be quite straightforward. Feed this script into MySQL with this command:

```
C:\>mysql -u root -p DB-DEVEL < insert-sample-data.sql
Enter password: *****
```

Now, log in to ensure this data went where it was supposed to go:

```
C:\>mysql -u root -p DB-DEVEL
Enter password: *****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 16 to server version: 4.1.11-nt

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> select * from users;
+-----+-----+-----+-----+
| user_id | username | firstname | lastname |
+-----+-----+-----+-----+
| 1 | bmclaugh | Brett | McLaughlin |
| 2 | ggg10012 | Gary | Greathouse |
| 3 | reckstei | Bob | Eckstein |
| 4 | mikeh | Mike | Hendrickson |
+-----+-----+-----+-----+
4 rows in set (0.00 sec)
```

You can perform the same check on the `user_websites` table with similar results. With some sample data in place, all you need now is some PHP code to piece it together with your Web site.

## Add MySQL support to PHP

Before you can write any code, you have to tell PHP to load the libraries it needs for working with MySQL. This is done through the `php.ini` file you set up way back in [Install PHP on Windows](#). Open that file and look for a line like this:

```
;extension=php_mysql.dll
```

This line tells PHP to load the MySQL DLL (dynamic link library), which then allows your PHP scripts to make use of the MySQL commands. However, the `;` at the front of the line is a comment indicator; remove that so the line looks like this:

```
extension=php_mysql.dll
```

Now, drop out to a command line and try to run the PHP interpreter with the `-i` switch, which will tell you what modules are loaded:

```
C:\>c:\php\php -i
```

You'll probably get an error here, something like this:

```
PHP Startup: Unable to load dynamic library 'C:/foo/ext/php_mysql.dll' -  
The specified module could not be found.
```

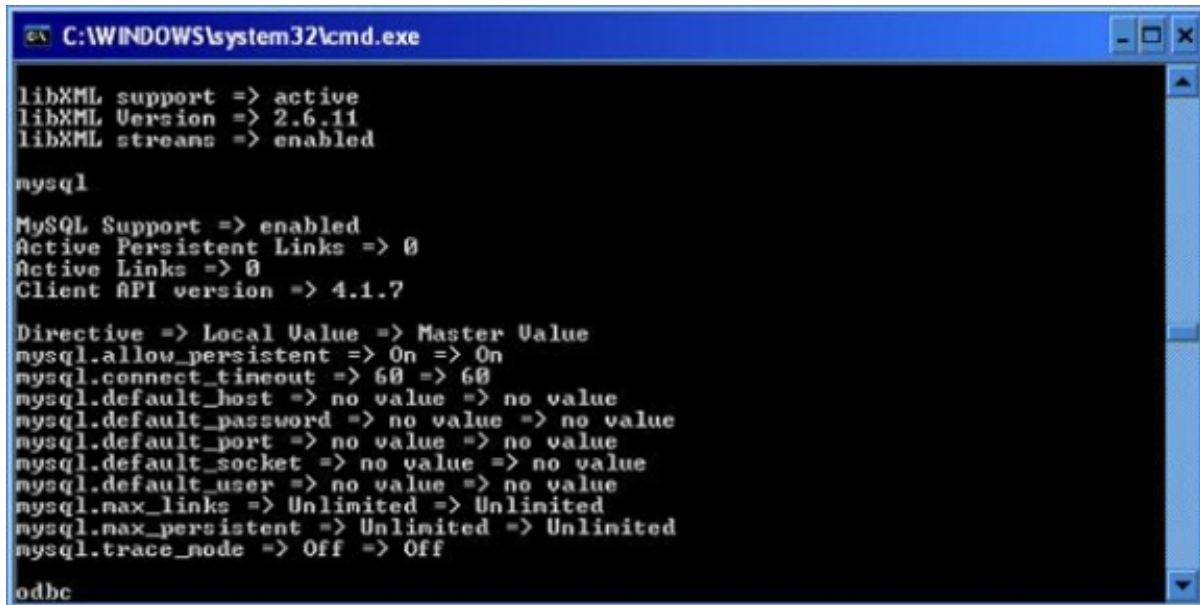
**Note:** For display purposes, the above message appears on multiple lines.

Don't worry. This is OK. PHP is now looking for the MySQL libraries, but doesn't know how to find them. Open `php.ini` again and search for `extension_dir`. You need to make the entry look like this:

```
; Directory in which the loadable extensions (modules) reside.  
extension_dir = "C:/PHP/ext/"
```

This tells it to look in the `C:/PHP/ext/` directory for the `php_mysql.dll` module; if you look in that directory, you'll see the file. Save `php.ini` and try running `php -i` again. If everything is set up correctly, you'll get a huge slew of information about your PHP interpreter. That's what you want. You can even scroll up and see some MySQL configuration data, as Figure 11 shows.

**Figure 11. PHP displays the modules it has loaded**



```
C:\WINDOWS\system32\cmd.exe

libXML support => active
libXML Version => 2.6.11
libXML streams => enabled

mysql

MySQL Support => enabled
Active Persistent Links => 0
Active Links => 0
Client API version => 4.1.7

Directive => Local Value => Master Value
mysql.allow_persistent => On => On
mysql.connect_timeout => 60 => 60
mysql.default_host => no value => no value
mysql.default_password => no value => no value
mysql.default_port => no value => no value
mysql.default_socket => no value => no value
mysql.default_user => no value => no value
mysql.max_links => Unlimited => Unlimited
mysql.max_persistent => Unlimited => Unlimited
mysql.trace_mode => Off => Off

odbc
```

Now, you're ready to write some PHP code.

## Connect to MySQL in a PHP script

With Apache set up, PHP running, MySQL installed, and your MySQL extensions loaded into PHP, you're finally ready to write some PHP. Close all those configuration files and open up a new, blank file. Enter in the following text:

```
<html>
  <head><title>MySQL Connection Test</title></head>
  <body>

<?php

$conn = @mysql_connect("localhost", "root", "your-password");
if (!$conn)
  die("<p>Error connecting to MySQL: <i>" . mysql_error() . "</i></p>");
echo("<p>Connected to Database Successfully.</p>");

if (@mysql_select_db("DB-DEVEL", $conn))
  die("<p>Error selecting DB-DEVEL database: <i>" . mysql_error() . "</i></p>");
echo("<p>Selected DB-DEVEL database successfully.</p>");

mysql_close ($conn);
echo("<p>Disconnected</p>");

?>

  </body>
</html>
```

This is a lot to throw at you if you're not used to writing PHP code, but it turns out to be fairly simple:

1. `mysql_connect` attempts to connect to a MySQL database using the hostname, username, and password you supply it. Be sure to replace *your-password* with your real password.
2. `die` spits out error messages and halts further PHP processing. It's used if the `conn` variable is false, which it will be if `mysql_connect` doesn't succeed.
3. `echo` does just what it says; it spits out text (in this case HTML).
4. `mysql_select_db` attempts to change to the supplied database using the connection object you pass to it (the `conn` object representing a connection to the database).
5. `mysql_close`... well, you can figure that one out for yourself.

As you can see, it really doesn't take a lot of code to make sure your development environment works as it should. Run this script by saving it as `test-connect.php` in the same directory as your other Web files (like `index.php`). Finally, load the file in your Web browser: `http://localhost/test-connect.php`. You should see something like Figure 12.

### Figure 12. Successful connection test

Assuming your test was successful, congratulate yourself! You've installed Apache, PHP, and MySQL, and even have a valid connection running to your database. You're almost finished. You just have to connect to your sample data and make sure you can retrieve it without any problems.

## Retrieve data from MySQL with PHP

Before wrapping up, I want to quickly show you how to connect to MySQL and retrieve rows from the database. I won't get into the details (this isn't a tutorial on PHP), but I just want to give you a simple script to run to ensure things work. It will also be a nice starting point as you begin to write your own custom PHP code. Save this file as `list-users.php`.

```
<html>
<head><title>Listing of Users in Database</title></head>
<body>

<?php
$conn = @mysql_connect("localhost", "root", "seth0723");
if (!$conn)
    die("<p>Error connecting to MySQL: <i>" . mysql_error() . "</i></p>");
if (!@mysql_select_db("DB-DEVEL", $conn))
    die("<p>Error selecting DB-DEVEL database: <i>" . mysql_error() . "</i></p>");
```

```

?>
<table border="1">
  <tr><th>Username</th><th>Full Name</th><th>Website</th></tr>

<?php
$select = '    SELECT u.username, u.firstname, u.lastname, w.website_url';
$from   = '    FROM users u, user_websites w';
$where  = '    WHERE u.user_id = w.user_id';
$order  = ' ORDER BY u.lastname, u.firstname';

$users = @mysql_query($select . $from . $where . $order);
if (!$users) {
  echo('</table>');
  die('<p>Error retrieving users from database!<br />'.
    'Error: ' . mysql_error() . '</p>');
}

while ($user = mysql_fetch_array($users)) {
  $user_username = htmlspecialchars($user['username']);
  $user_firstname = htmlspecialchars($user['firstname']);
  $user_lastname  = htmlspecialchars($user['lastname']);
  $user_website   = htmlspecialchars($user['website_url']);

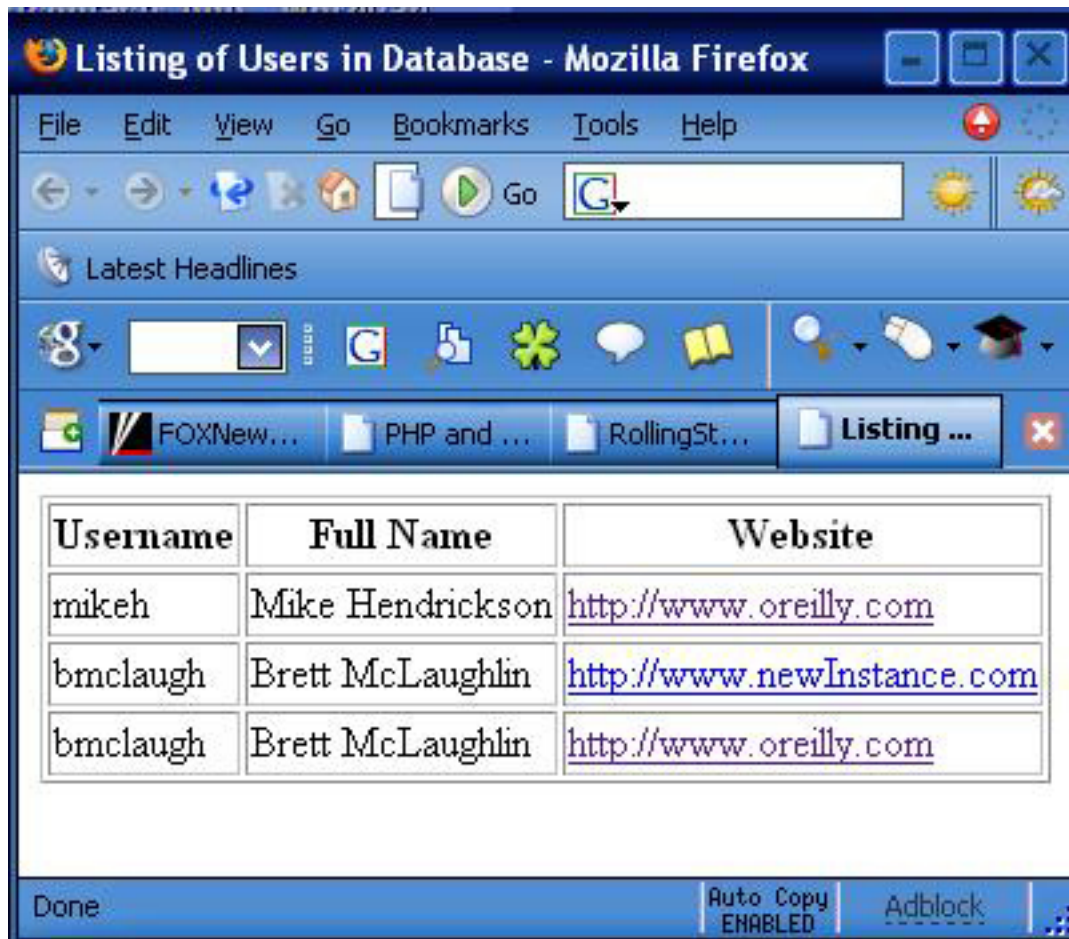
  echo("&<tr>\n");
  echo("&<td>$user_username</td>\n");
  echo("&<td>$user_firstname $user_lastname</td>\n");
  echo("&<td><a href=\"\$user_website\">$user_website</a></td>\n");
  echo("</tr>\n");
}
?>
</table>

<?php
  mysql_close ($conn);
?>
</body>
</html>

```

This simple PHP page connects to the MySQL database, switches to the DB-DEVEL table, and pulls all users with Web site URLs. There's nothing flashy to it, and the format looks simple, but it illustrates the point: using PHP to access MySQL data is very easy. Figure 13 shows you this script in action.

**Figure 13. HTML listing of users with Web site URLs**



That's it! Your development environment is set up and ready to go.

---

## Section 6. Summary

### Summary

At this point, you have a complete development environment set up on your Windows XP machine. Whether it's a laptop that isn't always connected to the Internet or a desktop you're on 8 to 10 hours a day, you won't have to connect to your ISP just to test out simple PHP scripts. With the latest version of Apache's Web server, PHP 5, and MySQL, you're ready to program and deploy -- all on your local machine. In fact, this is the exact setup I run on my laptop -- and I set up a new laptop using this tutorial.

You are now familiar with configuration files and might even be willing to tinker a bit with your personal setup. You might want to create your own virtual hosts for Apache, run multiple versions of PHP for backwards compatibility, or even generate several instances of MySQL to break out your data. Whatever specific configuration you use, I hope you've seen that you don't have to mess around with your ISP to build and test your applications, and you certainly don't need to buy and maintain a Linux server just for that purpose. While you can find plenty of good uses for Linux, Mac OS X, and heavy-duty operating systems like Solaris, using them purely on a staging machine for simple scripting applications sure isn't one of them.

However you use this tutorial, I hope it takes your development to the next level -- whether you're connected to the Internet or not. Enjoy!

# Resources

## Learn

- Download Apache projects and find out more about the Apache Software Foundation at the [Apache Web site](#).
- Learn more about the Apache Web server at its own [Web site](#).
- Download and learn about PHP at the [PHP Web site](#).
- Visit the [MySQL Web site](#) for free downloads, documentation, and even commercial support.
- Download PHPMyAdmin's [great Web-based tool for working with MySQL databases](#).
- Explore this [comprehensive FAQ](#) page that tells you everything about Cloudscape's functionality, and includes code that might be useful in set-up (developerWorks, February 2005).
- Explore [Creating dynamic Web sites with PHP and MySQL](#), by Md. Ashraful Anam (developerWorks, May 2001).
- Read how to [Access an enterprise application from a PHP script](#), by Caroline Maynard, Graham Charters, and Matthew Peters (developerWorks, February 2005).
- Find hundreds more Web architecture resources on the [developerWorks Web architecture zone](#).

## Get products and technologies

- [Download Cloudscape](#) for a free trial, and learn what's new in v10.0 (developerWorks, September 2004).
- In this demo, learn to [install Cloudscape](#) and to connect to it with Eclipse (developerWorks, October 2004).

## About the author

### Brett McLaughlin

Brett McLaughlin has worked in computers since the Logo days (remember the little triangle?). In recent years, he's become a well-known author and programmer in the Java and XML communities. He's worked for Nextel Communications, implementing complex enterprise systems; at Lutris Technologies, writing application servers; and most recently at O'Reilly Media, where he continues to write and

edit books that matter. His most recent book, [Java 1.5 Tiger: A Developer's Notebook](#), is the first book available on the newest version of Java technology, and his classic *Java and XML* remains one of the definitive works on using XML technologies in the Java language.