The Web Book

How to create Web sites and applications with HTML, CSS, Javascript, PHP and MySQL.

By Robert Schifreen

The Small Print. Please Read.

The Web Book is published by Oakworth Business Publishing Ltd.
Registered in the UK. Company number 2783266.
Telephone 01424 213872 (UK)
+44 1424 213872 (International)

Edition 2.1 (September 2009).

Contents

About The Web Book

If you want to create web sites, there are hundreds of books and web pages that claim to show you how. Some of them are very good indeed. But this book isn't like all those other books and web pages, for a number of important reasons:

1. The Web Book is an electronic book, or e-book. You simply download it as a PDF file from www.the-web-book.com and print it yourself. Or read it on-screen.

2. The Web Book covers all of the technologies that you need to know in order to create Web sites, both using static HTML pages and database-driven sites.

3. Unlike many books on the subject that were written some years ago, The Web Book teaches you up-to-date methods. Follow the instructions here and you can be confident that you're doing things in the right way, rather than using old-fashioned techniques that are now frowned upon.

4. I've tried my hardest to keep everything non-technical. If you're "into" computers, you should be able to follow everything just fine. You certainly don't need to be a professional techie. In fact, if you are, you'll probably take offence at the way I've simplified some things. For which I apologise.

5. If you already look after a web site, perhaps for your school or college, or the department you work for, you may be itching to take your skills to the next level. Or maybe you didn't actually get much training when you took on the responsibility, and you don't really understand how everything fits together. In which case, this book is perfect for you. We don't just tell you to press buttons. We explain what those buttons do, and why you need to press (or not press!) them.

6. Here's the best bit. The Web Book is **free of charge**. So if you want to teach yourself how to do Web stuff, whether for creating your own sites or to make sites for other people, just grab a copy of the PDF file, print out the book, and away you go.

One word of warning, though. Creating a web site and doing it properly isn't a simple task that can be done in an afternoon. Yes, we've all seen 2-page magazine articles that imply otherwise, but sadly it's just not true. Even at 329 pages, this book is only a basic introduction to some of the more complex topics. It's quite possible to buy books on HTML, CSS, PHP and MySQL which each run to 800 pages. I wouldn't recommend it, however, unless you have trouble sleeping.

Why We Recommend Hostmonster.com

If you want to create web sites, you need a company to host them for you. There are thousands of such companies out there. The examples in this book are based around one in particular, namely **www.hostmonster.com**. We chose this company because their products appear to be good, and they offer good value. They host the site from which you will have downloaded this book.

If you're looking for a web host, we'd love you to use hostmonster because, if you sign up via our web site at www.the-web-book.com, we get a small amount of commission. But the web is all about freedom of choice, so if you want to go elsewhere you're more than welcome to do so. You'll need to adapt the examples slightly, but details of any settings or concepts that are unique to hostmonster are clearly identified in the text.

If you're considering using a different host, check our web site first. There may be a custom edition of the book available for that host. We're working on new ones all the time.

A Custom Edition For Your Company

If your company sells web hosting services and you wish to give away or sell copies of this e-book, we can create a custom version of the book for you, where all the examples feature your hosting. You can then point your customers at our web site to download their copy.

For information on this service, see **www.the-web-boook.com/customize.html** or email **info@the-web-book.com**.

Who's Written This Book? And Why?

The Web Book is written by Robert Schifreen, a UK-based journalist, writer and technical author. I have more than 25 years' experience of writing all sorts of technical articles for various computer magazines, and 9 years' experience developing web sites for myself and clients. I've also written a "traditional" book before, that was published in 2006 and was available as a real hardback in real bookshops. A Google search for "defeating the hacker" will find it for you.

Having taught web development to the 2,500 staff at a UK-based university for the past few years, I thought it might be helpful to turn my course into a book, which is loosely based on the stuff I teach and is also inspired by the questions that my students ask. From now on, when people ask me how to create a web site, or whether it's easy, I don't have to spend time explaining everything. I can just say "go to **www.the-web-book.com** and download my free book, which tells you everything you need to know".

This is the book which I really wish had existed when I started doing web stuff all those years ago. But, you're probably asking, why am I giving it away instead of selling it? There are 3 reasons.

One. I quite like the idea of being able to write what I want for a change, without having publishers and editors to answer to.

Two. I want as many people as possible to see this book, because I think it's useful. The trouble with traditional publishing is that, of the hard-earned money that you hand over to pay for a copy of a traditional book, most of it is swallowed up by the publishers and their marketing costs. The author, the guy who sat in front of his PC for months writing the thing, typically gets around 15% of your cash. Sometimes less. So I've decided to try something different. This book is free to download and use. It is financed in 3 ways:

- If you want to make a small donation via **www.the-web-book.com/donate.html** you're more than welcome.
- If you run a web hosting company and you would like me to produce a custom version of the book that features your hosting service, please see the page online at **www.the-web-book.com/customize.html** for details and pricing.
- If you want to sign up with a web host, I'd really appreciate you signing up with Hostmonster (who are featured in this edition of the book). Just click the Hostmonster icon on **www.the-web-book.com/download.php** and everything happens automatically. This brings me a small commission payment, which helps to pay for the 5 solid months it took me to write The Web Book. It won't cost you any more, but it means a lot to me.

And the third reason why I'm giving away the book? Having been a professional IT journalist, editor and writer since 1983, I've seen what the internet is doing to the publishers of magazines and books. Especially those companies which produce IT-related publications. Computer magazines across the world are being shut down and book sellers are having a hard time, because the Web, not the pages of a book, is undoubtedly the best place to find information about technical subjects.

In many cases, those publications shut down not because the product was poor, but because the publishers failed to see the internet coming and failed to act accordingly. Publishers need to work with the internet, not against it. Sure, a search engine means that I can find an answer to a technical problem instantly without having to plough through shelves of books and magazines. But the Web is also the greatest, biggest, best, cheapest and most efficient way of copying and disseminating information that's ever been invented. Just ask the music industry. So by promoting this book on the internet as a free download I hope that I can exploit the power of the internet rather than trying to work against it.

Why We're Here

No one could have believed, in the last years of the 20th century, that the way we'd all been doing Web sites was about to change so dramatically. But it did, in two fundamental ways.

Firstly, Web sites generally consisted of a collection of HTML document files. Each page of the site was a separate file (don't worry if you don't understand much of what I'm saying here - everything will be explained more fully shortly). While this works for relatively simple sites, with just a few pages, there are better ways to create web sites. Using database technologies such as MySQL, and programming languages such as PHP, you can create not just simple informational web sites that allow you to publish information, but two-way sites that allow your visitors to interact. Stop thinking "electronic newspaper" and think Facebook or Ebay or Amazon.

The second major change was in the fundamental way that web pages were structured, in terms of layout and formatting. Font tags gave way to something called CSS, or Cascading Style Sheets.

In a nutshell, creating web sites has become vastly more complicated over the past decade. You'll see from the front cover that this book covers HTML, CSS, Javascript, PHP and MySQL. If you're daunted by all that, then don't be. We'll cover everything in a logical order, building up your knowledge in layers, so that all the new material you learn will make perfect sense because you'll be able to see it in context.

You are, of course, free to dip into the book as you wish, reading sections in whatever order pleases you. If you want to know what PHP or MySQL is all about, then by all means skip straight to those chapters. But if you can possibly spare the time, you'll find it much more beneficial to read everything in order. It'll make much more sense in the long run.

Of course, you may not actually need this book at all, if all you want to do is to put some information online as quickly as possible so that other people can access it over the web. There are plenty of free services out there that will allow you to do this, quickly and easily. Among the best-known of these services are:

www.myspace.com	A social networking site, especially good if you want to upload pictures and music rather than text.
www.facebook.com	The best-known social networking site, where you can post information about yourself.
www.wordpress.com	If you want a blog but you don't want to run your own version of the WordPress blogging software, this site will host a blog for you. Just sign up for free on the web.

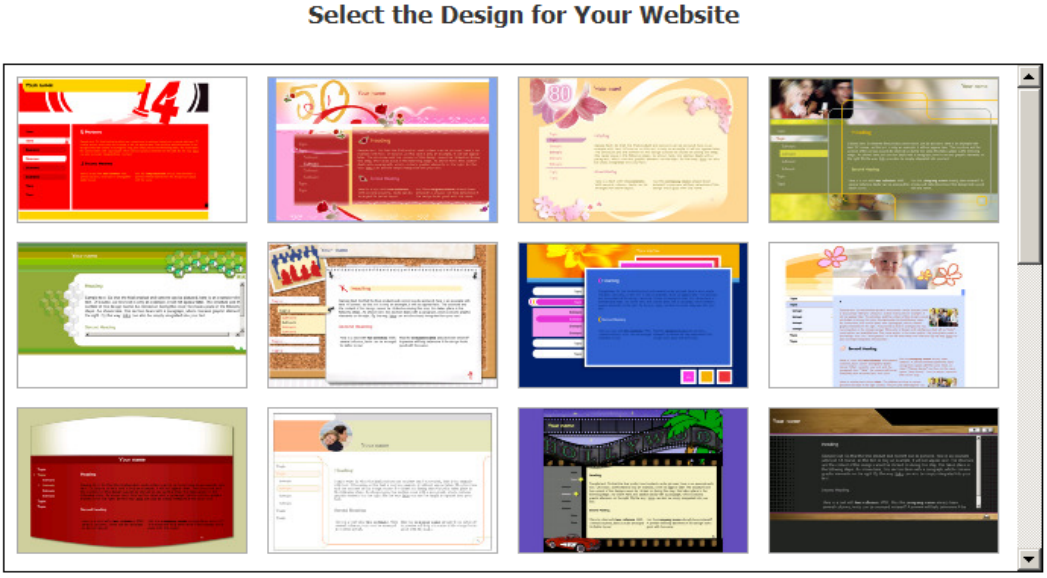
www.blogger.com	Another blogging site which is free to use. Owned by Google. If you want a personal blog with the minimum of fuss, then using this site or wordpress.com is by far the easiest way to do it.
sites.google.com	Create text-based pages and publish them online for free, courtesy of Google.
www.flickr.com	The best-known place to upload your photos and share them with the world.
www.youtube.com	Does for video what flickr does for stills. Upload 10-minute clips of just about anything.
docs.google.com	Google's web-accessible word processor and spreadsheet. Use it just for yourself, or share documents with friends for collaborative editing.
spaces.live.com	Free web space for you publish a blog, pictures, music, or whatever you like. From Microsoft.
www.twitter.com	The micro-blogging site, where you can tell everyone what you're up to.

Before you read any further, you may want to consider signing up with one or more of these systems, even if only to gain some inspiration into what your own site should, or shouldn't, look like.

Another option is to sign up with a site that offers an easy, automated way to create web pages. Many hosting companies offer such things, thus making it easy to create pages on the hosting space that you've just bought. Among such offerings are Easy Website Creator from leading hosting company 1&1, the design selection screen of which looks like this:

Just pick a design and a colour scheme, type in some text, and you have a web site.

Another example of the genre is from UK-based hosting company easily.co.uk. Again, their design selection screen is shown below:



Once again, choose a design and just start typing.

Although these systems allow you to create a web site quickly and easily, they are very limited. If you want JavaScript, PHP and MySQL, you're normally unable to add it. You

can't easily make major changes to the design. And you're frequently limited to just 20 or 30 pages, depending on which hosting package you sign up for. Sometimes the figure is as low as just one page.

From Word Processor to Web Site

There's yet another way to create web pages, and that's to write the text with a word processor such as Microsoft Word or OpenOffice Writer and then use the "save as web page" feature to create files which you can put online. This is often a tempting option, because a) it looks to be very easy, and b) lots of people write documents with a word processor and then realise that they'd actually quite like to publish them online.

The trouble is, the "save as web page" feature in Microsoft Word, including the most recent version, is horrible. It produces pages which contain ten times as much information as is strictly necessary. Which means that they'll take up ten times the space on your server, take your visitors ten times as long to download, and cost people ten times as much if they're accessing your site via mobile broadband or some other metered or pay-per-megabyte connection. Seriously, don't do it.

If you do have some Word documents that you want to put online, there are some excellent third-party products available to do the conversion for you. My favourite is "Click To Convert", which is what I used to create the web-based preview version of this book. All of the pages at **`www.the-web-book.com/preview.php`** were generated automatically by Click To Convert.

If you've bought a hosting package and a domain name, and you're not quite sure what to do with it, one of the quick-start pages as mentioned above is better than nothing at all. But if you really want to make the most of the web, there really is no substitute for learning how to do it properly, and that's what you'll do if you continue reading.

But How Long will All This Take?

I do hope that you decide to read this book all the way through, and follow the examples in it. Remember that this is a textbook rather than a novel. By all means read through it in a day or two, but you won't get the best from it. For that, you need to work through it gradually, at your computer, following the examples. You should allow a week or 2, at least, to get the greatest benefit. Maybe even a little longer if you want to experiment further, and enhance some of the examples with your own ideas.

What Is a Web Site Anyway?

A web site is, traditionally, a collection of pages of information. Creating a web page is, in many ways, very similar to writing a letter with a word processor and saving it on your computer, but there are a couple of important differences.

First, you have to save the page in a special format (ie, language) known as HTML, rather than as a normal document. HTML stands for Hypertext Markup Language. Why hypertext? That's the name given to the way that we move between web pages by clicking on hyperlinks (those bits of text which are normally in blue and underlined). A markup language is just a way of "marking up" text to specify that, for instance, when the visitor to our site clicks on Home he gets taken to the home page.

The reason we have to save our pages in HTML format, rather than as Word documents, Excel spreadsheets, PDF files, Zip files, etc etc, is that the way we read web sites is with a program called a web browser. For example, Internet Explorer, Firefox, Safari, Google Chrome, or Opera. The only type of information that web browsers are guaranteed to be able to display are HTML files. Sure, if you put a Word document on your web site, or a PDF file, some web browsers might make a good stab at displaying the file. But it's never guaranteed to work.

The second difference between creating web pages and writing a letter is that, having created your web page, you obviously need to save it. But rather than saving on your own computer, where only you can see it, you need to save it onto a web server. A web server is simply a normal computer, connected to the internet, which runs a web server program. This program means that other people's computers across the internet can connect to it, request a copy of your page, and display it.

In theory, any computer that has a permanent connection to the internet can be turned into a web server. Just install the necessary software, which is easily available free of charge, and the job is done. However, hackers love breaking into web servers and crashing them, or trying to change the contents of the pages they store. So unless you really know what you're doing, it's much easier and safer to rent some space on someone else's web server to store your web sites, rather than running your own server. It's very cheap to do, as we'll discover later.

How the Web Works

It's useful at this point to outline, in very basic terms, just how the World Wide Web actually works in practice. What really happens when you turn on your computer, open up your web

browser and type **www.the-web-book.com** into the address bar? How does the information get onto your screen? And where does it come from in the first place?

As I mentioned above, a web page is a document file, stored on a web server, created with a program that's a bit like a word processor but which saves its files in HTML format. When you open up your web browser and type **www.the-web-book.com**, your computer connects via the internet to a large, centralised directory in order to find out where the **www.the-web-book.com** site is stored. This directory is called a DNS Server. DNS is the Domain Name System, which gives each web server (or rather, each site) a unique name. In this case, the-web-book.com. Computers, of course, don't like names. They prefer numbers. In the case of the internet, each web server has a unique number known as an IP address. So the DNS directory allows your computer to look up the IP address of the server which holds the **the-web-book.com** site.

Once your computer knows the IP address of the web server which holds the site, your computer can then connect, again over the internet, directly to that server. Your web browser sends a request to the web server, asking for a specific page of the site. The server sends that page (or, if no specific page is specified, it sends the home page). Your browser then displays the contents of that page, and the process is complete. At least, until you click on a link in order to see another page, and the process starts all over again.

Domain Names

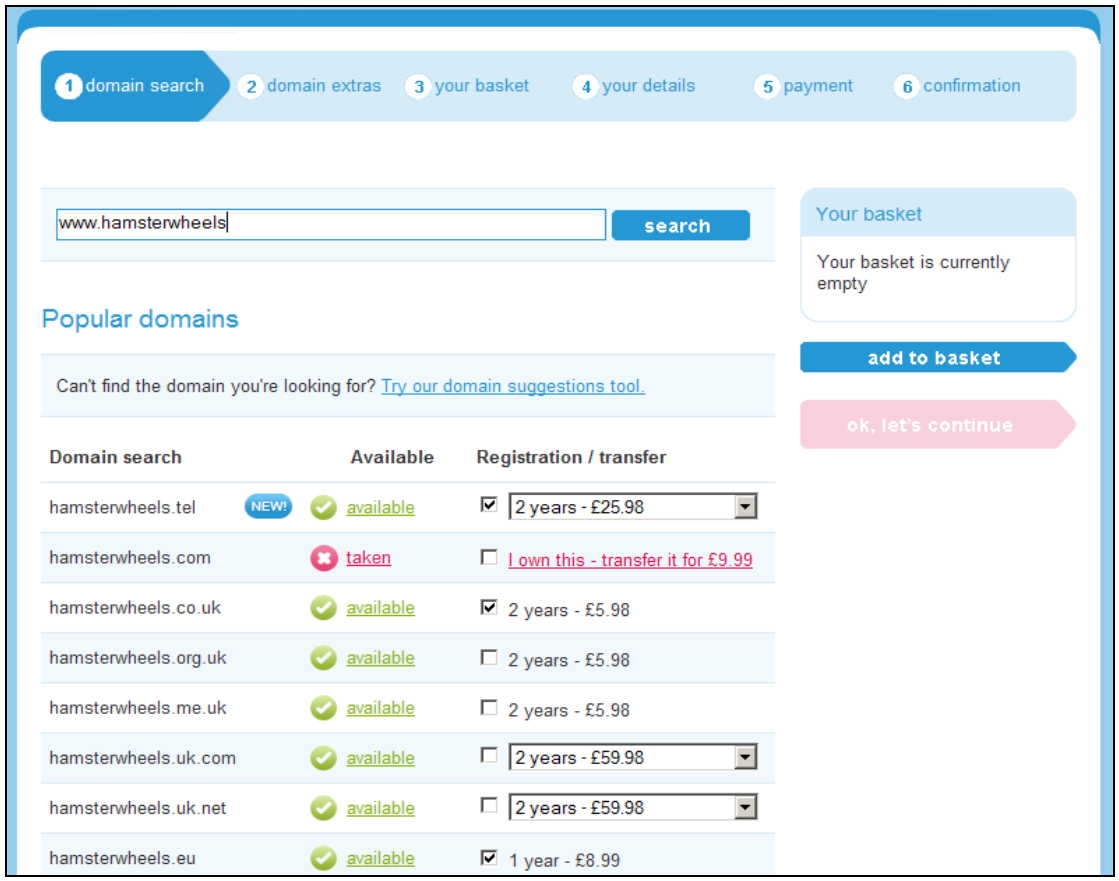
So how do new internet domain names get created? Or to put it another way, how do you get the name of your site into that master DNS directory? It's a relatively straightforward process.

Let's imagine that you want to set up a brand new web site from scratch. The first thing you need to do is to think of a name for your site. For this example, we're going to start selling hamster wheels online. We need a web site, and we want to call it **www.hamsterwheels.com**. To create a new web site name, and add it to the global DNS directory, we need to call upon the services of a domain name registration company. There are thousands of such companies, any of which will sell you an entry in the global DNS directory. Prices vary, but they're all selling you the same service, so shop around and find a registrar that you're happy to deal with.

It's easiest if the registrar is based in your own country, but it's not essential. If you don't know where to find such a company, type "domain name registration" into Google or look at the adverts in your favourite computer magazine.

For this example, we'll use a UK-based registration company called 123-reg.co.uk. Once you surf to their web site you'll see the following:

In the centre of the screen, under "domain search", you can type in the name of the domain you want to register. Although there are thousands of domain name registration companies around, they're all selling space in the same, single global directory, and you can only register a name if no one else has already registered it. So the first step is to see if our required hamsterwheels.com name is still available. We type hamsterwheels into the domain search box, and the results come back a few seconds later:



It's bad news. The hamsterwheels.com name is already taken, so we can't register it. We could, though, register hamsterwheels.co.uk or hamsterwheels.eu, or any of a whole host of alternatives (many more than fit on the picture above). Alternatively, we could try searching for variations, such as hamster-wheels.com or wheels4hamsters.com.

Those 2 or 3 letters at the end of the domain name are important. They say a lot about your domain. Generally, .com sites are international in nature, or US-based. Sites ending in .co.uk are used by companies which are primarily based in the UK. Similarly, there's a domain for Germany (.de), France (.fr), New Zealand (.co.nz), and every single country in the world, all of which were assigned by an official world body called ICANN, the Internet Committee for Assigned Names and Numbers.

There are of course exceptions to the rule that says most suffixes are country-related. Schools, colleges and universities in the US, for example, have .edu (for education) whereas similar institutions in the UK use .ac.uk (for academia) instead. Charities and non-profit organisations often use .org rather than .com, to emphasise the fact that they're not commercial. There's also .net for internet-related companies, and many more, such as .mobi for mobile communications. But rules are mostly made for breaking, especially in the area of domain names. With a few exceptions (you can't be .gov unless you're a Government department, for example), you can normally choose whatever you want, so long as you can

find a company that will register the name for you. So if, for example, you're based in Germany and you want a .es domain name (for Spain, ie España), there's nothing to stop you doing so.

The only country not to have its own domain suffix, as they're called, is the small Polynesian island of Tuvalu. Having been assigned the suffix of .tv by ICANN, the government of the island accepted an offer of many millions of dollars by a Canadian entrepreneur, for the sole rights to be able to sub-let .tv domain names to television stations around the world. So whenever you see a TV station whose web site ends in .TV, and you wonder why the television industry was given its own internet suffix, the answer is that it wasn't given any such thing.

As you can see, registering a domain name isn't expensive. In the example above, hamsterwheels.co.uk is £5.98, or around US$10, for 2 years. You'll also notice that you can't buy domain names outright. You merely rent them for a period of between 1 and 10 years. When the registration expires, you automatically have first refusal when it comes to renewal. If you renew (which means paying again), the domain name remains yours. If you don't, it goes back into the general list, and anyone else can register it. There is, as you might expect, a steady stream of individuals and companies who attempt to make money by renewing domains that their owners have forgotten about, and then attempting to sell back to the previous owners.

There are pretty strict rules in place to stop you registering a domain name if your primary motive is to trick visitors into believing that you're someone else. If the owners of coca-cola.com forgot to renew their registration, for example, and you registered it for yourself, there's every possibility that Coke would complain to the company through whom you registered the domain and ownership would be transferred. The whole area of dispute resolution is, however, plagued with difficulties. Comrie Saville-Smith, from Edinburgh, an avid fan of the CS Lewis "Narnia" novels, was given the domain name narnia.mobi as a gift by his parents for his birthday. But lawyers acting for the estate of CS Lewis complained, and Comrie had to hand it back.

My favourite domain name dispute of all time happened in the very early days of the Web, when an enterprising young man registered baa.com and set it up as a web site for fans of sheep. The organisation known as the British Airports Authority, then colloquially (and now formally) known as BAA, complained, claiming that this was a blatant attempt to extort money by selling the domain name for a profit, and that the registrant had no real interest in sheep. After a protracted legal battle, BAA won its case and the world's sheepophiles had to find a new home on the web.

Anyway, assuming that we want to register our hamsterwheels.co.uk domain, all that remains is to tick the box on the screen as shown above, provide some valid credit card

details to the company when asked, and the job is done. Within just a few minutes, a confirmation email message will arrive. We are now the proud owners, at least for a year or 2, of an internet domain name.

Having bought the domain name, how can we turn it into a web site? There are two options but, whichever we choose, the first step is always the same.

A web site consists of two things. First, the domain name, ie the www.something-or-other.com name that someone has to type into their web browser in order to get to the site. Second, the site itself, which is frequently (but not always) a collection of HTML files.

One of the most important steps in creating a new site is creating the link between the domain name and the site's files. This is often referred to as pointing the domain name at the site.

The Simple Option

In the simplest of our 2 methods for turning a domain name into a usable web site, we merely register a free account at an existing web space provider such as MySpace, Facebook, WordPress.com, Blogger, Google Sites, Flickr, YouTube, Google Docs, Microsoft's Windows Live Spaces, and so on.

Each of these systems offer you the ability to create content (documents, pictures, text, video etc) that is accessible to everyone (or to whoever you want to make them available to) via the web. You don't need any special software or skills, just a computer with internet access.

For example, in the case of hamsterwheels.co.uk, the easy option for creating a simple web presence would be:

1. Register the hamsterwheels.co.uk domain name
2. Create an account on a blogging site such as www.blogger.com, on which we can create some basic pages. When asked to choose an account name, choose "hamsterwheels". Our blog will thus be assigned a URL of something like hamsterwheels.blogspot.com. We could now simply tell people that the address (URL) of our site is hamsterwheels.blogspot.com. But we have a better plan, which will give our site a more professional feel.
3. Point our newly-purchased domain name at our blogger.com space. This is as simple as logging into the web site of the company from which we purchased the domain name and typing hamsterwheels.co.uk into a box on the "where do you want to point this domain?" page.

4. Now, whenever anyone types www.hamsterwheels.co.uk into their web browser, they will end up at our new blog.

Having your own domain name for your online presence (eg www.hamsterwheels.co.uk) rather than relying on the one that gets generated for you (ie hamsterwheels.blogspot.com) gives you a valuable amount of extra control that would otherwise not be available to you.

If you tell all your friends and customers that your web site is at hamsterwheels.blogspot.com, what happens if you subsequently decide to move your pages to wordpress.com or MySpace? Answer: Your site will effectively disappear because existing users will no longer be able to find it.

However, if you've told everyone that your site is at www.hamsterwheels.co.uk, you won't have this problem. For as long as the content is held on blogger.com, that's where your visitors will end up when they type your URL into their browser. When you move the site elsewhere, the visitors will end up at the new location instead. All you need to do is spend a couple of minutes pointing your domain at the new location, and everything then happens automatically.

So, if you're tempted to start setting up a web site in a low-risk way, and simply get some free space on a site such as MySpace or Blogger, it's a really good idea to keep the myspace.com or blogger.com address private, and register your own domain name for it instead. If you do this from the start, it'll only cost a few dollars a year but will mean that you retain the freedom to move your hosting whenever you wish, without the risk of confusing or losing your existing visitors.

On page 33, once we've registered our domain name, I'll show you how to point it at your page on Facebook.

The Flexible Option

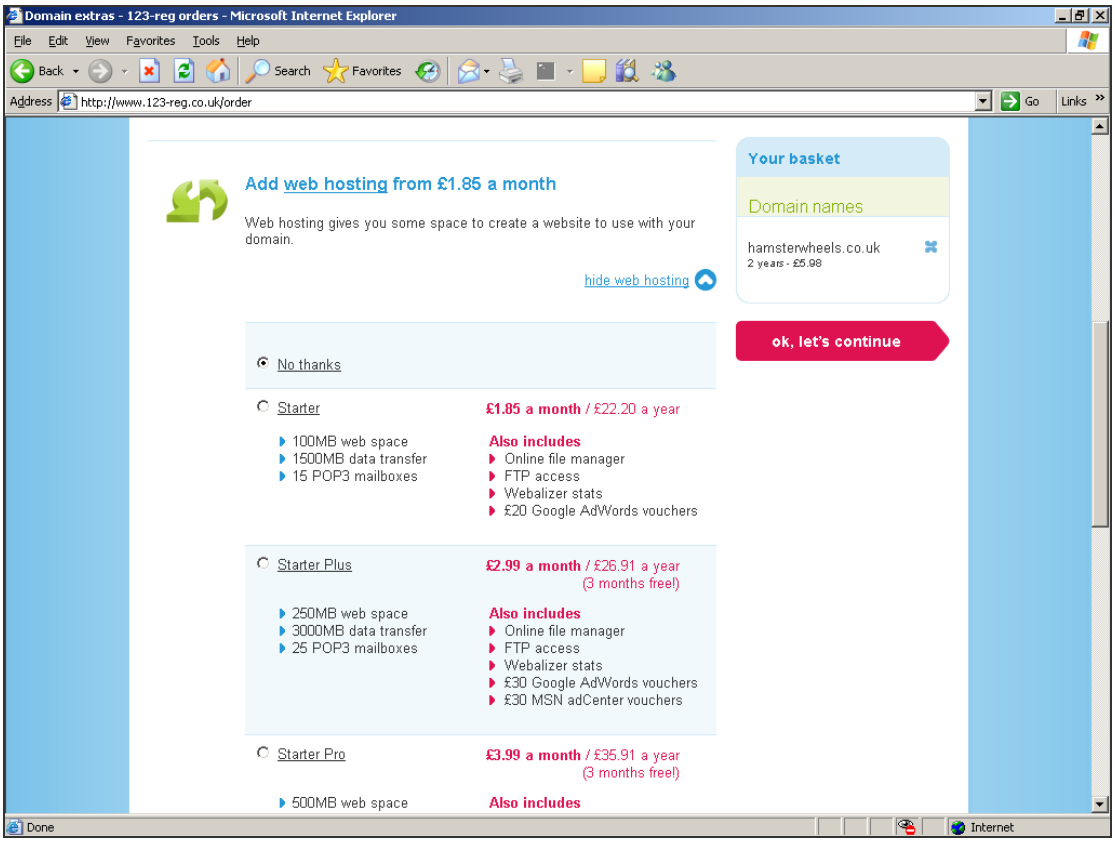
Instead of pointing your new domain name at your area on a site such as MySpace or Blogger, the "grown-up" option is to create a fully-fledged web site of your own and to point the domain name there instead. To do this, you need to rent some space on a public web server in order to host your site, ie to store your pages.

It generally makes sense to buy your hosting space from the same company that rents you your domain name. Although it's not essential, it does keep things simple. Whenever you rent a domain name, the registration company will also ask you whether you want to buy some hosting space. Simply tick the "yes" box and a few more pounds or dollars will be automatically added to your bill. It's not particularly expensive. Renting the domain name

will be around $5 per year, and the hosting space will be roughly $5 per month (generally payable annually in advance). You won't be able to run the next Facebook.com on a hosting service that costs $5 a month. But for a typical site that attracts a few thousand visitors per week, a $5 plan should be plenty. If it turns out not to be, you can always upgrade later.

For what it's worth, the site from which you downloaded this book runs on a basic hosting plan costing roughly $5 a month, and copes just fine.

Here's a screen shot from 123-reg.co.uk, showing the options available for hosting packages to complement our domain name registration. As you can see, it's not expensive. For less than £30 a year, or around $50, you can set up a web site from scratch, including the domain name and the hosting. Just add some content.



When you're buying hosting, the main things to look for are:

1. How much storage space for your site's content do you get? Some companies only give you a measly 100 MB or so, while others offer up to 10 GB or 10,000 MB. Others offer unlimited storage.
2. How much data transfer do you get? That is, the traffic that flows between your web site and your visitors' computers. If you have a page on your server that takes up 100 KB of storage, and 750 people look at the page each month, that's 75000 KB or

75 MB of traffic each month. If the page contains a picture in JPG format that's 210 KB, that's an additional 210 KB of transfer each time. So you'll want to make sure that the amount of data transfer offered by your hosting company is sufficient for your needs.

3. What other facilities do you need, and are they included in the price? In our case, for developing the sites covered in this book, we're going to need the PHP programming language and the MySQL database server. Most hosts offer this, but not all of them do. And in some cases you have to pay extra (maybe $1 or 2 per month).

As you'll see, hosting companies offer dozens of additional features that may or not be useful to you. Some offer 100 FTP accounts, for example, allowing up to 100 different people to be able to upload pages to your site, each with their own username and password. Unless you're a multinational organisation, this is unnecessary.

Equally, every hosting company will offer you a number of email inboxes associated with your new domain name. In my opinion, it's easiest not to bother with them. Instead, create one or more email forwarding entries instead (these are free), to forward incoming email from your domain to, say, hotmail or gmail. Or to your existing email account.

Assuming you get your hosting space and your domain name from the same company, the process of setting up a Web site from scratch goes like this:

1. Register the domain name online, and tick the box to say that you want hosting space too.
2. You'll receive a confirmation email that contains two sets of important information. First, how to log into the domain name control panel in order to point the domain name at your hosting space. Second, the necessary passwords etc that you'll need in order to upload the HTML files, pictures, and all the other files that comprise your new site, onto the server space that you're renting. I call those passwords etc the "Fundamental Four", for reasons that will be explained shortly.

As for the information on how to point your domain name at the server space, you can ignore it for now. Because you bought the domain name and the server space from the same company, this step will already have been done for you. All you need to do is to start creating some content for your web site and then upload it to the server. As soon as you start to do this, your web site will begin to take shape. And in the remaining chapters of this book I'll show you how to do just that.

As for those fundamental four pieces of information that allow you to upload content to your hosting space, what are they? They comprise:

1. The address of the web server
2. A username for connecting to the web server
3. A password to accompany the username
4. Details of which particular folder or directory you need to upload your files into.

About Web Content

Having rented some space on a web server, you can now get to work on the ongoing task of creating, and maintaining, the content of your site. There are 3 ways to do this.

1. Install some ready-made software

There are plenty of web site management systems available, both commercially and as free downloads, which you can install on your new hosting space and which will fulfil some or all of your needs. If you want a web site that works like a blog, which you can log into and post new content via a web browser, you need to be looking for a CMS, or Content Management System. In the world of free software, the leaders in this area are Joomla, Drupal and WordPress. Just download the software from the web, install it on the hosting space that you are renting, and you're ready to roll.

If you want additional features such as picture galleries, real-time chat, etc etc, there are add-on modules for most CMSes that provide such things.

If this sounds like the way you want to move forward, you may wish to skip the next few chapters and go directly to page 118.

2. Develop a Database-Driven Site with a WYSIWYG Tool

The second way to create a working site within your hosting space is to use a product that lets you use drag-and-drop, and point-and-click features to design your own CMS-based system, which the tool then turns into a number of custom-written programs and databases for you to upload to your hosting space. If you want to go down this route, the leading commercial product is Adobe Dreamweaver, which is supremely powerful and correspondingly expensive. If you want do build your web site in this way, much of what follows is not for you. Instead, go buy a Dreamweaver book, or look on Adobe's web site for some tutorials.

3. Create The Site Yourself

The third option, and the one which gives you the most power and flexibility, is to create the web site yourself. This means starting off by creating a few simple pages using HTML, and

subsequently progressing to using database techniques with PHP and MySQL. If you choose this method, you can make your web site do anything you want. It's the way that most professional web developers work.

Don't worry if you're not an experienced programmer or designer. The programming side, with PHP, isn't particularly difficult. And there are loads of great web sites with ready-made page designs that you can use for free.

If this sounds like the way you want to do web stuff, everything that follows is most definitely for you. We'll start with the basics of HTML and CSS to create simplew web pages, and then move on to techniques such as web-based programming with PHP in order to create dynamic Web-2.0 online applications.

Do you need a development server?

In order to create web sites, you need to store the content of your site on a web server. If you're renting some hosting space, you already have a perfectly usable web server for your site. So, is it OK to use that server during the development process, for testing your site as it takes shape and for storing pages that may not be quite finished?

Chances are, yes it's perfectly OK. There's no harm in using your "live" hosting space, or part of it, as a test bed for an unfinished site. And there's certainly no need to set up a completely separate web server, accessible only to you, for development purposes.

However, setting up a test server might occasionally be required. For example, if you're working on a new business idea that you hope will make you rich, you really don't want half-finished pages appearing on a publicly accessible site before the project is officially launched. Also, if you're working on a particularly complex piece of programming, which might crash the server if it doesn't work, you don't want to risk damaging a working site that is already being used by existing clients.

In such cases, you might want to consider setting up a test web server for development purposes. Also, setting up such a server is a fun thing to do, which can teach you a lot about how web sites and the internet work.

To find out how to set up a test web server, step by step, refer to Appendix A on page 319.

Getting Everything Together

You should now know all about domain names, web hosting, HTML files, and uploading. With this knowledge, you can now create your first web page. This means creating an HTML document file and uploading it to the hosting space on a web server.

Throughout this book, for all of the example web content we create, we'll need some hosting space on a server somewhere. As I mentioned, there are thousands of companies that will rent you space on their servers. In this book we'll use a US-based organisation called hostmonster for the examples. If your hosting is with someone else, that's no problem – they all work in roughly the same way.

So having acquired some server space, what's next? To create web pages and upload them to a web server you'll need some suitable software installed on your computer. First, you'll need an HTML editor. This is a program which works much like a word processor, but which saves the finished document in HTML format (ie, as a web page) rather than as a Word or RTF file. Second, you'll need a program which can upload (ie, copy) HTML files across the internet from your computer to the web server. The standard method for doing this is called the File Transfer Protocol, or FTP.

There are many HTML editors on the market. Some are expensive, complicated commercial offerings such as Dreamweaver from Adobe or Expression Web from Microsoft. There are also some free ones, such as nVu, Amaya and Kompozer, which you can download from the internet. Similarly, there are lots of commercial FTP programs such as ws_FTP (the market leader), and free ones such as FileZilla. Most commercial HTML editors, such as Dreamweaver, also have FTP capability built in, but the free ones tend not to.

If you subscribe to a computer magazine that has a monthly CD or DVD on the cover, check your collection of discs. You may find that there's one or more HTML editors on them, which you could consider using.

Microsoft and Adobe offer time-limited trial versions of their software which you can download from their respective web sites. While products such as Expression Web and Dreamweaver are certainly powerful, they're also confusing for beginners so I'm not going to suggest you use them for now.

If you already have an HTML editor or an FTP program installed, feel free to use it for all of the examples in this book. If you don't have one, I'll assume that you'd rather use a free one than have to buy something. That goes for all the software mentioned in this book. Everything that we'll be using is completely free.

For editing our HTML files we'll use Amaya and for our FTP uploads we'll use FileZilla. These are, in my opinion, the best of the freebies.

So before you progress to the next chapter, here's what you need to do.

1. If you haven't already registered a domain name and bought some hosting space for it, now is the time to do so. For the examples in this book we're using the services of www.hostmonster.com, but other companies are also available. A Google search for "web hosting" will get you started.
2. Make sure you have the details to hand of how to access your hosting space. You should know the address of the server, the folder to upload files to, and a username and password.
3. Download and install Amaya on your computer. This is your HTML editor.
4. Download and install FileZilla on your computer. This is the FTP program for uploading finished pages to the server.

The following chapters will explain how to do all this.

It's assumed that you're working on a Windows computer, using XP, Vista or Windows 7. Apologies to those of you who prefer Apple Macs, but this book isn't compatible with your alternative lifestyle and I have no plans to produce a Mac version.

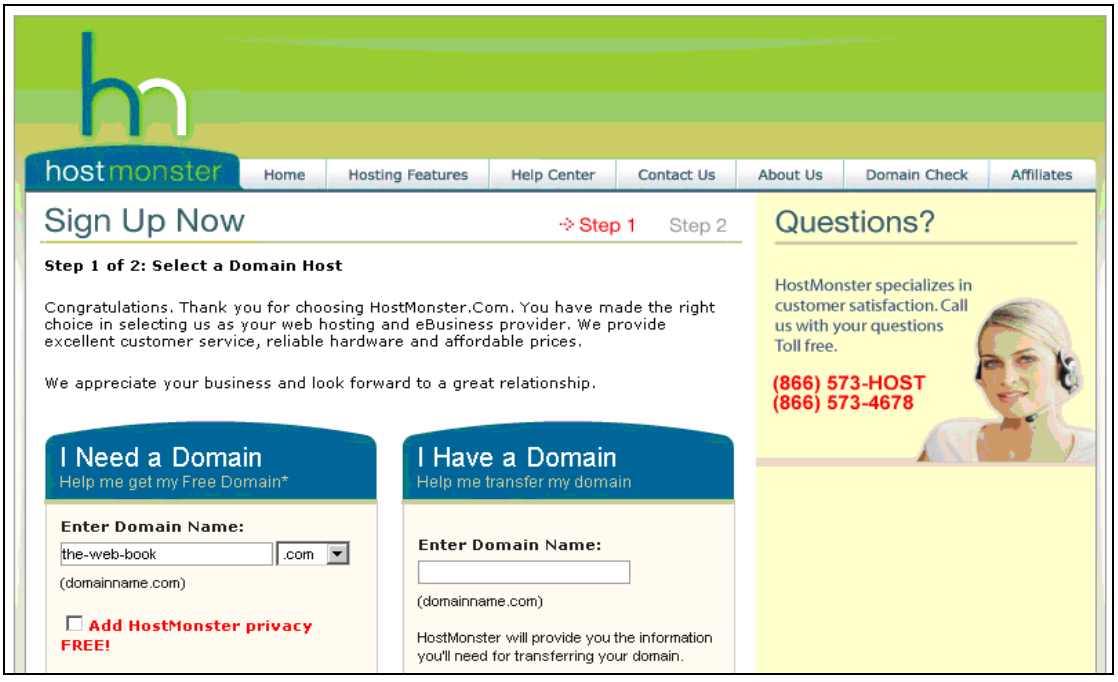
Domain Name and Hosting

The first stage in creating a new public web site is to register the domain name and rent some hosting space, so let's do that now. Let's sign up at www.hostmonster.com, and register the domain called www.the-web-book.com. That's the domain that I'll be using for all the examples that follow. Obviously you'll need to substitute the-web-book.com for whatever domain you decide to register.

Incidentally, a minor technical point that's worth knowing at this point. We won't actually be registering **`www.the-web-book.com`** but merely **`the-web-book.com`**. That's the way that DNS works. You register the top-level name, such as **`the-web-book.com`** or **`fabnews.co.nz`**. How to handle prefixes is technically up to you, but in practice is automatically handled by your hosting company. You could, if you wish, change the prefix from www to web. Or configure things so that **`www.yoursite.com`** ends up at one place but **`web.yoursite.com`** ends up somewhere else. Not that you'd do that, because it would confuse people, but it's possible.

These additional prefixes are called subdomains, and can be useful in some circumstances. The BBC's news pages, for example, are at **`news.bbc.co.uk`** whereas its main site is **`www.bbc.co.uk`**. Creating subdomains isn't something that we'll cover again in this book, but most hosting companies support the facility if you particularly need to do it in the future.

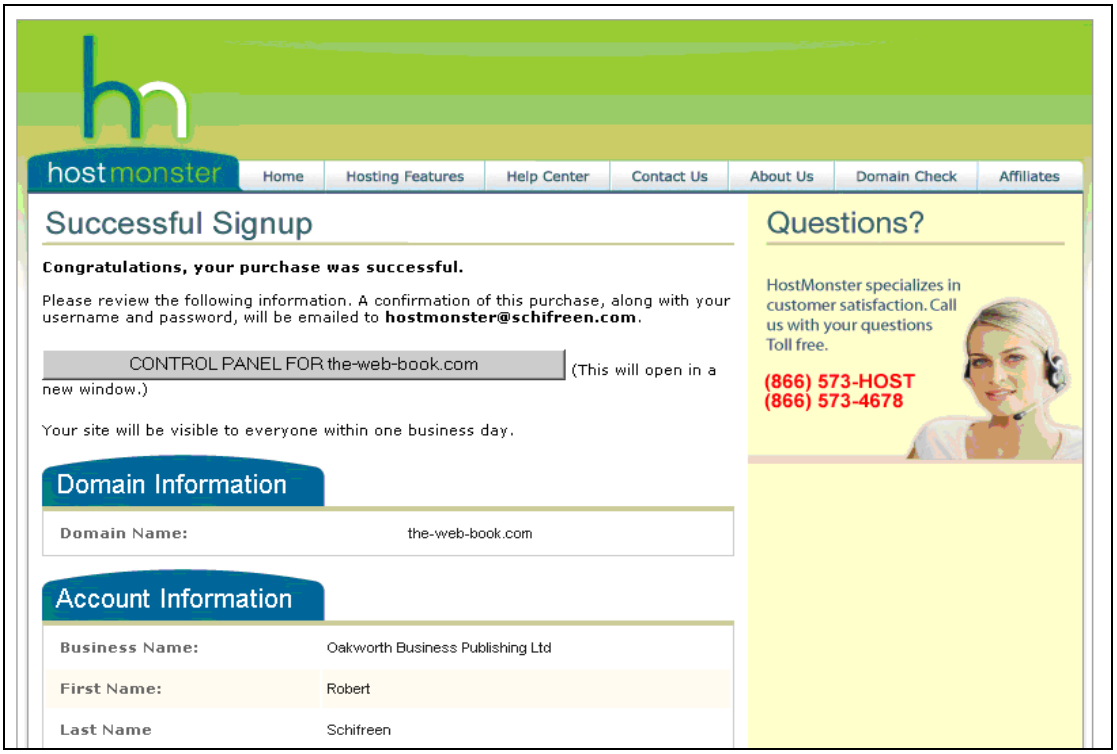
Right, back to hostmonster.com. Let's start by registering our name.



We enter the domain name we want to register, and allow hostmonster to check whether it's available.

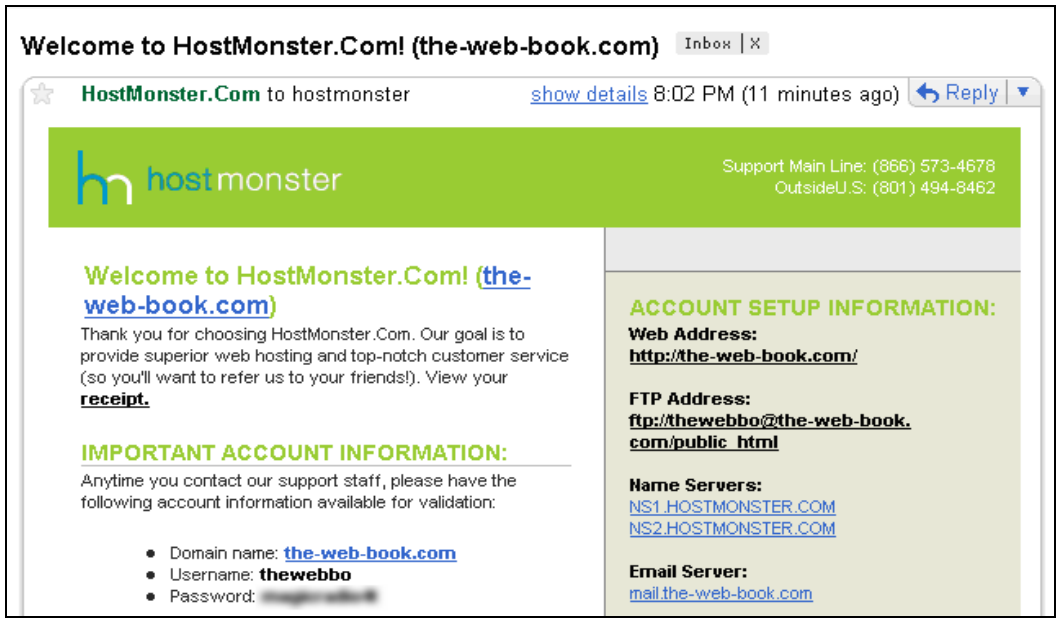
Note the option for privacy. There's a public directory of internet domain name registrations that allows anyone to find out the name and address of the person or organisation which registered any domain. It's known as the "whois" system. By selecting the privacy option, which all domain name registration companies offer in one form or another, your personal name and address are withheld from the directory. Unless you're a large company, you'll probably want to do this.

Our domain name is available, so let's sign up and buy it, along with the basic hosting package so we get some space on the server for our web content. We'll need to supply our credit card details and an email address, and also choose a password for logging into the control panel.



And we're done. That's all there is to it. We've now set up the required infrastructure for a brand new web site. We've got a name, and some space on which to host the site. All we need to do now is wait for the email message with details of how to connect to our hosting space.

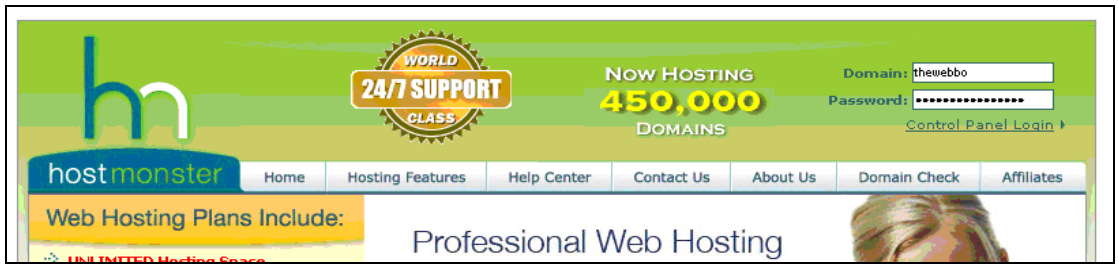
After a few minutes, it arrives. Here's what it looks like:



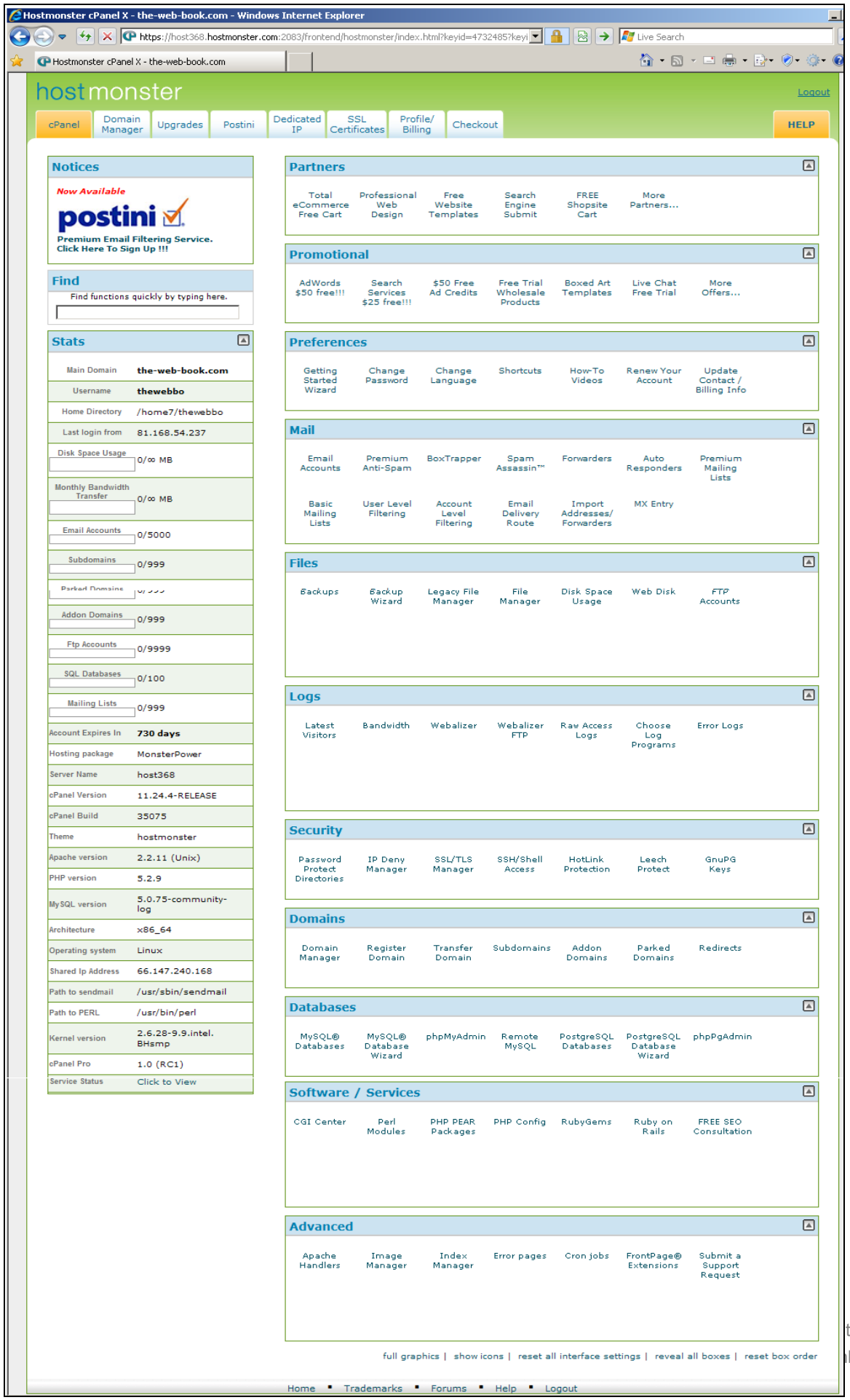
So now we have those "fundamental four" bits of information we need in order to upload content to our new hosting space. The address of the server for FTP uploads is the-web-book.com and the folder is /public_html. The username is thewebbo.

For obvious reasons I've obscured the password. Remember to keep all web hosting passwords safe and secure, and not let anyone else use them. Anyone who finds your password could, in just a few seconds, delete your entire site from the server. Or, worse, replace your carefully-crafted pages with gibberish or obscenities.

Having bought some hosting space, the only way you'll ever log into the hosting company's systems is via FTP in order to upload files. But all hosts also offer a control panel of some sort, where you can maintain your billing information and do all sorts of other interesting things. Let's have a look at our control panel now. To do this, we surf to **`www.hostmonster.com`** and we'll see that there's a control panel login option. We simply log in with our username and password as specified in the email we received earlier:



Having done this, the control panel appears.



If you've not seen a web control panel before, it probably looks rather daunting. Especially when you see that there are 8 tabs/sections and our screen shot shows just one of them. The good news is that you'll hardly ever need to touch it. But the other good news is that, should you ever need to change anything about the way your web hosting works, it can all be done via this control panel. One word of advice, though: each time you change something on the control panel, make a note somewhere. It'll come in useful one day.

Let's take a brief look through some of the most important sections of this control panel. The design of which, incidentally, is not unique to hostmonster. It's a commercial product called cPanel, which many hosting companies buy because it saves them having to write their own.

Stats
The area down the left hand side of the screen is your web site dashboard. It shows you how much storage space and transfer capacity you're using, when the hosting account expires, and so on. There's also a link to the Service Status page. If you ever find that your web site isn't working properly, check this page before assuming that it's your fault. The server may be down for maintenance.

Partners/Promotional
This section shows various additional services that you can sign up for, some of which are free. Feel free to click on the links and explore further. But none of these services is essential, at least for the time being, so don't bother signing up for anything unless you really want to. It'll only complicated things further on.

Mail
From here you can set up the way that email works for your domain. When you buy a domain name, you have control not only over the website at **`www.yourdomain.com`**, but also any email sent to **`anyone@yourdomain.com`**. At some point we'll need to configure this, because we want to ensure that mail sent to, say, **`info@my-web-book.com`** actually ends up somewhere where we can access it. Equally, we want to be able to send mail which appears to have come from **`someone@my-web-book.com`**. But for now, this can wait. We need to get the web site up and running first.

Files
The Files section of cPanel lets you browse and back up the files on your site, ie the HTML documents that comprise your web pages. It's simplest to do this via an FTP program rather than the web control panel, though, so my advice is to ignore this feature.

Logs
It's useful to know how many people have been looking at your web site. All web servers keep logs of such things, and all hosting companies offer you a variety of ways to view those

logs. One of the friendliest log viewers is webalizer which, as you can see, has been installed on the server for us. To view the logs we need do nothing more than click on the link. We'll talk more about logs, and how to market your web site, in a later chapter.

For now, though, click on Choose Log Programs and make sure that Webalizer is enabled. That way, when you come to the chapter on how to look at the stats for your site, there will be some nice Webalizer graphs to look at. If you don't tick this option, there won't be,

Security
These options allow you to add security to your web site. You can add password protection to some or all of your pages, or prevent access to your site from certain IP addresses. *Leech Protect* adds additional protection to password-protected areas of your site, by allowing you to specify how many different IP addresses a particular password is permitted to be used from. This prevents misuse of your site caused by passwords to protected areas becoming known and being distributed among hackers. We'll cover web site security, and how to protect the information on your site, in much more detail in a later chapter. Generally, protecting your web site is something you do within the site itself, rather than from a control panel.

Domains
This area of the control panels lets you manage the domain names that you have registered via the hosting company. You can transfer their registration to a different company if you wish, or move other domains that you own from a different hosting company to this one. You can also create subdomains, as mentioned on page 27.

Databases
In later chapters, we'll set up MySQL databases on the server and then create web sites that can access them. From this section of the control panel you can manage those databases.

Software/Services
This section allows you to change the way that certain key software applications on the server work. For example, you can change the master configuration file for the PHP programming language if you need to tweak certain settings.

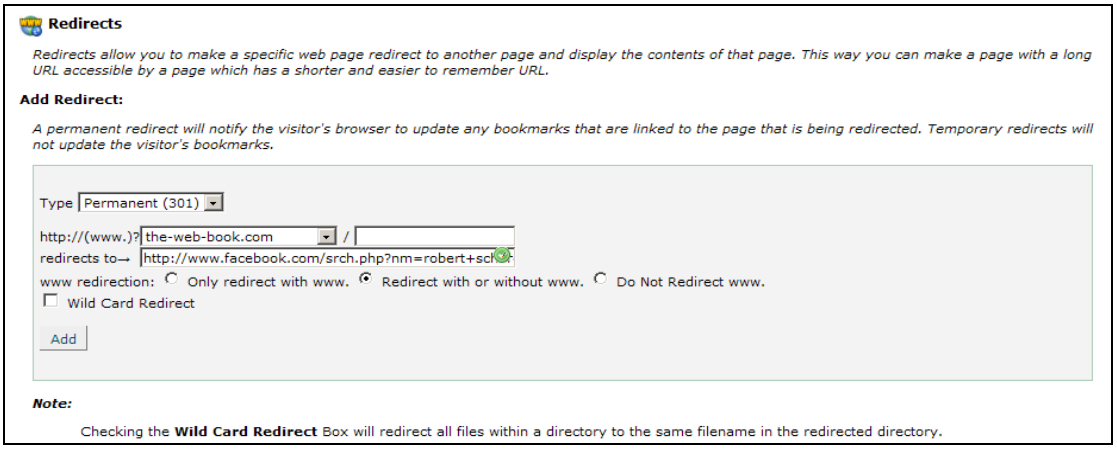
Advanced
Among the options available here are "cron jobs". Most web servers, including this one, use the Linux operating system rather than Windows, and a cron job is the Linux equivalent of what Windows refers to as a scheduled task. This is a way for you to tell the server to automatically run a specific program at specified times throughout the day, week, month and/or year. You might, for example, set up a cron job that runs a PHP program on the first day of every month, which checks the membership database of your web site and sends an email reminder to everyone whose subscription to the site is about to expire.

It's Not Rude to Point

Earlier on, I mentioned that the easiest way to get a professional-looking web presence is to register your own domain name but, instead of buying hosting space and spending time creating pages, just point the domain name at your personal area on a site such as MySpace, Blogger.com, Youtube, Facebook etc. Now that we've registered our domain name, I can show you how to do just that.

First, make sure you know the web address (URL) of the site that you want to point, or redirect, your domain name to. In this example I'm going to point **www.the-web-book.com** at my page on Facebook, to make up for the fact that I haven't created any web content for my new domain yet.

Log into the web control panel. If you're using cPanel, you need the Domains section. Specifically the option called "Redirects", which brings up a page that includes a section which looks like this:



As you can see, I've typed (actually, pasted) the address of my Facebook page into the "redirects to" box. I've also selected **the-web-book.com** as the source.

That's all you need to do. Click the "Add" button, wait a few seconds for your server to update itself, and the job is done. If I now open my web browser and attempt to surf to **www.the-web-book.com**, I actually end up at my Facebook page instead.

Obviously, if you're intending to go down this route from the start, there's no need to pay for hosting space when you register your domain name. Indeed, you can stop reading this book now, as we've covered everything you need to know. However, with so many good offers around for combined registration and hosting, you may decide that it's easier to just buy both at the same time, even if you start off by pointing your domain name at an external site for the first few months rather than creating your own content.

If you really do want to register just a domain name and have no intention of occupying any hosting space, most hosting companies offer this service. They all offer redirect facilities, too, so that you can point your new name wherever you wish.

HTML Editor and FTP Client

With our hosting space now purchased but sitting empty, we need some programs that will let us start to fill it. This means an HTML editor to create pages, and an FTP program to upload those pages to the server.

FTP programs come in two flavours, namely clients and servers. For copying files between 2 computers, you need a client. An FTP server is what the web server runs, allowing clients to connect to it. Don't mistakenly install an FTP server on your computer, as it will allow everyone in the world to attempt to connect to your computer via the internet and copy files to or from your computer.

There are plenty of free HTML editors available for Windows, and the one we'll be using throughout this book is Amaya. It's relatively easy to use and, unlike some of the other free programs, doesn't attempt to install additional spyware or toolbars that you don't need.

There are many free FTP clients available. One of the most popular is Filezilla, so that's what we'll use.

Amaya

To download Amaya, visit **`http://www.w3.org/Amaya`** and follow the Distributions link to reach the download page. It doesn't matter whether you get the http or FTP version – the programs are both the same, the only difference is how your computer performs the download.

New versions of Amaya are released every 4 months or so. The examples in this book are based on version 11.2, released in mid-2009.

Once you've downloaded the Amaya program and saved it to your computer, double-click the file to start the installation process. Keep clicking Next to accept the suggested options, and the installation will complete. The installer will add an icon to your Windows desktop, making it easy to run the program whenever you want.

Make A Web Work Folder

Now is a good time to give a moment's thought about where you want to store work-in-progress web pages on your computer. When you're creating web content, such as HTML files, the normal way of working is to create and edit the files on your computer, and only to upload them to the web server (and thus make them available to the public) when they're

finished. This has a number of advantages. Primarily, it means you always have backup copies of your pages, because there's the version on the server and the version on your PC. Also, it means that information which you haven't finished writing or checking can't be accessed by anyone except you. If your company is about to launch a new product, for example, uploading unfinished, unannounced details on the web site for Google (and thus the world in general) to find is unwise.

You need a folder on your computer to store your web work. For the purposes of this book, I'll assume that this folder is on your desktop and is called Web Work. If you want to create it somewhere else, or give it a different name, that's fine. But if you're happy with my suggestions, right-click on your desktop and create a new Web Work folder now.

Filezilla

Visit **`http://www.filezilla-project.org`** to download the FileZilla FTP client. Choose the option to download the latest Windows version. The examples in this book use version 3.2.3.1, released in mid-2009, but you may find that a slightly later version is available by the time you read this.

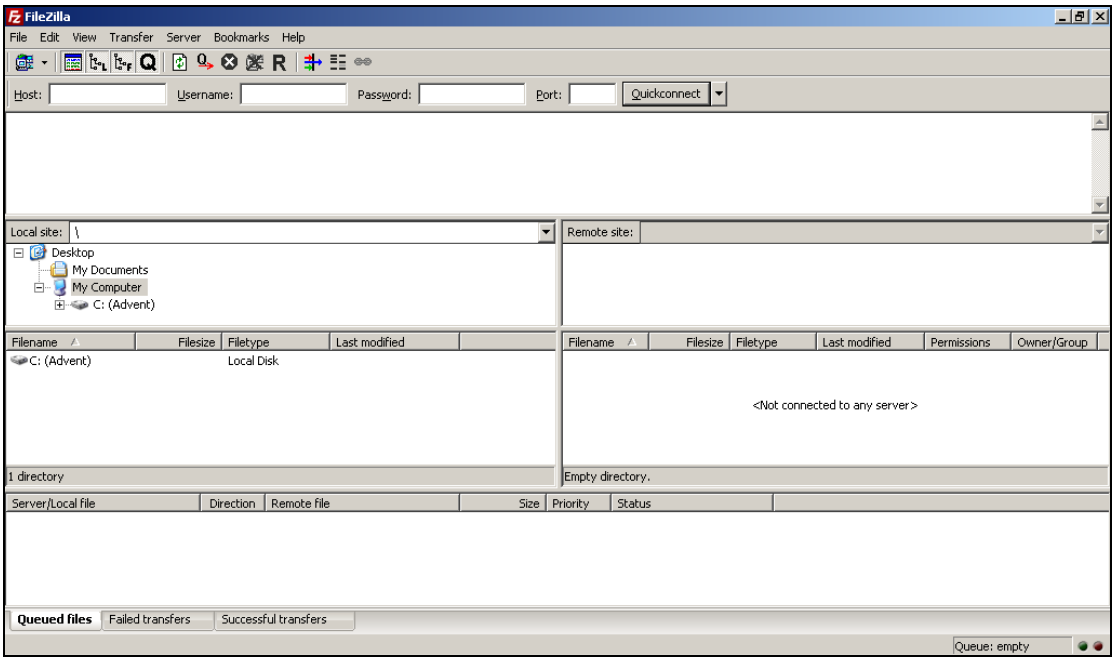
Having downloaded the .EXE file, double-click it to start the installation process. Or if you downloaded the .ZIP file, you'll need to extract it first (right-click the file and choose Extract All), and then double-click the extracted installer file.

You might be asked whether to install for all users or just for you. Unless you specifically don't want to do so, choose the "all users" option. It means that, if you have multiple user accounts on your computer, everyone will be able to use FileZilla. (You'll need to be logged into your computer as an administrator to be able to do this "All Users" installation. If you're not, try the other method.)

You'll then see a list of components that you wish to install, such as icon sets, shell extensions and so on. De-select them all, as they aren't necessary. But you'll probably want to select the "desktop icon" option in order to have the installer create an icon on your desktop.

We now need to configure FileZilla so that it can connect to your hosting space on the web server to upload the files you create. This requires those "fundamental four" pieces of information I mentioned earlier.

If FileZilla isn't already started on your computer, double-click its desktop icon to run it. You'll see a screen that looks something like this.



Note the references to Local Site and Remote Site. The former shows the files on your computer, while the latter shows files on the web server. Or at least, they will when we have configured the program.

From the File menu, choose Site Manager and then click the New Site button. Choose a name for the server you want to connect to, and press Return. Web Work is a suitable choice, or perhaps use the name of the site you intend to create. FTP programs allow you to create multiple connections like this, each with their own name, so that you can upload files to different servers (or different folders on the same server) if you wish.

Having named your connection, look under the General tab and enter the host name. That's FileZilla's term for the address of your web server. In this case, I'll use **`the-web-book.com`** because that's what the account information sent by the hosting company tells me to use. Change the Logontype to Normal and enter your server username (thewebbo in my case) and password.

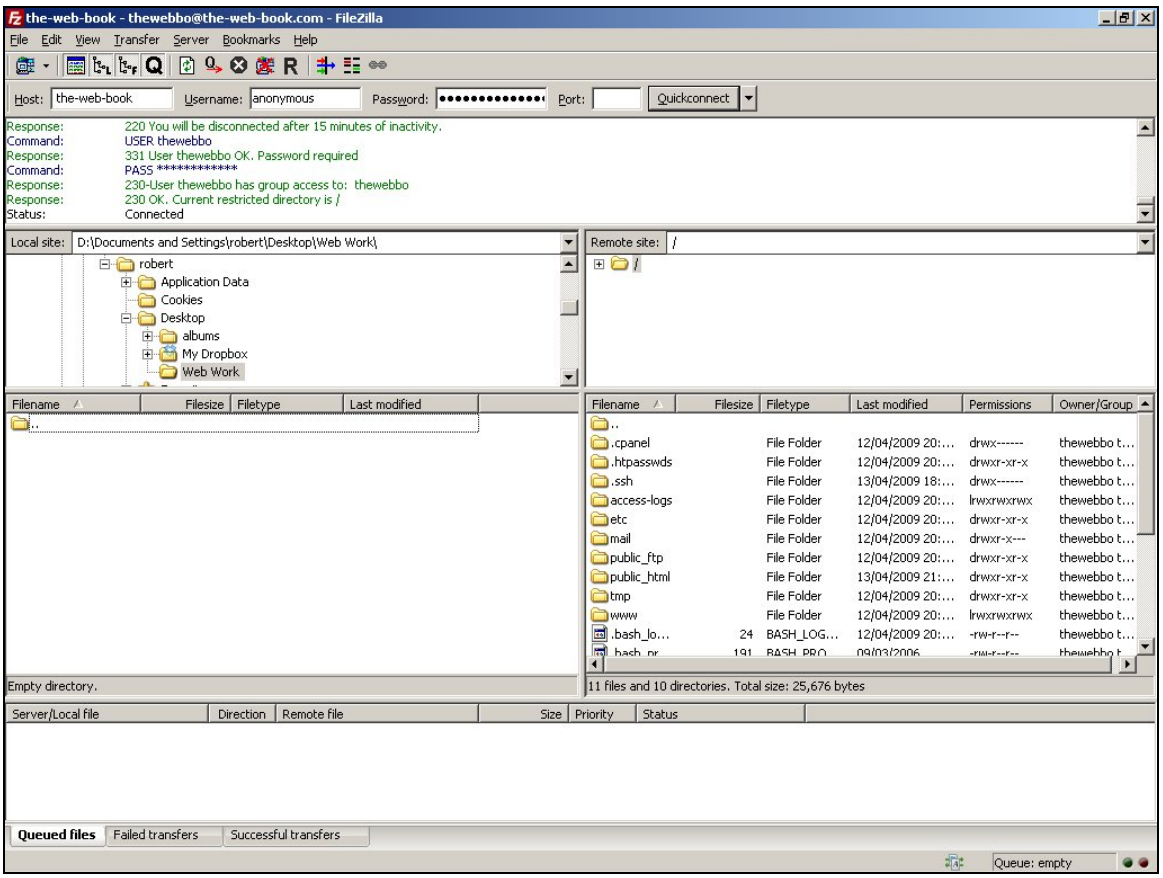
Now click on the Advanced tab. In the Default Local Directory box, click the Browse button and select the Web Work folder you made earlier. For the default remote directory, just type a **`/`** (forward-slash) character.

That's all there is to it, so click the OK button. FileZilla is now configured to connect to our web server.

We can now test that it works so, from the File menu, choose Site Manager again. Click on the name of your server and then click the Connect button. After a few moments, you should

be connected. Depending on how Windows is configured on your computer, you may receive a pop-up warning from the Windows Firewall asking if it's safe to unblock connections to the FTP server. If so, just click on Unblock.

Assuming that FZ can connect to your server, you should see something like this:



On the left hand side of the screen, under Local Site, you can see the local folder on your computer which we designated to hold our HTML files. In this case, it's the Web Work folder, which currently has nothing in it because we haven't yet created any HTML files.

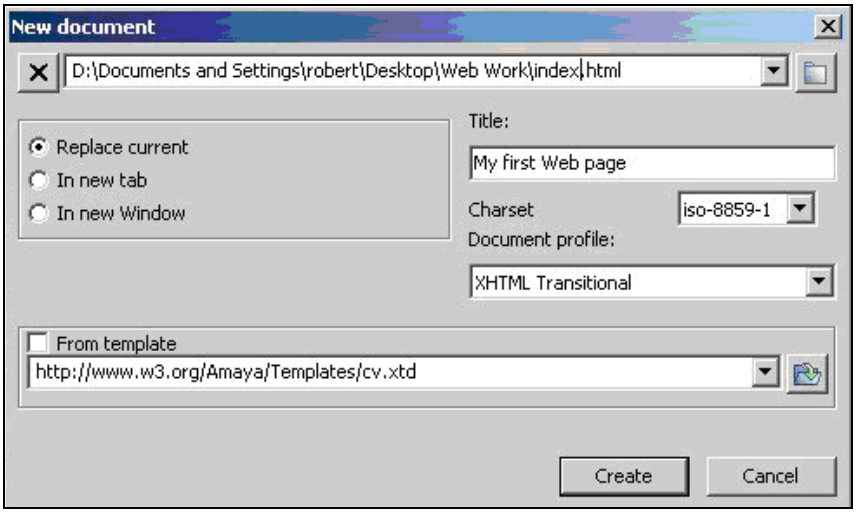
On the right, the Remote Site column shows the folders and files on your server. You'll see various folders such as .cpanel, .htpasswds, .ssh and others. The folder that is of primary interest at this point is public_html. This is where you will put all of the content that makes up your web site. All of the other folders are for use by the web server only, and are not accessible to the general public via your web site. For now, it's safest to leave them alone and not be tempted to explore or delete them.

With an HTML editor and an FTP client now duly installed and configured, we're finally ready to create web pages. So close FileZilla and we'll get started.

Creating Your First Web Page

Time to create a web page. Start Amaya, and close the tip of the day if one gets displayed.

From the File menu, choose New Document and you'll see a box something like this:



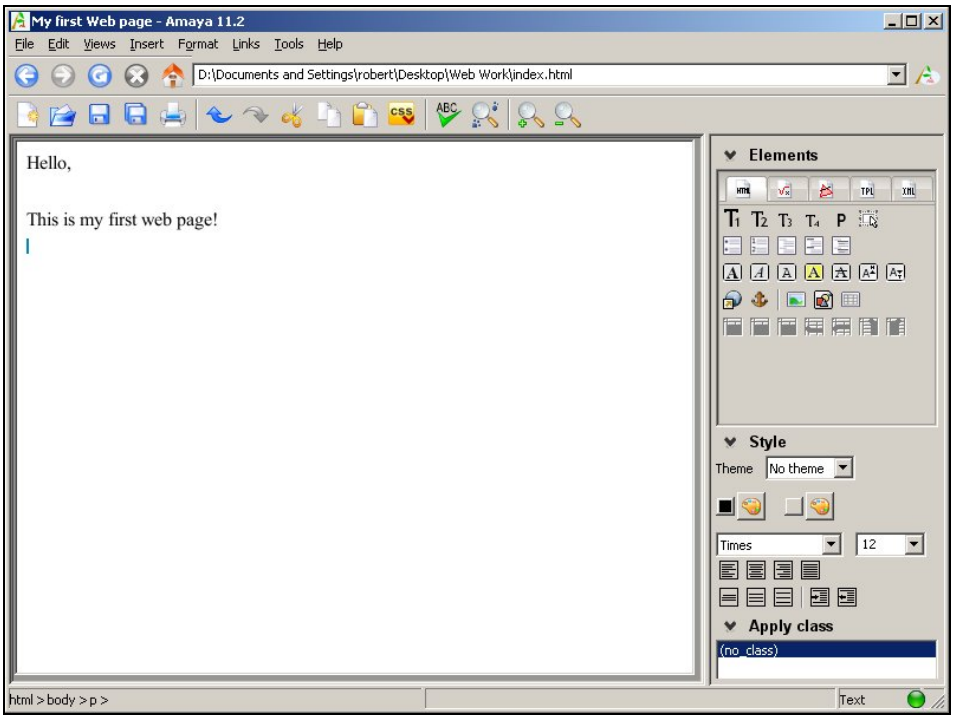
Click the folder icon near the top right hand corner, browse to your Web Work folder and click OK. Amaya will now generate a filename for your new page that ends with New.html, but this isn't what you want. So change it to index.html, as illustrated above.

Why index.html? Because if someone visits your web site and doesn't specify a particular page that they want to see, the server will show them the home page. And the way that you tell your server which is the home page, is by naming it index.html.

If someone visits your site and explicitly types **`www.yoursite.com/products.html`** into their Web browser, the server will send them products.html. But if they merely type **`www.yoursite.com`**, the server will attempt to send them index.html. And if index.html doesn't exist, the visitor will see an "Error 404 – page not found" error message. So, you always need to start with index.html when you're creating a new site, or any new folder in an existing site.

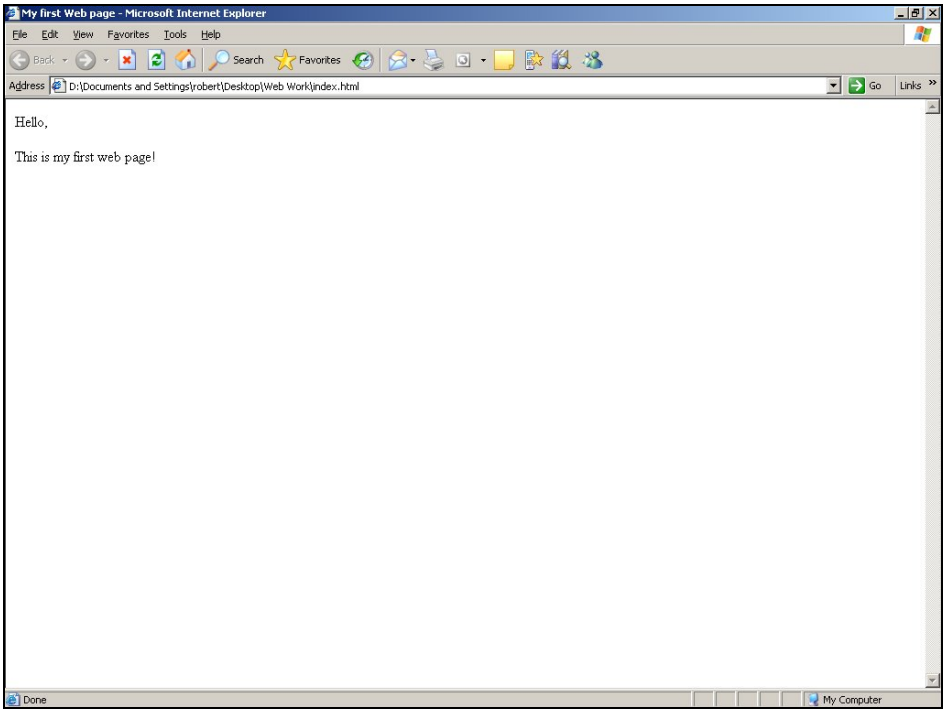
Incidentally, you may have noticed we named our page index rather than Index or INDEX. This is important. Generally, you should always name web pages in lower case, without any CAPITAL LETTERS or spaces. That's because most Web servers are case-sensitive, and regard Index.html and index.html as two separate files. To avoid confusion caused by pages not being found because they were typed by the visitor (or specified in a hyperlink) in the wrong case, it's best to standardise on everything being in small letters.

So, we've named our page. Now choose the "Replace Current" option, give your page a title, and un-tick the From Template box. Then click Create, and go ahead and type some text onto your page. It'll look something like this:



When you're finished, click File/Save, then File/Exit Amaya.

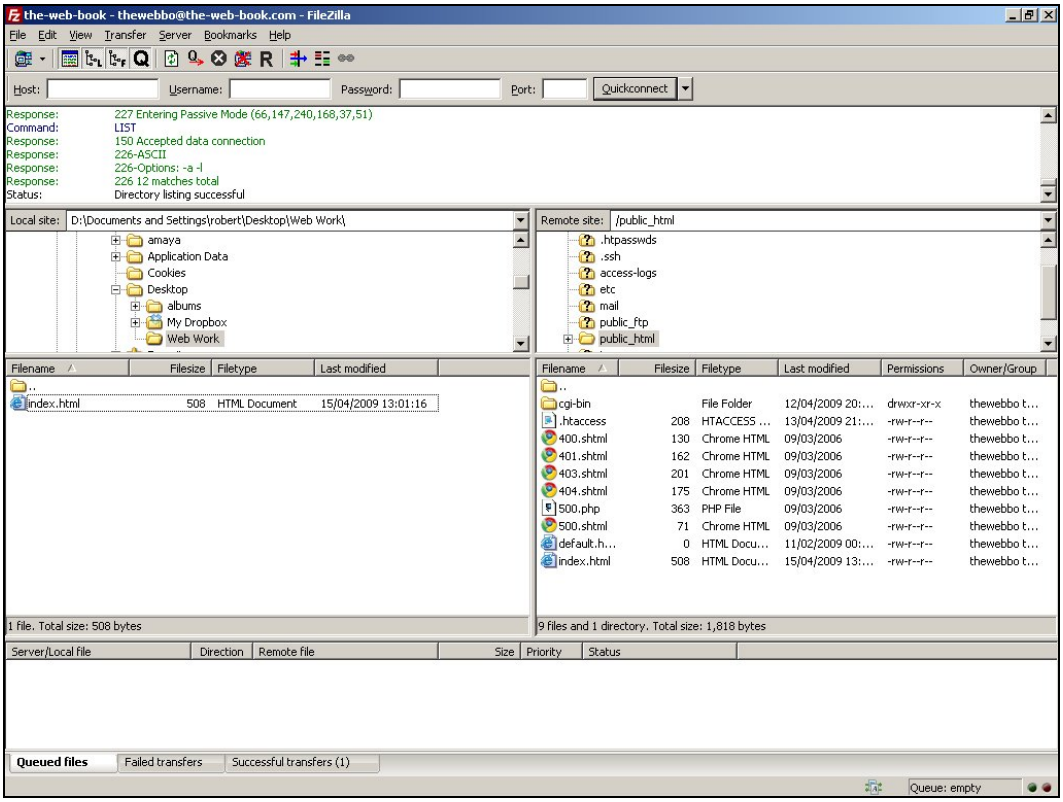
We've now created an HTML file, which should be in your Web Work folder. So double-click the folder and then double-click the index.html file that's in there. The file will open in your web browser and should look like this:



We're nearly there. The page is on your computer, but it's not on the web server, so only you can see it. The final step is to use FileZilla to upload a copy of the HTML file to the web server.

Open FileZilla and connect to your server (from the site manager, click your site and then the Connect button). On the left hand side of the screen, in your local area, ensure that index.html in the Web Work folder is listed. On the right hand side, in the Remote Site area, double-click the public_html folder to open it.

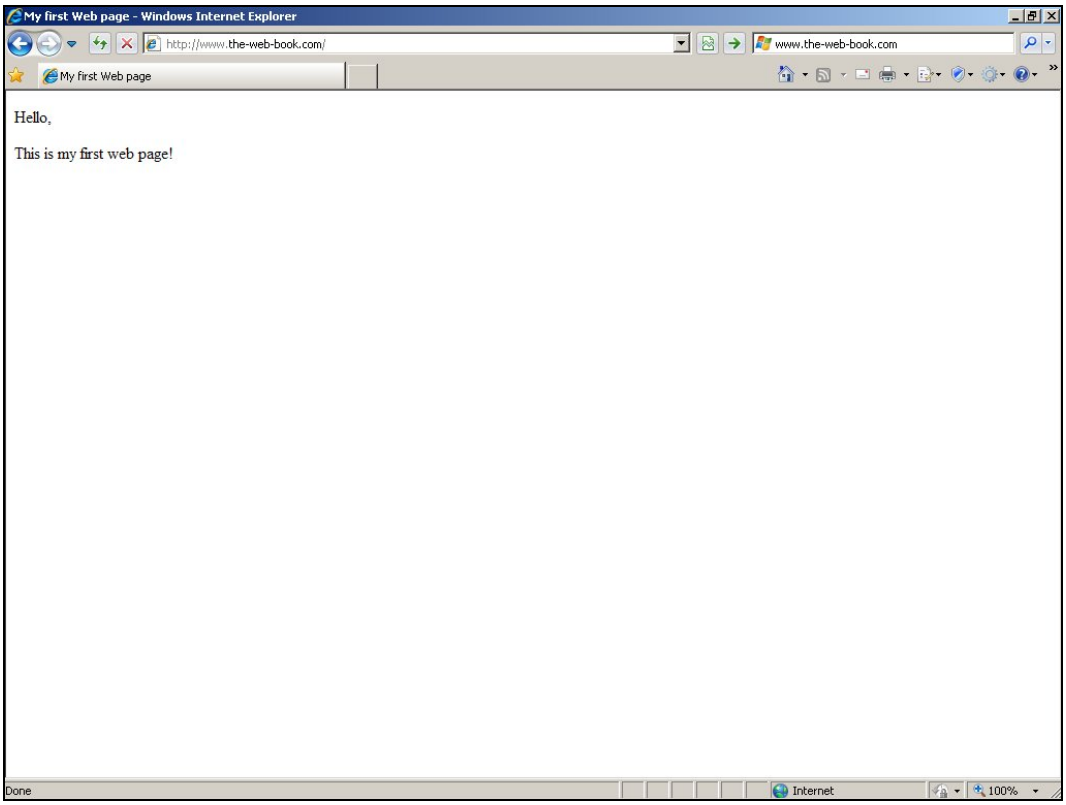
Now double-click the index.html file. FileZilla will copy it from the local folder to the remote one. You should see it listed among the contents of the remote site:



There it is, right at the bottom of the list, after assorted files such as .htaccess, 400.shtml and others.

If your hosting company puts any folders or files in your public_html folder, it's generally quite safe to delete them (albeit after a cursory check to ensure they're not vital) if you don't know what they are. After all, your public_html folder is your public web site, and anything in there is accessible to anyone in the world who types the correct filename and site name into their web browser. For now, though, you can rest assured that these particular files are harmless so don't do anything with them yet.

With your index.html file uploaded, you now have a real, live, publicly-available web site. At least, that's the theory. To find out for sure, close FileZilla and open your web browser (Internet Explorer, Firefox or whatever you prefer). Type the URL of your site (www.the-web-book.com in our case here) and you should see your page like this:



We've done it. A working web site, hosted on a public server, available to the world! Now's the time to mail all your friends with the URL so they can admire your efforts.

Now step away from the computer!

Once you know the basics of creating web pages, it's hugely tempting to plough straight ahead and start creating a complete site. Although that's almost certainly what you want to do right now, it's a bad idea for two reasons. First, although you know the basic method for creating pages, there are lots of time-saving techniques that we haven't covered yet. Much better to know about them first, before it's too late to go back and start using them. Second, there's more to creating web sites than simply typing text into an HTML editor and hitting the Save button. You need to give some serious thought to the design and usability of the site, how you're going to display the text, how you're going to import existing content from other documents, and, most importantly, who and what your site is actually for.

So if you're tempted to skip this chapter and get on with building your site, please don't. If you stop reading now, and start creating your site, you'll end up with lots of pages that are very difficult to update and which don't conform to recommended standards. By all means create a few test pages, and feel free to keep on experimenting, but please don't assume that there's nothing else important left to learn.

At the very least, please don't start creating a site for real until you've read:

1. The remainder of this chapter.
2. The chapter about CSS, which starts on page 59 .
3. The introduction to Content Management Systems (CMSes) which starts on page 118.

WWW – What, Why, Who?

Before you start to create a site, it's vital that you have clear in your mind:

1. What type of site you're creating.
2. Why you're creating the site.
3. Who the site is for.

What type of site are you creating, and why? Is it an information site, where people expect to quickly find what they're looking for and then leave? Or is this an entertainment site, where visitors browse and read with no particular purpose, and won't complain if it takes a while to find what they're looking for among all the pretty graphics and animations?

This matters because it will affect the layout of your pages. If you want people to quickly find what they're looking for, your menus need to be clear and concise. Preferably stick with just one (or a maximum of 2) menus on each page. If a visitor is looking for your contact details, they shouldn't have to scan 6 different menus. Either the item is on the menu or it's not, and this should be possible to discover almost instantly.

Equally, keep your menu choices unambiguous and mutually exclusive. For example, if your site is all about vehicles, don't have a menu which contains items called "vintage vehicles" and "buses". Or if you're writing the site for your company's HR/Personnel department, avoid having menu options called "new staff" and "for managers". You'll merely confuse the visitor who's looking for vintage buses, or the person who's a newly-recruited manager. A confused visitor won't spend time exploring the options – he'll simply try someone else's site instead, or give up entirely. Therefore, spend some time planning your menu headings and hierarchy so that the choices are obvious.

The best way to plan your layout and menus (navigation) is with pen and paper, or a whiteboard. There's no need to use a computer at this stage, so put away the PC and start scribbling. When you have some rough designs, create some PC-based mockups and show them to a few people for comment. Ideally, those people should be potential users of your site.

There's no need to create the mockups as proper HTML pages. Use a graphic editor such as Windows Paint, or a spreadsheet, or whatever else you want at this stage. The pages don't have to function properly. If you'd rather use a computer to create mockup pages, rather than pen and paper, one excellent program is called Mockups, from Balsamiq Software. You can download it for around $50, or use it online for free. Check out **`www.balsamiq.com`** for more.

Put page content where visitors expect to find it. A logo at the top of the page. Menus on the left or across the top. Main content to the right. Don't set out to be different – you'll only end up being frustrating.

Importing Existing Content

HTML is the primary document format used for Web pages. It's the only one that every Web browser can be guaranteed to read, whether that Web browser is a PC, Mac, mobile phone, iPod, PDA, Blackberry, television, or any other device that has built-in internet access.

All of which leaves the question of what to do with all those existing Word documents, Excel spreadsheets, PowerPoint presentations, PDF files and so on, that you'd like to make available online via your Web site.

Generally, you have two choices. The first option is to upload the files to your site, just as you would with any other file such as an HTML document or a css file. You can then provide a link to the .DOC, .XLS, .PDF or .PPT file just as you would to any other page on your site. However, there's no guarantee as to what will happen when the visitor clicks on the link. It will depend greatly on which Web browser the visitor is using, whether he left- or right-clicks on the link, and what software is already installed on his system.

Among the various things that might happen are: the browser will open the file and make a stab (sometimes a pretty good one, sometimes not) at displaying it; the browser will pass the file to another program on the computer, which will open and display it; the browser will complain and do nothing; the browser will offer to save the file to the visitor's hard disk.

Despite these potential problems, not to mention the fact that non-HTML files aren't as easy for search engines such as Google to find, uploading non-HTML files to Web sites is still common practice and may sometimes be the only feasible option. Or at least, the only option that you have time to implement.

The second option is to convert the document to a "proper" Web page, ie to an HTML file. This means starting from scratch, copying and pasting unformatted plain text from the non-HTML file into your HTML editor (eg Amaya) and then formatting it. If you have the time, this is by far the better route.

One technique which you should definitely avoid is the "save as web page" feature built into programs such as Word and Excel. Although this will produce an HTML file, the conversion process is inefficient at best and downright appalling at worst. The resulting file will be many times larger than a version created by copying and pasting, and probably totally unintelligible. Which makes it hard to maintain, and totally impossible to apply CSS styling to. This is a classic example of a short cut which will lead to major problems in the long term. Do not be tempted to go down this route. There's more about this on page 13.

Writing For The Web

Writing Web pages isn't like writing printed documents, because people read Web pages in a different way. For a start, if a Web page doesn't provide the reader with the information they're looking for within 5 seconds, they'll give up and use Google to find a different site. Here are some tips to consider when you're writing a new Web page or editing an existing one:

1. Think about who you're writing for. What sort of audience? Young or old? Technical or non-technical? Are they looking for information or entertainment? Do they want a plain, easy-to-read design, or would something more arty appeal?
2. Don't start with a long "welcome" intro, saying what the page is for. The visitor will quickly skim-read the page and make up their own mind as to whether it seems suitable. The quicker they find out, the more chance of them staying around.
3. Don't assume that everyone reading your Web site will have an enormous screen or a fast internet connection. They may be browsing on a mobile phone or PDA, where the screen is small and every minute online costs money.
4. Keep paragraphs short and columns narrow. It's much easier to read this way. Also, don't pack too much information on each page. Users prefer to click rather than scroll.
5. Line spacing of 1.5 or 2 is good. Small text with 2.0 line spacing is easier to read than large text with 1.0 spacing.
6. Never use blue or underlined text. People will think it's a clickable link.
7. Stick to the various conventions, such as a click on your main logo taking readers to the home page. Someone looking to contact you will be seeking a "Contact Us" button, not something that says "Get in touch".
8. Don't arrange text in columns such that the reader has to scroll the screen up and down. It's a Web page, not a newspaper.
9. Don't waste too much above-the-fold (the area of the screen that can be read without scrolling) with waffle or large pictures.
10. Check the spelling of your pages, but don't rely solely on any automatic spelling checker.

Fonts and Colours

Fonts (typefaces) are important. Sans-serif fonts tend to work best on screen, rather than serif fonts. But don't use non-standard fonts that your readers are unlikely to have installed on their computer. You can read more about fonts on page 67.

Readers will use your site's colour scheme to form a subconscious opinion about you. When did you last see a bank's Web site that was bright red and yellow? When did you last see a hamburger restaurant site that was dark blue? Choose appropriate colours for your pages, but don't go overboard. And don't force people to rely on colours, such as referring in your text to "the yellow box", because the visitor might be colour-blind or might be reading your page on a black and write printout.

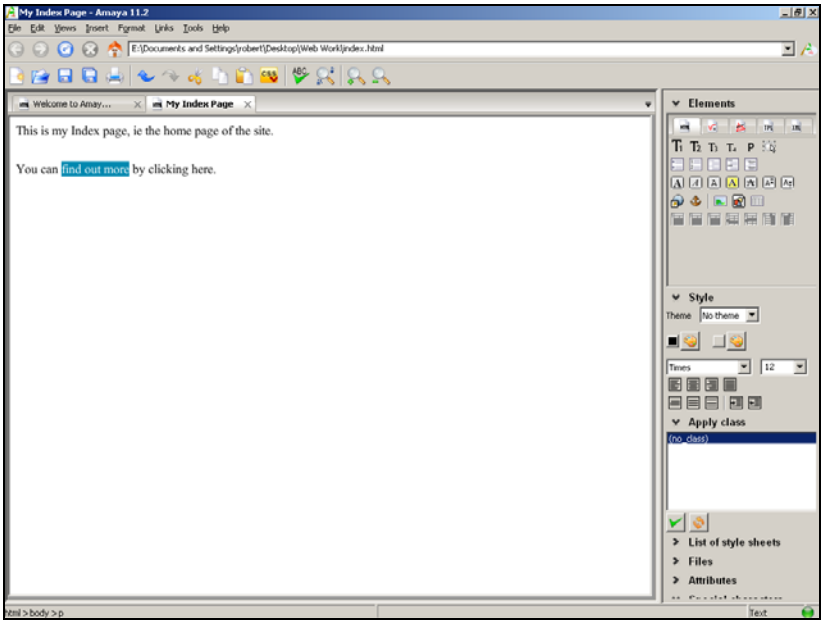
Hyperlinks

Probably the most important concept underlying the entire World Wide Web is the hyperlink. Those 1 or 2 words, normally in blue and underlined, which your visitor can click on to move to another page or another site. Once you know how to create simple web pages, and some hyperlinks to allow your visitors to move between them, you have the basic building blocks for constructing just about any site you could ever need.

Before we can create hyperlinks we'll need to create a second page for our site to complement our index.html home page. So fire up Amaya and, using the instructions on page 39, create another page in your local Web Work folder. This time, name it more.html instead of index.html. With 2 pages created for our site, we can now create some hyperlinks between them.

Using Amaya, open your index.html page. To do this, go to the File menu, choose Open Document, then select File, then browse to the index.html file in your Web Work folder.

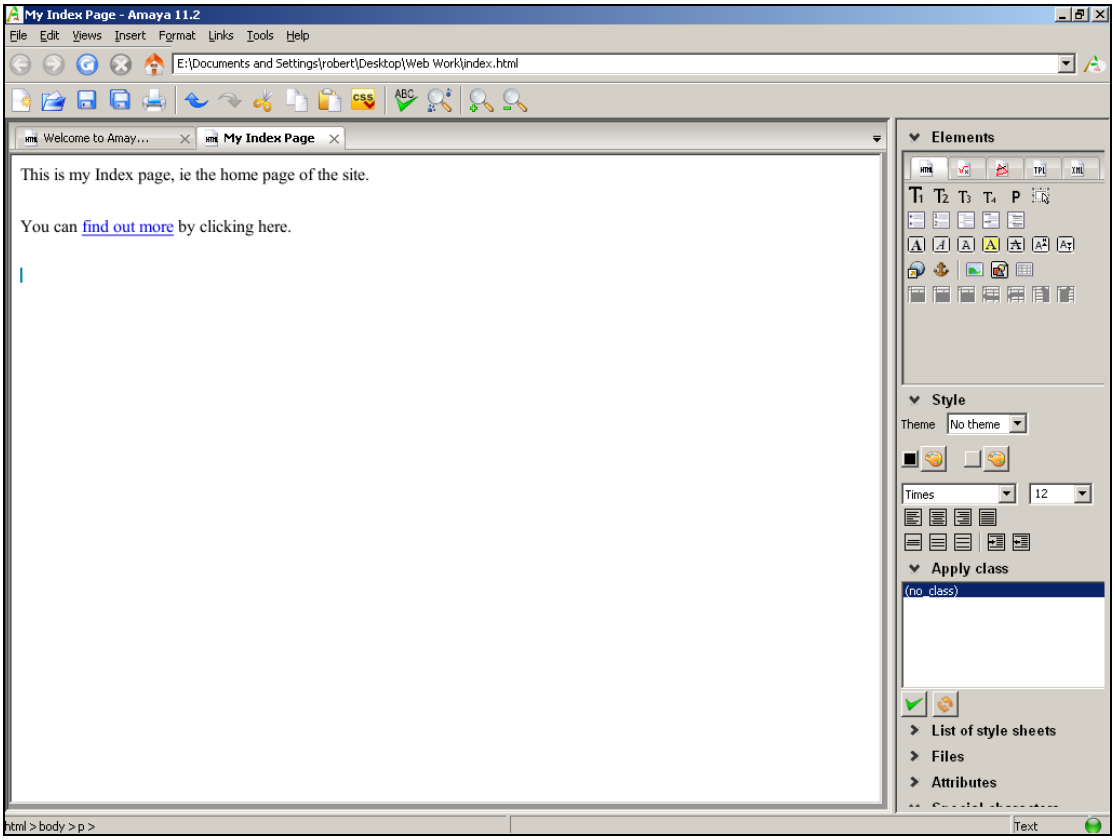
Edit the page so as to include some text which will include the clickable hyperlink. Then select the actual words that you want to turn into a link, by dragging over them with your mouse like this:



By selecting the text, we've told Amaya which words we want to turn into a hyperlink. All that remains is to specify which web page to open when the link is clicked. So, from the Links menu, click on "Create or change link" and, in the box that appears, type **`more.html`**

and click the Confirm button. Then click somewhere on the blank white space within your index page.

Your link is now created, and should appear on your page as underlined text like this:



You've probably noticed my choice of clickable text, namely "find out more". You might have expected me to choose "click here" instead. After all, doesn't it make sense to always turn "click here" or "click this link" into, well, something clickable? Actually, no it doesn't.

One important aspect of web site design is accessibility. This means making sure that your site can be used fully by those with disabilities. For example, that bit of text which pops up when someone hovers over a picture (in case they can't see the pictures), or visitors being able to navigate through your site using keyboard shortcuts rather than a mouse. Making your site accessible is, if nothing else, common sense and a polite thing to do. In many cases, depending on which country you're based in and whether your site is produced by an organisation in the public or private sector, creating an accessible site is actually a legal obligation and you risk prosecution if disabled users can't make full use of it.

People who are blind or partially-sighted often use screen reader software to help them navigate the web. This special software reads out the contents of web pages via a voice synthesizer. Having read out the contents of the page, it will then scan the page for

hyperlinks and read those too, so that the user can press a key as soon as he or she hears the particular link they want to visit. And that's the reason why I chose to highlight the words "find out more" rather than "click here". Because a blind person would much rather listen to a list of choices that consisted of "find out more", "go to main menu", "print this page", "browse our catalogue" and so on, than a list that said "click here", "click here", "click here", "click here" or "click here".

Anyway, back to our page. We've created a link from our home page to our "more info" page. Before our fantastic new two-page site is ready to test, we should also create a link from more.html page to the index page. So close the index page, open the more.html file instead, and, using the same process as above, add a line to more.html which says something like "click here for the home page". Then turn "home page" into a clickable link which goes to index.html and save the file. Then quit Amaya.

Before we can test our links, we'll upload them to the server. So refer to the instructions on page 41 for uploading files, and use FileZilla to upload both index.html and more.html to your server. Finally, close FileZilla and use your Web browser to surf to your web site. Click on the links, and you should be able to move between your two pages. Isn't that neat?

Now that you know everything that's required to create a simple web site with links, why not try creating a third and fourth page for your site, and set up some more hyperlinks between them?

Linking to Other Sites

The destination for your hyperlinks doesn't have to be limited to pages on your own site. You can easily create links to other places. In the box where you type in the destination for the link, just type in the complete address (URL) for the destination site, including the http:// part at the start. Including the http:// is vital, otherwise external links won't work.

If you want to try this, add a line to one of your web pages that says "You can search the web with Google", then highlight the word Google and create a link to **`http://www.google.com`**. Upload the page to your server and check that the link takes you to Google when you click it.

Mailto: Links

Here's a neat idea that allows people to send you email messages via your web site.

Open one of your web pages (either your home page or more.html) and add a line which says "You can send us an enquiry via email if you wish". Now highlight the phrase "send us an

enquiry" and turn it into a link. In the Link box, where you normally type the destination page, type:

```
mailto:yourname@yourmail.com?subject=Enquiry
```

where yourname and yourmail make up your own personal email address.

Now save the page, upload it, surf to it via your Web browser, and see what happens when you click on the link. Your computer's default email program will automatically open, ready to send a message to the address you specified. Plus, the subject line of the message will already be filled in with the word "Enquiry".

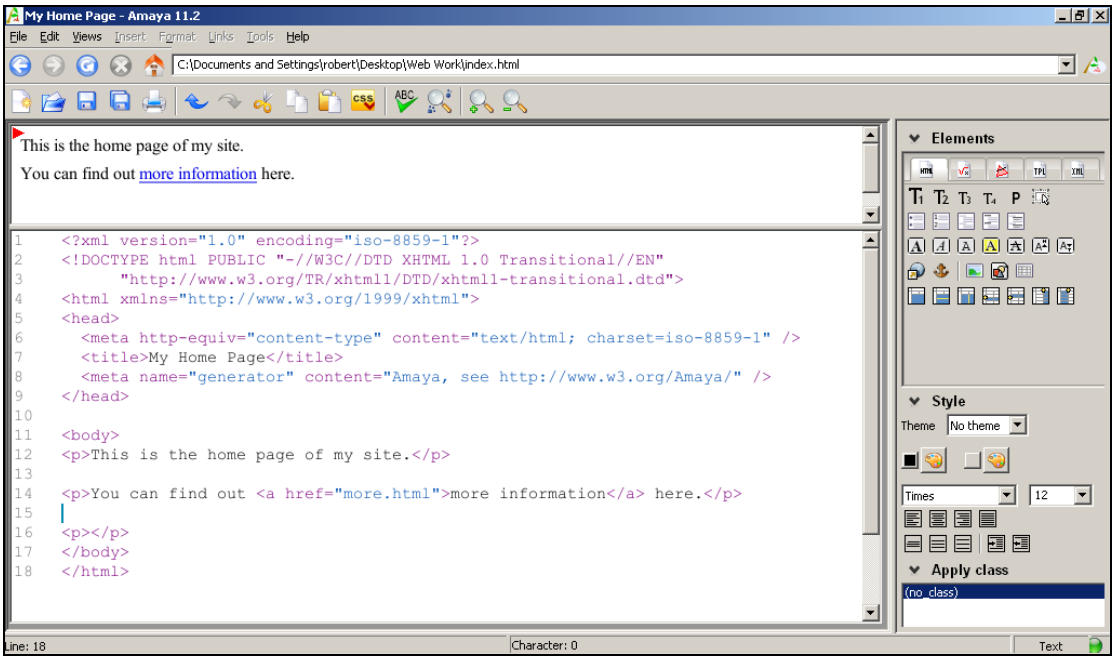
This is a simple way to allow visitors to your site to interact with you. However, there are drawbacks. Primarily, it'll only work if the visitor's computer has been set up to send email, and their computer knows which program to open when it encounters mailto: links in Web pages. Not all computers can handle this, and there's no way you can anticipate the problems. In a subsequent chapter, when we come to discuss HTML forms and PHP, I'll show you a much better way of allowing visitors to send messages via your site. But for now, mailto: links will have to suffice.

Understanding The Basics of HTML

On page 14 I mentioned HTML, the language of the Web. Or at least, the language that all Web pages have to be written in. I also pointed out that you don't have to understand the intricacies of HTML, because programs like Amaya automatically generate this special code for you.

However, for reasons that will become apparent when we start to learn about CSS style sheets, you do need to know some HTML. Thankfully, the amount you need to know is relatively little, and it's not difficult.

Let's take a look at some of the HTML which Amaya has, until now, been hiding from you. Use Amaya to open your index.html page. Then, from the Views menu, click on Show Source. The screen will split to show your page at the top and the HTML code version at the bottom. Use your mouse to drag the dividing line upwards, so you can see more HTML code like this:



Although the code looks daunting at first, two fundamental rules will help you to make sense of it.

1. An HTML file contains a mixture of page content (ie, the words on the page) and formatting commands. Page content appears normally, and Amaya generally displays it in black. HTML formatting commands, called tags, always appear in

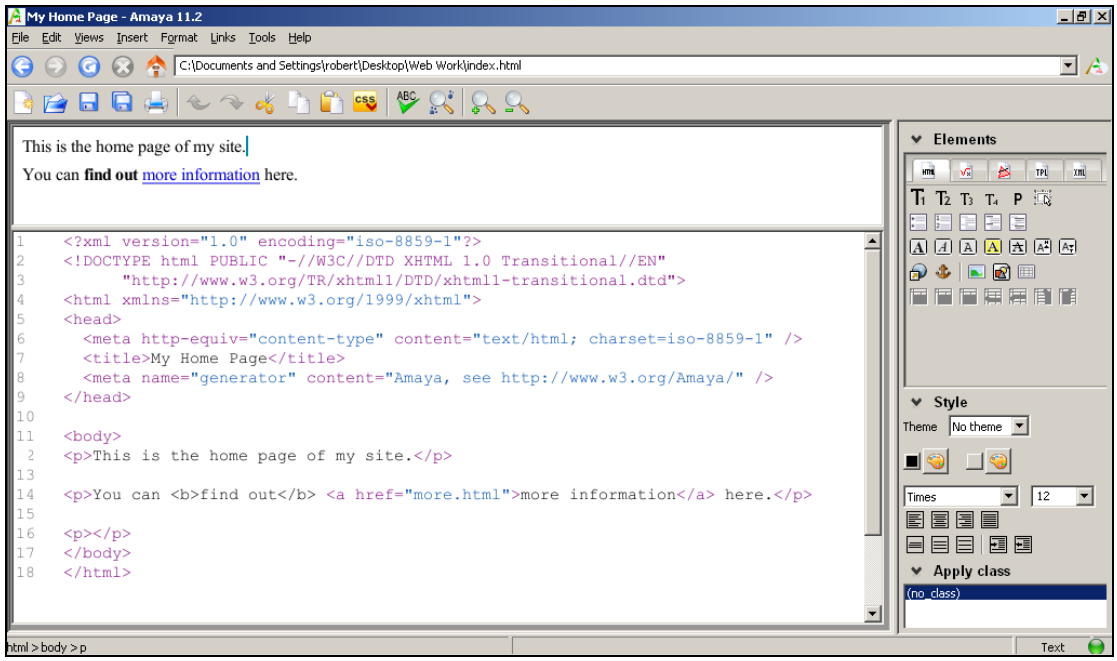
pointy brackets <like this> and Amaya generally displays them in purple, unless you change the program's settings.

2. HTML tags generally occur in pairs. The first occurrence turns on a particular effect, or marks the beginning of a particular section. The second occurrence turns off the effect, or marks the end of the section. The second occurrence uses the same tag name as the first, but with a forward-slash at the start of the tag name.

Armed with these two vital facts, you should now be able to start making sense of the HTML code that comprises our page. There are 2 main sections to the page, namely the head and the body. The head section, which starts with a <head> tag and ends with </head>, contains information about the page as a whole. There's a meta-tag called Generator, the value of which is Amaya. This tells anyone who cares to look at the HTML code of your page that it was created with Amaya. There's also a head tag called <title> which is where your page's title gets put. The page title is what appears in the title bar of the web browser window when visitors (or you) look at your site.

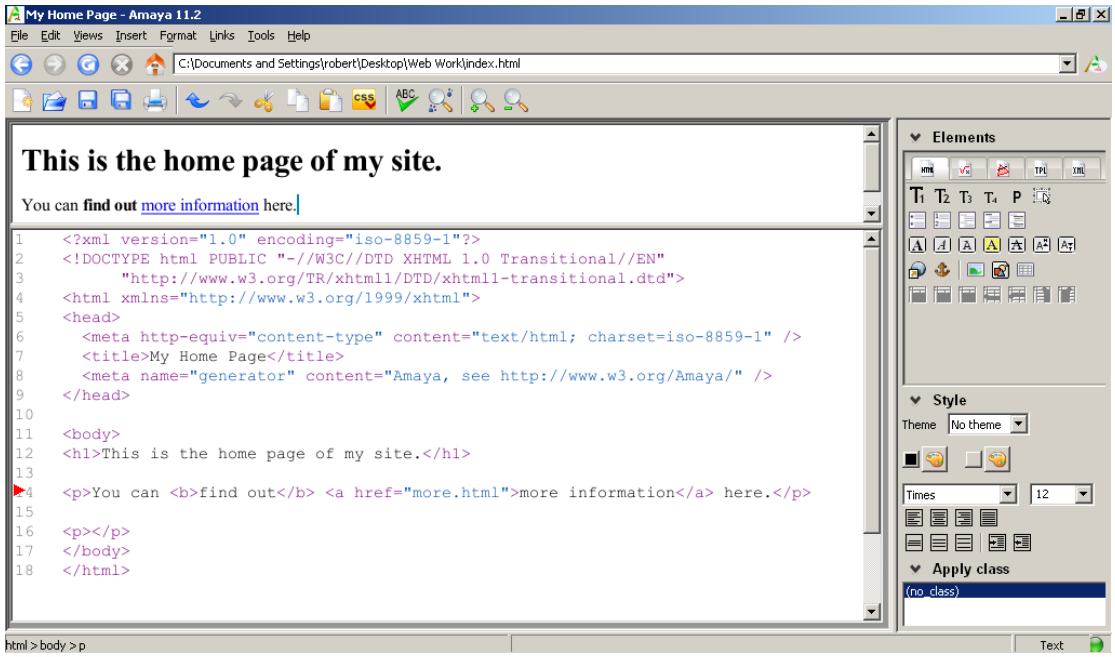
Some tags don't actually need a closing tag, ie the version with the forward-slash at the start. The meta tag is one such example. However, recent changes to the HTML specification say that all tags must be closed, even those that don't officially need to be. This helps to avoid errors on pages, which could confuse some browsers. Hence the </ > tag, which means "close the previous tag".

The most common tag that you'll see on web pages is , which starts each paragraph. The paragraph then ends with , and then a further tag starts a new paragraph below it. There are other tags that affect the way that text looks on the page. For example, the tag makes text bold and <i> makes it italic. To see how these work, click your mouse somewhere in the HTML window and then, somewhere within a paragraph, type . Then, a few words later, type . Now click back into the top half of the screen, and you'll see that your chosen words are now displayed in bold like this:



One set of tags that you'll use often are the heading levels. There are 6 of these tags built into the HTML language, which allow you to create headings of up to 6 levels. Typically, the main heading on your page will be a level one heading, which starts with an <h1> tag and ends with </h1>. You might then want to include a sub-heading later on, for which you'd use a matching pair of h2 tags.

Amaya can generate heading tags for you automatically. In the top half of the screen, highlight the first line and then click the T1 button in the right hand panel. Then click into the lower half of the screen and see how that text is now surrounded by h1 tags like this:



Meta tags

Earlier, I mentioned meta tags. Specifically the meta-tag with a name of "Generator" and a value of "Amaya". Meta-tags are a way for a web page to supply information about itself to the web browser that is displaying the page. One of the most popular uses of meta-tags is to include a brief description of the page, and a few keywords about it too. For example, the <head> section of an HTML file might contain the following code:

```
<meta name="description" content="Latest availability for the Grand Hotel">
<meta name="keywords" content="availability, late bookings, vacant rooms">
```

It's always a good idea to include these two tags (a description and some keywords) on every page you create. It will help your site to be found and indexed by search engines such as Google.

In the past, search engines based the contents of their indexes on little more than this information. However, that's no longer the case, as many web site operators tried to cheat the system by, for example, including explicit sexual keywords to describe fairly dull sites. So although meta-tag information is no longer as important as it was, it is still taken into account by search engines as part of a much more complex process.

There's more about this in the chapter on Search Engine Optimization, on page 313.

HTML Accessibility, Accuracy and Privacy

You maybe surprised to know that anyone, and not just you, can view the HTML code for the web pages you create. Equally, you can view the HTML code of any page on the Web. Just about every Web browser, including all the leading ones such as Internet Explorer, Firefox and Safari, have an option that will show you the source code, as it's called, of the page you're looking at.

For example, open your web browser and then surf to one of your favourite pages, such as Google, Wikipedia, Facebook, or any other page that you prefer. If you're using Internet Explorer, bring up the View menu (press the alt key to show the menu bar if it's not displayed) and then choose View Source. All other browsers have a similar option, either from the menu line or via a right-click. You'll then see the HTML source code of the page.

This total lack of privacy has two implications. First, it means that you should never hide things in your HTML code that you wouldn't want others to see. For example, there's a method for putting comments into HTML code which don't show up on the finished web page but which **will** show up if someone views your source code.

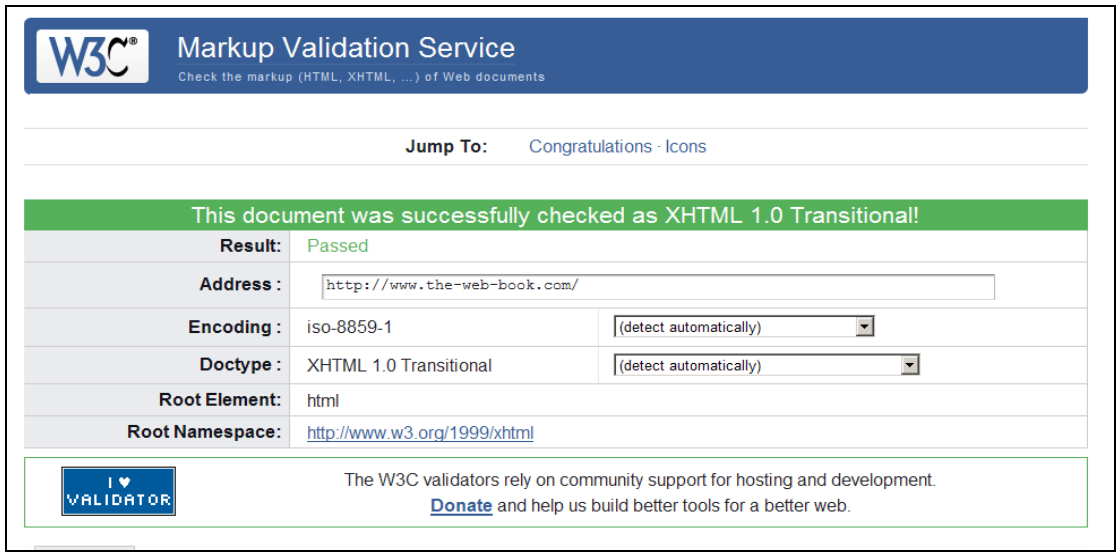
Second, the public nature of HTML means that, if you want to know how someone has achieved a particular effect on their web page, you can just view their code and find out. There's nothing wrong with doing that. Professional web people do it all the time. However, if you're tempted to copy and paste entire chunks of someone else's hard work into the code view screen on Amaya, don't be. It's technically possible, of course, but morally and legally wrong.

Validating your HTML

Designing a web page isn't particularly easy, especially if you want to ensure that your HTML code is correct. Thankfully, there are various web sites available on the internet which will automatically check one or more pages on your site and report any errors. These sites exist because of the above-mentioned lack of privacy when it comes to HTML. If your page is on the internet, anyone, or any automated system, can look at its HTML code.

The best-known HTML validator, as such services are called, is run by the World Wide Web Consortium (WWWC, or W3C for short). To use it, just go to **`http://validator.w3.org`** and enter the address of your page. If you haven't yet uploaded the page to a server, there's an option that lets you upload the HTML file to the validator, or you can even copy and paste a block of HTML code straight from Amaya into the validator.

Either way, the end result of the validation is (hopefully) a congratulatory screen such as this:



If your code fails validation, the validator will normally tell you why, so you can go back and fix any problems.

It's always a good idea to ensure that all your pages validate correctly. If they don't, there's no guarantee that every visitor's web browser will be able to display your pages properly. In some cases the layout might look slightly wrong, while in others the page might not display at all. Some browsers are much more forgiving of minor errors such as unclosed tags, or tables where a new row is started before the previous one ends. Some, though, will complain bitterly.

A Bit More about Accessibility

Like many aspects of web site design, accessibility (making it possible for people with sight and mobility problems to use your site) is one of those things that's remarkably easy to build in from the start but very time-consuming to retrofit as an afterthought. Therefore, it makes sense to consider it as early as possible in the overall design process.

Some of the things that you should do to ensure that your site is accessible include:

- Provide alternative ALT text for every picture or image that you use.
- Don't create information which can't be understood by those who can't see colours properly. Eg, "Click the red box rather than the green one in order to browse our brochure" will be of little use to someone who's colour-blind.

- Ensure that there is sufficient contrast between text and background colours.
- Avoid scrolling text and other dynamic effects. They are hard to read.
- Avoid any technique which may produce flickering effects on-screen.
- Use the clearest and simplest language for site content.
- Keep paragraphs short, and include some white space between them.
- Do not use frames. Screen readers don't like them.
- Dark text on a light background works better that light text on a dark background.
- Use a line spacing of around 1.5 to ensure that the text on a page doesn't look too "heavy".

Now that you know the basics of HTML, and specifically the way that formatting tags work for things such as paragraphs, headings, bold text and so on, we can move on to learn about CSS.

Cascading Style Sheets

One of the key techniques used in modern web page design techniques is something called the separation of presentation and content. In the early days of the web, there was no way to do this, which meant that HTML pages were bloated, repetitive and difficult to understand. For example, if you wanted the main body text of the site to use the Verdana typeface, in a size of 12 pt, every single paragraph of text would need to start with a **<font face=verdana size=12pt>** tag. That could amount to hundreds or thousands of such tags across the entire site. If you subsequently wanted to redesign the site, changing all those tags was a lot of work.

A few years ago, along came a new revision of the HTML language (and a new generation of web browsers that were capable of displaying pages created with it). Top of the list of new features was something called CSS, or Cascading Style Sheets, or HTML Style Sheets.

A style sheet is a separate file that complements one, some, or all of the HTML pages on your site. It contains all the information about how the text, pictures and layout of your pages actually looks. So, to update the above example, there might be a section in the CSS file which states that all instances of the **<p>** tag should use the Verdana typeface in a 12pt size. Now, all the pages of your site merely need to surround the paragraphs with nothing more than a <p> tag and they'll all magically appear in the correct typeface and size. The best bit, is that you can tweak the appearance of every paragraph in your entire site just by changing one entry in a single CSS file, rather than having to change any <font> tags.

CSS has other benefits too. When you specify which CSS file is to be used to format your page, you can actually specify different CSS files for different situations. For example, one set of styles for when the page is viewed on screen and a different set for when the user prints the page on their printer. And yet another set for when someone views the page on a smartphone or a PDA or a projector.

I mentioned accessibility, and making your site disability-friendly, on page 49. CSS can help here too. You can set up a separate style sheet for people who have basic sight problems or dyslexia, ie large text, no graphics, and a more readable background colour. You can then easily add an option to your site that uses this style sheet rather than the normal one, for those visitors who prefer it. And all, of course, without having to maintain separate copies of the page content, because this is now separate from the presentation side of things.

The ability to separate presentation and content has other benefits too. Creating good web sites requires skills in both writing and design. CSS means that one person can concentrate on writing the content for the site, while someone else can work their magic on the style sheets.

CSS is so much part of web design nowadays, that it's rare for anyone to create a new site from scratch which uses the old <font> tags. That's not to say that there aren't millions of old-fashioned pages still around, but CSS is definitely a better way of doing things, so that's the method that we'll cover in this book.

About DOCTYPEs

Before we start with CSS, a word about document types. You'll have noticed, above the **<head>** section in the HTML code shown on page 53, , a line that starts with the word DOCTYPE. In full, the line is:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xHTML1/DTD/xHTML1-transitional.dtd">
```

This is known as DTD, or Document Type Definition.

HTML is continually evolving. As mentioned above, for example, the advent of CSS now means that you are advised not to use <font> tags. The DOCTYPE is a way for you to tell the visitor's web browser which version of HTML you'll be using. With this information, the browser can ensure that your page displays correctly.

The two main document types that you'll see used most are XHTML 1.0 Strict and XHTML 1.0 Transitional. Transitional is the version that allows both new-style CSS as well as old-fashioned <font> tags. "Strict" mode doesn't allow <font> tags. If you specify a Strict document type, and then proceed to use <font> tags, you are technically producing invalid HTML which the browser might well reject.

The simplest option is therefore always to use a Transitional doctype, but to adhere to the latest standards and techniques.

Getting Started with CSS

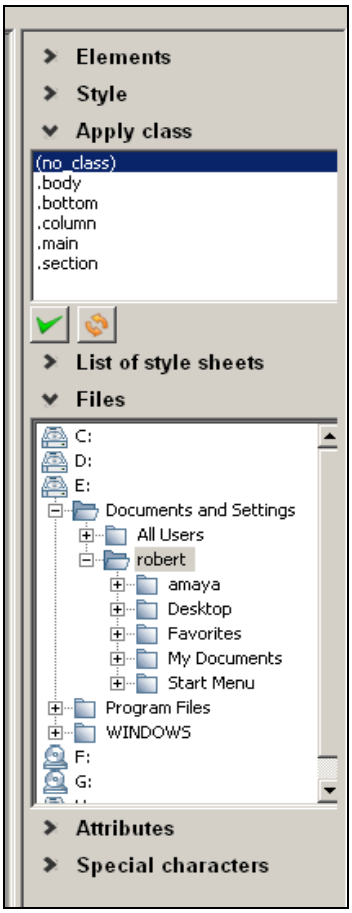
If Amaya isn't already open on your computer, open it now. Then load your index.html home page. The quickest way to do this is to use the right-hand menu pane, which is not something we've covered before.

The right-hand side of your Amaya screen probably looks, right now, something like this:

There are various menu options that can be opened and closed on this panel, each of which is preceded by a ➤ character, such as Elements, Style, List of style sheets, Files, Attributes, and so on. When a menu set is visible, the prefix shows as ➧. When it's hidden, the character shows as ➤ instead. You can show or hide these sections by clicking on the arrow characters.

Perhaps the most useful of these sections is Files, which brings up a simple file manager that makes it quick and easy to open any of your existing web pages. The Apply Class box is also useful, for reasons that we'll come to shortly.

So, click on the arrow characters to close all of the sections, then open the Files section and the Apply Class section. Your screen should now look thus:



Note, by the way, that the Style section is nothing to do with CSS. It's a built-in feature of Amaya that uses a non-standard way of changing the look of web pages, so we won't be using it.

Now it's time to open your index.html page. Use your newly-found file manager to browse to your Web Work folder, and double-click on index.html. Or, if you prefer, go to the File menu on the top line and then use the Open Document option. Either way, you'll end up with your index.html page open and ready for editing.

We now need to create the style sheet file for our site. From the File menu on the top of the screen, choose New, then New Style Sheet. Use the folder button to browse to your Web Work folder if it's not already shown. Amaya wants to name the file New.css but this is unwise. It is conventional to name the main style sheet file style.css and not to include any capital letters in the name. So change New.css to style.css in the New CSS Style Sheet box.

In the same box, select the **`In New Tab`** radio button for where to open the file, then click Create.

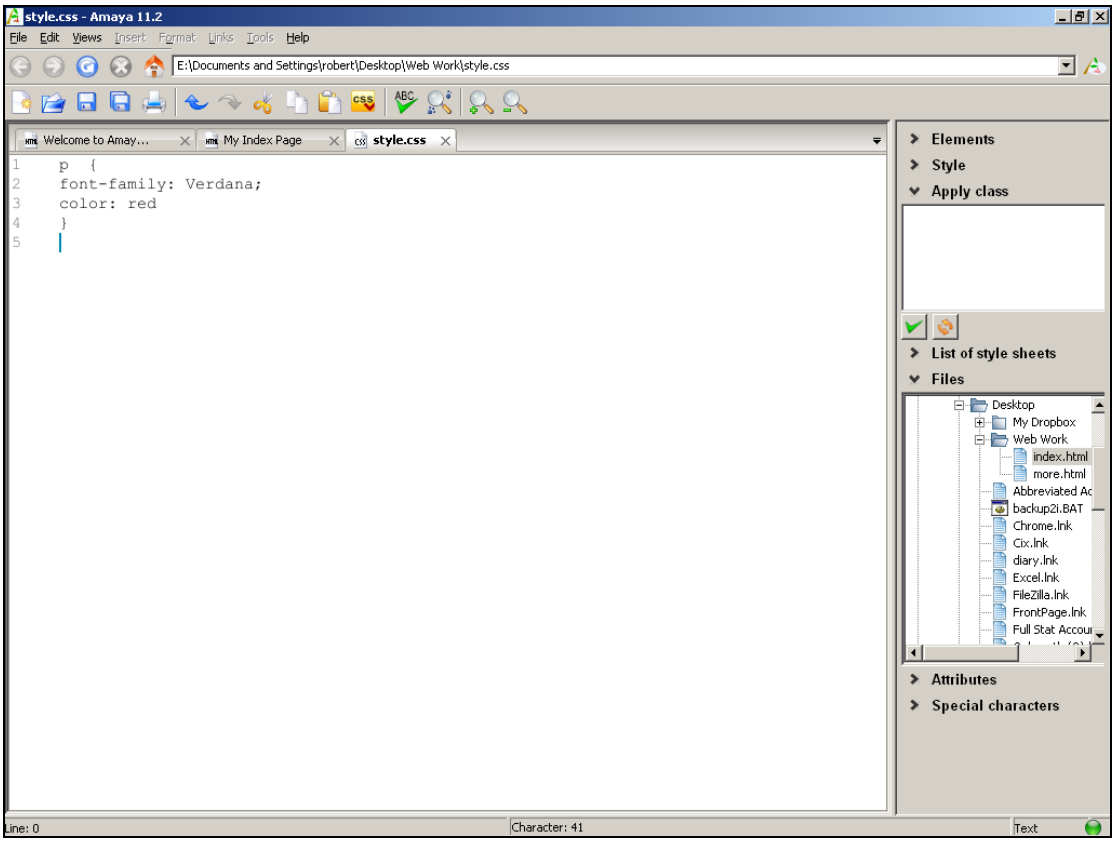
You will see that there are now 2 (or maybe 3) tabs open in Amaya. One is your index page, and the other is your css file. You can click on the tabs to switch between files.

In the previous chapter, we learned about HTML tags. Now, we can start to create a style sheet that applies particular formatting to particular tags.

Into your blank css file, type the following:

```
p  {
font-family: Verdana;
color: red
}
```

Your screen will look like this:



We've now created the first entry for our site's style sheet. This specifies that all <p> tags should use the Verdana font and should be coloured red.

The format of a CSS file is fairly easy to understand, once you know the basics. On the first line of each entry goes the name of the HTML tag that you want to specify a style for, followed by a curly bracket. On subsequent lines, specify the various aspects of that style,

such as font-family and color. I'll explain later how to find out what other style aspects are available.

Each style aspect ends with a semi-colon. Finally comes a closing curly bracket.

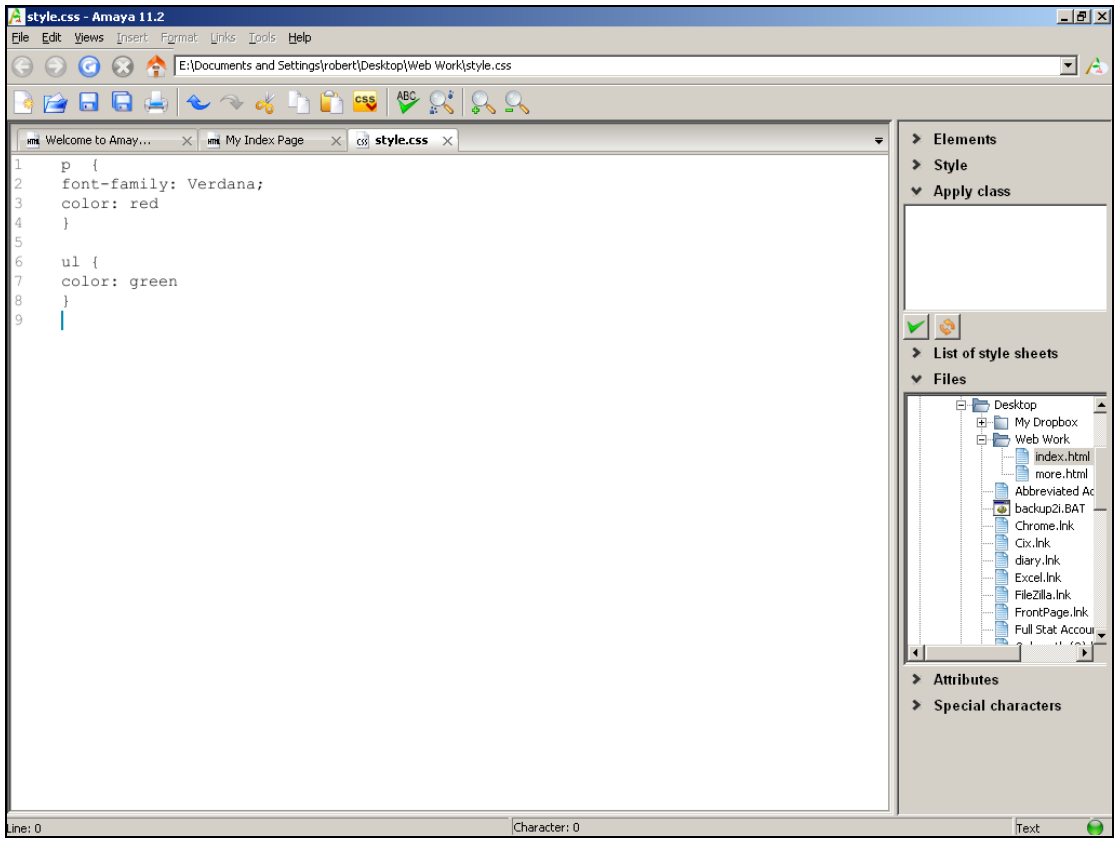
Note that the semicolon on the final style characteristic is optional, as you can see from the way that it's not been included in the examples on the illustration below. However, it's generally good practice to get into the habit of always including the semicolon at the end of every line. Otherwise, if you add new style options below what is currently the last line in a style definition, and you forget to add the semicolon to what is no longer the last line, your style sheet file won't work properly because the web browser won't always know where one line ends and the next one begins.

While we're here, let's add another style definition to our CSS file, this time for the <ul> tag which defines a bulleted list (ul stands for Unordered List, in contrast to a numbered list which uses the <ol> tag for an ordered list).

So, add a new section to the CSS screen as follows:

```
ul {
color: green;
}
```

Your CSS file should now look like this:



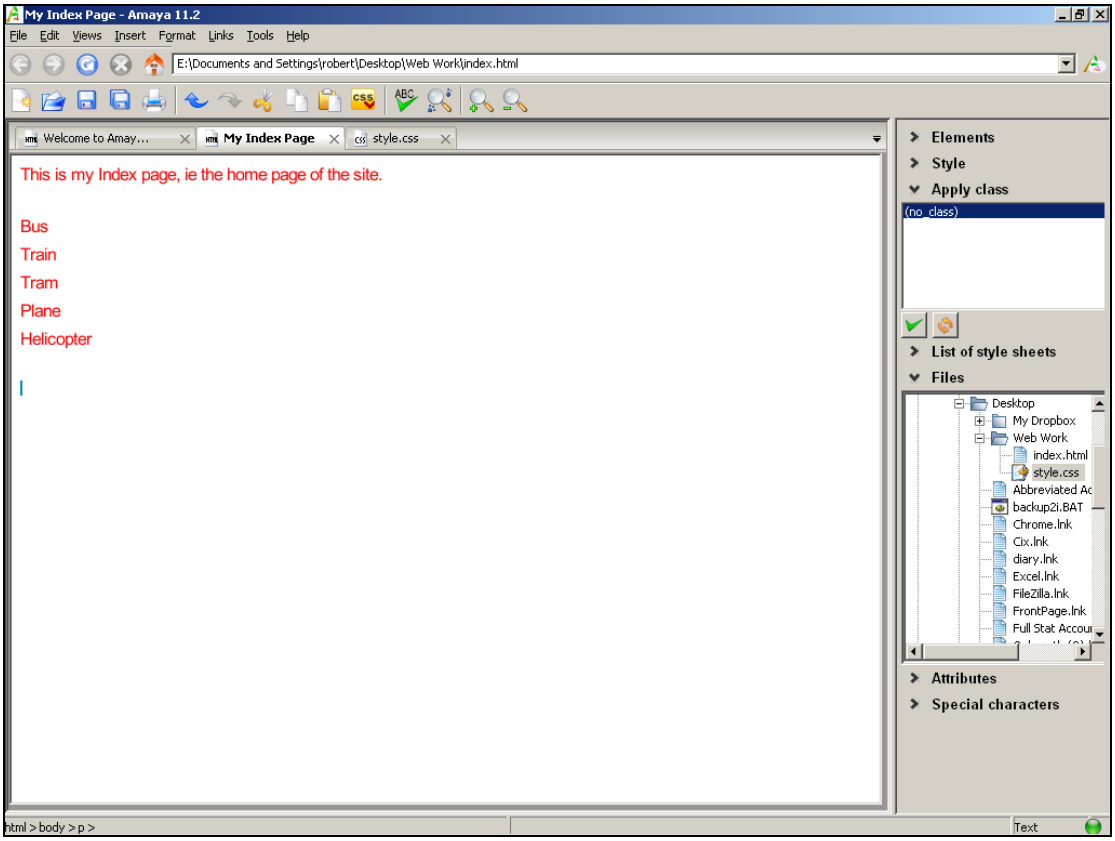
Flip back to your index.html page by clicking on its tab. Has anything changed? Are your paragraphs now in Verdana and coloured red? Sadly not, for one very good reason. Although you've created a style sheet, we haven't yet added the necessary command to our index page to specify where it should get its style information from. So let's change that.

First we need to save the style sheet that we were working on. So flip to the style sheet tab and, from the File menu, choose Save. Now flip back to the index.html page tab.

From the Format menu, choose Style Sheets and then the Link option. In the box, just type style.css and press the Confirm button. There's no need to specify the full path, because both your index.html page and your style.css file are in the same folder.

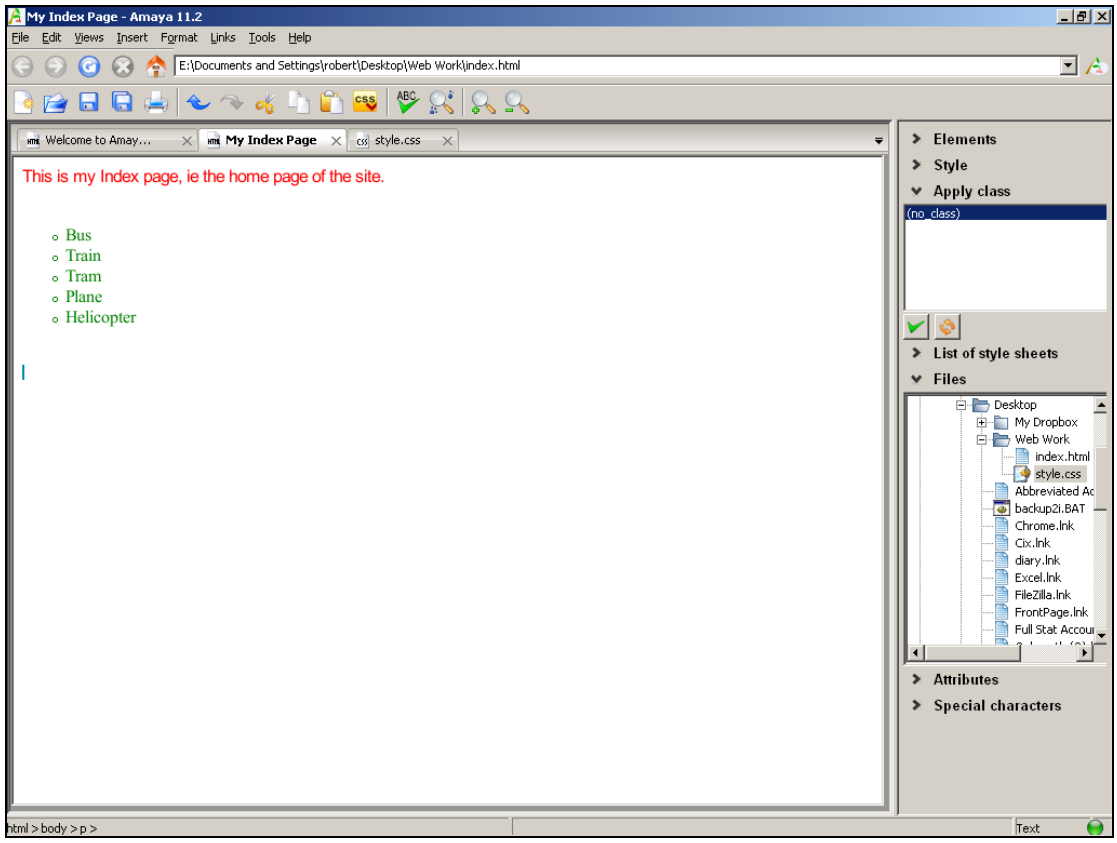
And now, as if by magic, your index page has changed. Your paragraphs of text are now, as requested, using the Verdana font and are red.

We also specified a style for our unordered lists, so let's see if that works too. First, type a few short paragraphs into your index page like this:



As expected, they will be red and Verdana because that's what the style sheet says that paragraphs should be. But that's about to change.

Select your entries, using the keyboard or mouse. Then, from the Insert menu, choose List, then Unordered List. Here's what the screen should look like now:



Your list uses <ul> tags, and your style sheet says that <ul> tags should be green. So they are.

The list items aren't using the Verdana font, though, because your ul tag style didn't say that they should. But we can fix that quickly and easily. Flip back to your css file by clicking on its tab. Add a new line to the ul tag style to specify font-family: Verdana. Save the css file and then flip back to the index page, and you should find that the font used by your list has now changed.

The ability to change one, or indeed hundreds, of pages by making a single change to a single style sheet file is what makes CSS so useful. If all of the pages in your site are linked to the same style sheet, changing the CSS file changes all of the pages instantly.

A Word About Fonts

You probably have one or two hundred fonts installed on your computer. Most people do. Every time you install a new program, it frequently comes with a selection of new fonts that are installed. If you look at the list of available fonts in Microsoft Word, or whatever word processor you happen to use, the list will be enormous. However, that doesn't mean that you can use all of them on your Web pages.

When you specify a font in your CSS file, the actual data (all those lines and curves) that make up the font don't get included in the page. All that makes it into the HTML is an instruction to the visitor's web browser to display your chosen text in your chosen font. Which only works if that font happens to be installed on the visitor's computer. If it's not, the browser will make a stab at choosing a font that looks vaguely similar, but the end results are rarely identical.

Because of this, it's always safest to ensure that you only specify fonts in your CSS files which are pretty much guaranteed to be installed as standard on most visitors' computers. Although there aren't many such fonts, they do provide a fairly wide cross-section of styles. The fonts in question are as follows, in their actual style so you can see what they look like:

Arial
Verdana
Trebuchet MS
Courier New
Georgia
Times New Roman

Note that these names are case-sensitive, so specifying verdana in a style definition isn't the same as Verdana, and probably won't work unless the visitor's web browser is feeling particularly forgiving at the time. Which most modern browsers never do.

These 6 fonts can be classified loosely into three categories. Arial, Verdana and Trebuchet MS are what's known as sans-serif fonts. Serifs are the embellishments on the ends of characters, and sans-serif fonts don't have them. Compare this with serif fonts such as Georgia (which is used throughout this book) and Times New Roman.

For body text, ie the main bulk of the text on your pages, sans-serif fonts tend to work best. For headlines and picture captions, serif fonts look good.

Courier New is an example of a fixed-width font. In this font, each character is the same width. So the word RAILWAY is the same width as the word ILLICIT, as they both contain 7 characters. Like this:

RAILWAY
ILLICIT

In fonts that aren't fixed-width, each letter is a different width. Like this:

RAILWAY
ILLICIT

Such fonts are often known as proportionally-spaced fonts, as each character's width on paper is proportional to its actual width.

Using a fixed-width, non-proportional font is useful in lists of numbers if you want all the columns to line up correctly. For just about all other purposes, proportional fonts tend to be more readable and look more professional. Unless you specifically want to create a web page that looks like it was produced on a typewriter.

Classes

So far, in our discussion of CSS, we've talked about defining styles based on HTML tags. By creating a style definition for the <p> tag we can specify how all of the body paragraphs on our pages look. Equally, by creating a style called ul, we can style all of our unordered lists. If you were to add a style for the h1 tag, you could change the look of all the level-1 headings on your pages.

As you might expect, there's a lot more to CSS than letting you choose a font and a colour to be associated with HTML tags. Actually, CSS is hugely complicated, and is the bane of most professional web developers' lives. A typical reference book on CSS runs to over 600 pages, but thankfully we're not going to go into that much detail in this book. There is, though, one more highly useful feature of CSS that you'll want to know about, and that's classes.

A class is a style that you can use anywhere on a web page, rather than for a particular HTML tag. Rather than being named after a tag, you can name your classes as you see fit. For example, if you often find yourself mentioning the name of your favourite cheese on your web pages, you could create a class called fave-cheese. Then, wherever the name of that cheese appears, you assign it to that class. The text can be an entire paragraph, a few lines, or even just one word. Here's how to create a CSS class and refer to it within a web page.

Make sure that Amaya is open, and that you have both your index.html and style.css files open in separate tabs. Now, in your style sheet file, add a new style as follows:

```
.fave-cheese {
color: yellow;
font-family: Arial;
font-size: 18px;
}
```

There are 2 differences between this style and the others that we have created. First, we chose our own name for the style, rather than using the name of an HTML tag. Second, the style name starts with a dot. The dot is what tells the browser that this is a class rather than an HTML tag style.

Save your css file (File, Save), then flip back to the index page. Notice the Apply Class area on the right hand side of the screen, which now lists your class.

Type some text onto the page, highlight it, then double-click your class name in the list., Your chosen text is now formatted according to your class. Highlight some more text and apply the class to it too.

Between them, tag styles and classes allow you to format your web pages just about any way you wish. If you want to style all instances of a particular tag, ie all of your h1 headings or all of your paragraphs, use tag styles. Where you want to just style a few words or sentences here and there, use a class.

Making Styles Work For You

The main benefit of using styles in HTML pages is that they make it easy to make wide-reaching changes to a web site with a single click. Just change a single entry in your css file and the entire site changes. However, if you are to gain the maximum benefit from using css there are two things you need to know about.

First, keep the number of styles (tag and class) to a minimum. If you find yourself creating 6 different versions of the fave-cheese class in order to fit the particular idiosyncrasies of your site's layout (eg one version for headings, one for picture captions etc), you lose much of the benefit of CSS because there are now 6 different things that you need to change if you want to alter the way that the names of your cheeses are displayed.

Second, choose your class names with care. Pick names based on what the class is used for, not what it looks like. Consider what would happen if I'd named my cheese class "arial-yellow" rather than "fave-cheese". If I ever redesign my site and decide that I want the cheese names in purple rather than yellow, I'll end up with a class called arial-yellow which actually displays in purple. This will make it much less obvious to me, or anyone else in the future, to remember what each class is used for.

HTML Tags Names

To make the most of HTML tag styles, you obviously need to know what tag names are available to you. You can find a complete list on the internet (just type HTML Tags into Google), but here are a few to get you started:

<h1> to <h6>	Heading level 1 to 6
<table>	Used at the start of a table.
<tr>	Starts each row of a table.
<td>	Stands for Table Data. Used for each cell of the table.
<b>	Bold text.
<i>	Italic text.
<body>	The entire body of the web page.
	Unordered list, using bullets at the start of each item.
	Ordered (ie, numbered) list
	A list item (used within ordered and unordered lists).
<a>	Anchor. Used to create hyperlinks.

A Better CSS Editor

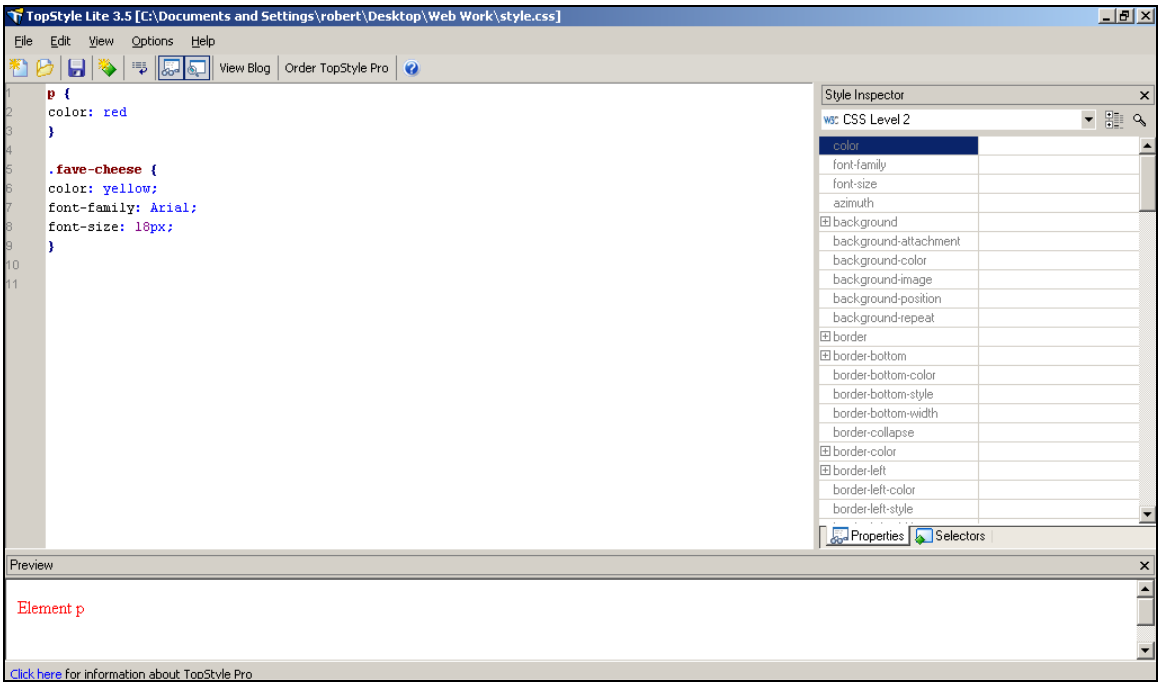
Style sheets are an incredibly powerful feature of modern web design, and there's absolutely no reason why you should use any other method to format your pages. Unfortunately, Amaya doesn't make it particularly easy to create and maintain CSS files. So far, you've created styles manually, line by line. This is slow, prone to errors, and only works if you happen to know the language of css (font-family, color, and so on).

Now that you understand the basics of what a css file is, and the sort of commands it contains, we need to find a better program for editing css files. Preferably one that's free. There are quite a few such programs available, downloadable from the internet. The best one, in my opinion, is Topstyle Lite. It's a cut-down version of a commercial product called Topstyle Pro, which offers even more features. But for anyone who needs a basic program to easily create and edit css files, Topstyle Lite is perfect.

Your next task, therefore, is to download and install Topstyle Lite. Close Amaya for the moment, but don't uninstall it. We'll continue to use Amaya for editing HTML pages.

To download Topstyle Lite, go to **`www.newsgator.com`** and follow the links to Topstyle and then to Topstyle Lite. Or type "topstyle lite" into Google. Once you have downloaded the installer file, double-click it to start the installation. The version of Topstyle Lite that we're using in this book is 3.5.

With TopStyle Lite installed, launch the program. Then go to the File menu and choose Open. Browse to your WebWork folder and locate the style file that we have already created, and double-click it to open it with Topstyle Lite. The screen will look something like this:



One of the great things about the web is that HTML and css files are just standard text files, rather than using proprietary or complex formats such as Word or Excel. This means that you can switch editors easily. If you don't like Amaya, you can load your HTML file into another application. Equally, the css file that we created earlier with Amaya can now be loaded into Topstyle Lite.

Let's start by changing the colour of our favourite cheese text, from yellow to blue. Not just any blue, but a nice bright pale blue. In Amaya, you'd have to know the name (or code number) for the particular colour, but Topstyle Lite makes things much easier.

Click somewhere within any of the 3 lines that make up the definition of the .fave-cheese class, and you'll see the full details of the style shown on the right hand side of the screen. Click the yellow block and an arrow linking to a drop-down list of colours will appear. Click that arrow. Choose a colour from the list of options, or select the top option from the list which says Choose Colour. This then brings up a colour picker from which you can select the particular shade of blue that you want. When you're finished, click OK. See how the definition of the class has been edited.

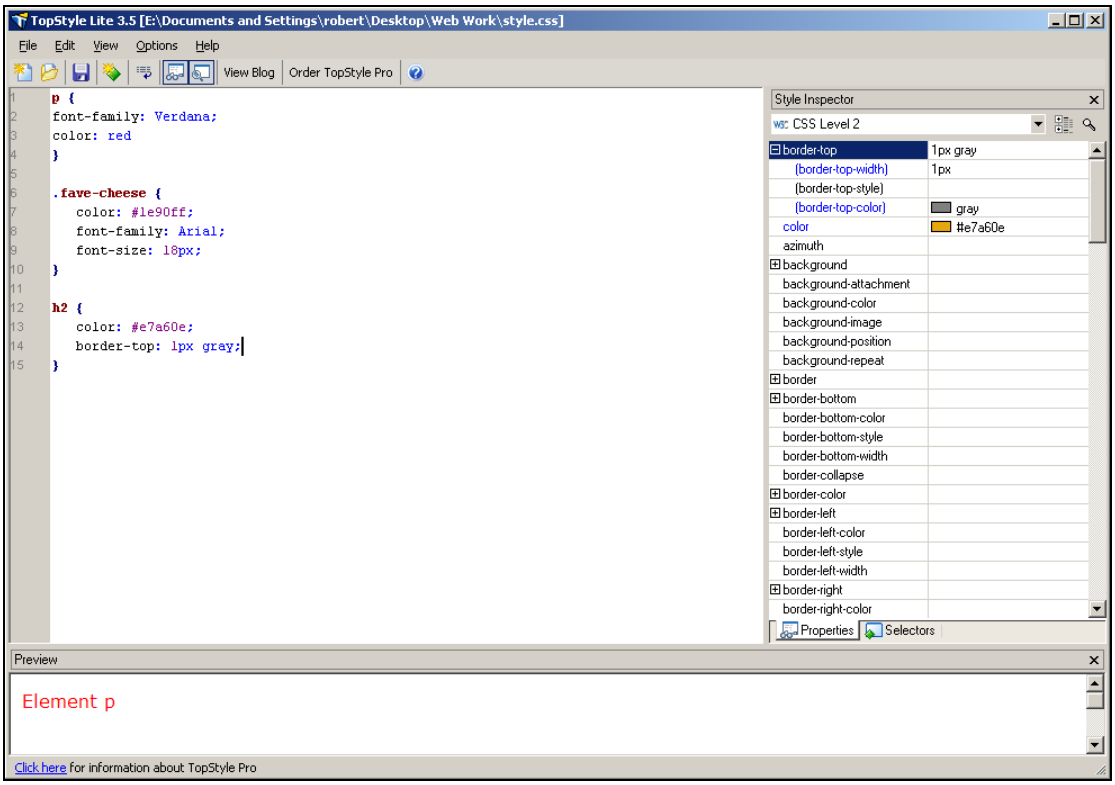
Let's use TopStyle Lite to create a completely new style. We'll create a tag style for a second-level heading, ie the <h2> tag.

Move your cursor down to the bottom of the screen, below the existing pair of style definitions. Then, from the Edit menu, click on New Selector. A selector is the official name for a style's title (whether it's a tag or a class). In this case it's a tag so, in the HTML Element box, scroll down to h2, select it, and click OK.

TopStyle Lite will create the first and last lines of the style definition, like this:

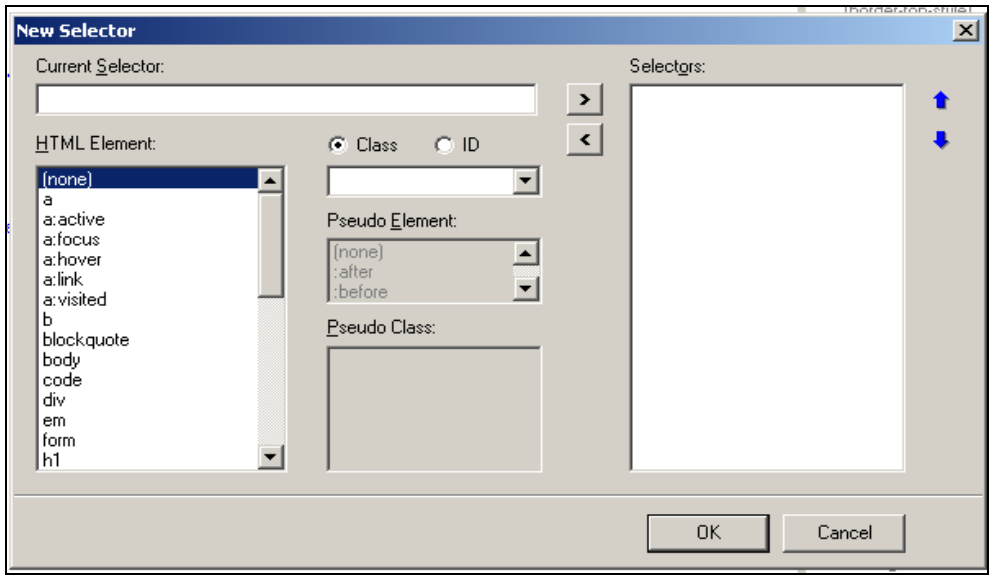
```
h2 {

}
```

Now you can use the panel on the right hand side to select the style and design of your h2 tags. Here's an example:



As you can see, I've chosen a text colour of dark orange, and a top border of a grey line that's 1 pixel thick.

Using TopStyle Lite to create a new class, rather than a tag style, is just as easy. Again, choose New Selector from the Edit menu to bring up the New Selector box that looks like this:



From the set of radio buttons that offer a choice of Class or ID, choose Class. Then, in the box below, type the name of your class, and then choose the options as you wish. TopStyle Lite will automatically add the dot at the front of the class name if you forget.

When you've finished experimenting with TopStyle Lite, save your css file by choosing Save from the File menu, then quit the program.

Before we leave the subject of CSS and TopStyle Lite for the time being, let's just check that our new <h2> tag style works correctly.

Open Amaya, then open your index.html page. Type a few words, and highlight them. Then, from the Insert menu, choose Heading and then H2. This will mark those words as a level-2 heading by surrounding them with an <h2> tag at the start and a </h2> tag at the end. You should find that your words now display correctly, using the h2 style that you created with TopStyle Lite.

ID-based Styles

Before we leave this brief introduction to CSS, there's one more thing you need to know about, before you can move on to the next chapter. So far we've covered two different selector types, namely tags and classes. There's actually a third type, called an ID. Here's how it works.

HTML allows you to assign a unique name, or id, to any tag on a page. The only stipulation is that a name must be unique on that page. For example, consider this level-2 heading:

```
<h2>Further Information About Cheddar Cheese</h2>
```

We can give this heading a unique id by changing the opening h2 tag thus:

<h2 id="further-cheese">Further Information About Cheddar Cheese</h2>

Unless you say otherwise, this heading will appear as any h2 heading would, according to the h2 entry in your style sheet. But because this particular heading has its own id, we can do something rather clever in the css file, like this:

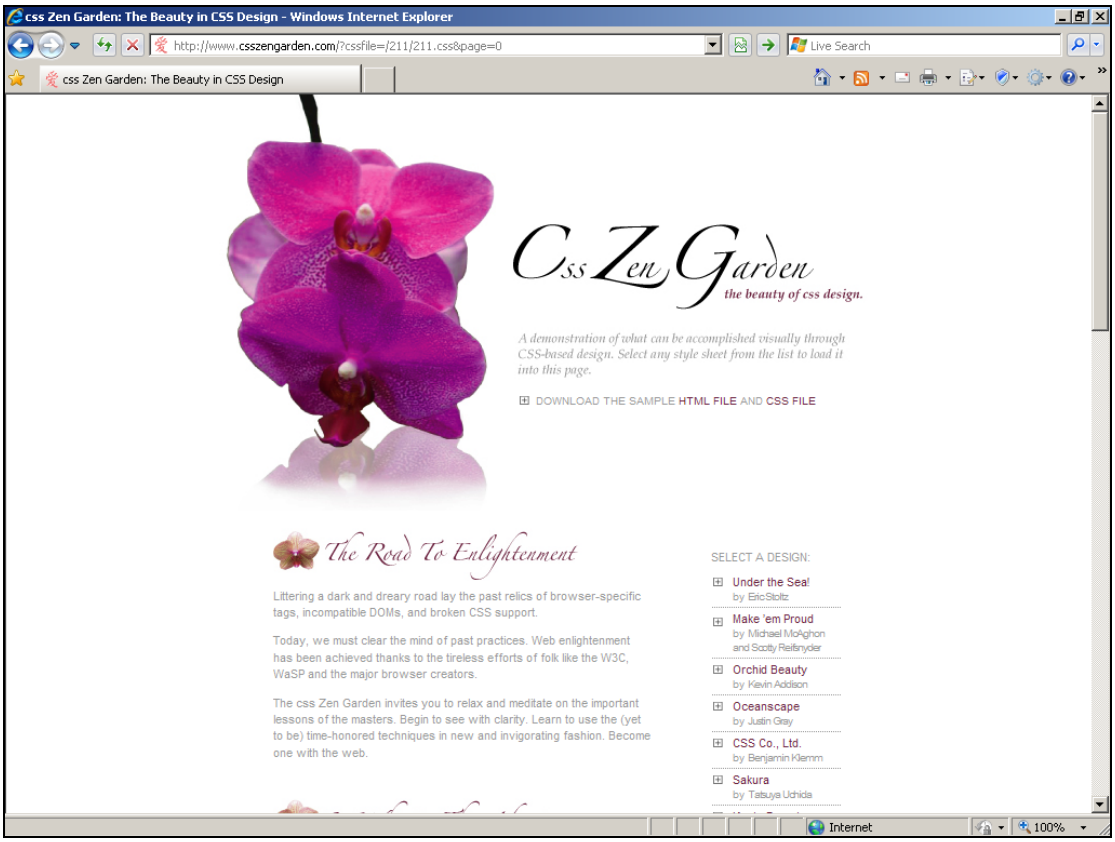
```
#further-cheese {
      color: navy;
}
```

Notice the # symbol at the start of the style name. This signifies an ID-based style. It simply means that the style is only applied to the HTML tag which has the id of further-cheese. It doesn't matter whether that tag is an h2 heading, a paragraph, a list item, or indeed anything else. The only rule is that you mustn't assign the same id to more than one HTML tag on the same page. If you need to apply the style in more than one place, use a class instead.

Extreme CSS

When used relatively simply, CSS is a powerful tool that will help you create web pages that are easier to understand and easier to maintain, when compared to the old way of doing things with <font> tags. But if you thought that this chapter has taught you everything there is to know about CSS, you're sadly mistaken. By using the advanced features of CSS you can gain even greater control over the look of your pages. For example, here's the home page of one of the best exponents of CSS-based design, namely the folks at **www.csszengarden.com**:

By clicking on the menu entries on the right hand side of the page, you can choose other CSS style sheets to apply to the page. For example, here's the Orchid Beauty style:

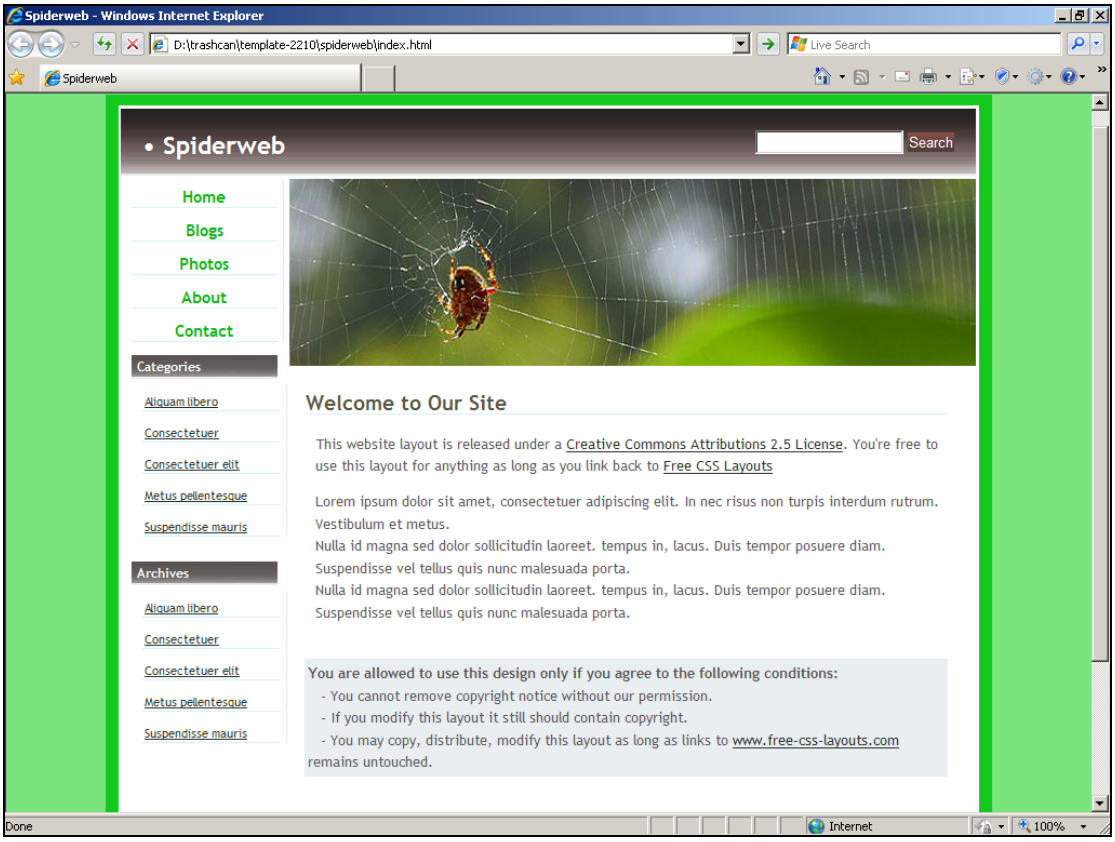


The point to remember here is that the HTML file which contains the page content (ie, the text such as "Littering a dark and dreary road...") is the same each time. It hasn't changed. All that changes is the CSS file that it's linked to, which controls the position and appearance of the text, and the positions and names of the image files.

If you want to know how more about how it's done, order a copy of "The Zen of CSS Design", written by the authors of the site. It's available from Amazon and just about every major bookstore.

Page Layouts and Div Tags

So far, our look at CSS has concentrated on visual appearances. We've covered how to specify fonts, sizes, and colours. But CSS is much more than that. Most importantly, it also covers the layout and arrangement of pages. For example, look at this page.



Notice that the page is divided into two columns. Actually, the top section of the page, which contains the Spiderweb title and the grey bar, uses the full width of the page. The remainder of the page is split into two vertical columns. The left-hand column holds some links, while the main content of the page is on the right hand side.

This layout is typical of many web sites. They use a two- or three-column layout in order to fit all the required content on the page in a logical place. Having the page's collection of links and menus (known as the navigation, or nav for short) in a separate column like this also means that the main content is narrower than the full width of the screen. This makes it easier for people to read the page on screen.

So, how do we use CSS to create a page layout like this? The key is the tag. A div, in this context, stands for a divider or a division. It lets you separate entire blocks of your page,

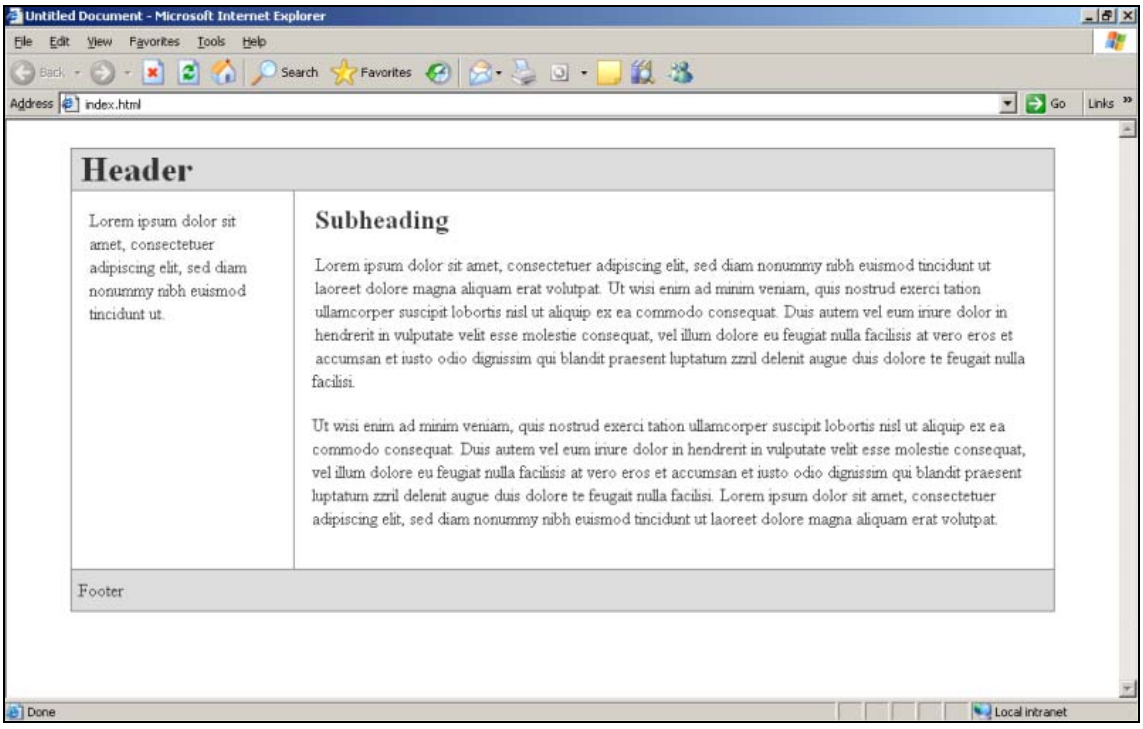
including other HTML tags, into discrete items whose layout can then be controlled by the css file.

A div is not like any of the HTML tags we've encountered so far. It's not the sort of thing that you'd assign a font or colour to. Instead, you use css to specify its position on the page, relative to the top left hand corner or to other divs. You can also specify its background colour, border, margin and so on.

By carefully nesting your divs, ie opening a new one before closing the current one, you can create the hierarchy that leads to the columnar effect shown above.

So in the above example, the page comprises 4 divs. First is the overall outer div which holds the entire page. Within this are 3 more divs, to create the top panel, the left hand column and the right hand column.

As a simpler example, let's use divs and css to lay out a page. Here's the effect we want to end up with:



Here we have a very common web page format. A header line at the top, a left and a right hand column, and a footer area at the bottom.

Here's what the body section of the HTML file looks like, for this page. I've added some line numbers to assist in the explanation which follows – these are not part of the HTML and you should remove them if you want to paste this into Amaya to experiment further.

```
1  <body>
2  <div id="container">
3  <div id="top">
4  <h1>Header</h1>
5  </div>
6  <div id="leftnav">
7  <p>
8  Some text goes here…
9  </p>
10 </div>
11 <div id="content">
12 <h2>Subheading</h2>
13 <p>
14 Some text goes here…</p>
15 <p>
16 Some text goes here
17 </p>
18 </div>
19 <div id="footer">
20 Footer
21 </div>
22 </div>
23 </body>
```

You may be surprised at just how short and un-complicated it is. Let's take a look at this code in a little more detail, because understanding the basics of div-based page layout is vital if you want to know more about how web sites work.

On line 2 we create a div and we name it container. This is the overall div that contains the rest of the page. It doesn't matter what names you choose for your divs, but many web people choose to call the main div the container.

On line 3 we create another div called top. This is effectively inside the container because we haven't yet closed the container div. The top div gets closed on line 5, and its contents (just an h1 heading) appear on line 4.

Between lines 6 and 10 we create another div called leftnav, the contents of which is just a single paragraph. Lines 11 to 18 hold the content div, which is the right-hand column in the illustration above. Lines 19 to 21 create our footer div.

Finally, on line 22, we close the container div.

With the page created, we can now create the css file for it. Here's what it looks like. You'll see that I've used a slightly different formatting layout, with the { symbols on a separate line. It's quite OK to do this if you prefer it.

```
#container
{
width: 90%;
margin: 10px auto;
background-color: #fff;
color: #333;
border: 1px solid gray;
line-height: 130%;
}

#top
{
padding: .5em;
background-color: #ddd;
border-bottom: 1px solid gray;
}

#top h1
{
padding: 0;
margin: 0;
}

#leftnav
{
float: left;
width: 160px;
margin: 0;
padding: 1em;
}

#content
{
margin-left: 200px;
border-left: 1px solid gray;
padding: 1em;
max-width: 36em;
```

```
}

#footer
{
clear: both;
margin: 0;
padding: .5em;
color: #333;
background-color: #ddd;
border-top: 1px solid gray;
}

#leftnav p { margin: 0 0 1em 0; }
#content h2 { margin: 0 0 .5em 0; }
```

The first section of this css file is #container. This, if you remember, means that we're about to specify the styles for the HTML tag whose id is container. Which just happens to be our main div. Of primary interest is the fact that we're setting its width to 90%. This means that the overall width of our page will be 90% of the screen width, on whatever screen the visitor happens to be using to view our site. If you look at the picture of the page, the remaining 10% is taken up by the vertical areas of white space to the left and right of the main text area.

The second section, #top, is used to specify a few more things about the top div. Below this is a section entitled #top h1. This is something we've not encountered before, and is yet another useful feature of css. In this case, it means that the styles which follow are only to be applied to level-one headings within the #top div.

The final 2 sections, #leftnav p and #content h2, use the same technique. They specify what the standard paragraph text will look like in the leftnav column, and how the h2 heading in the content pane will appear.

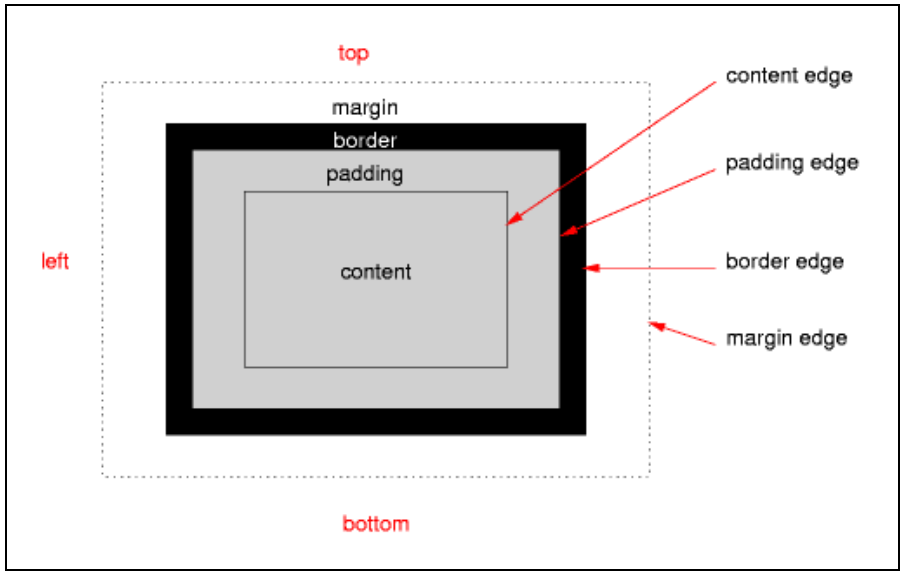
Although this may all seem very complicated at first, it's fairly easy to get used to with some practice. If you want to use this example as a starting point for your own pages, just type or paste the HTML code and the css file into Amaya, and then tweak the content and styles as you wish. By experimenting with TopStyle Lite to change the styles, you can make the page look just how you want it to.

If you want to upload the finished product to your hosting space, remember to upload both the css file and the HTML file. Otherwise the layout will look completely wrong. IN addition, you'll also need to add HTML and BODY tags to the html file, as well as linking the style sheet to it.

The CSS Box Model

Using css to create page layouts, rather than just using it to format your text, is complex. You would be well advised to start with a ready-made design from a site such as **`www.opendesigns.org`** rather than designing your own from scratch. But if you do want to learn more about the intricacies of CSS for page layout, there are plenty of good books and online resources available. A Google search for terms such as "css tutorial" or "learning css" will get you started.

One of the most important concepts in css-based page layout is something called the box model. It explains how padding, spacing, justification (left, right, centred etc) and margins interact to affect the final position of an item on a page.



It is beyond the scope of this book to cover this topic in any more detail, but you can find out more by searching online for "css box model".

Pictures On Pages

To put a picture on one of your Web pages, here's what you need to do:

1. Obtain the picture as an image file (typically .JPG or .TIF) from your scanner, camera or any other suitable source.
2. Copy the image file into your web folder along with the HTML files that make up your site
3. Add an <img> tag to the HTML page at the point where you want the picture to appear.

Let's try doing this now.

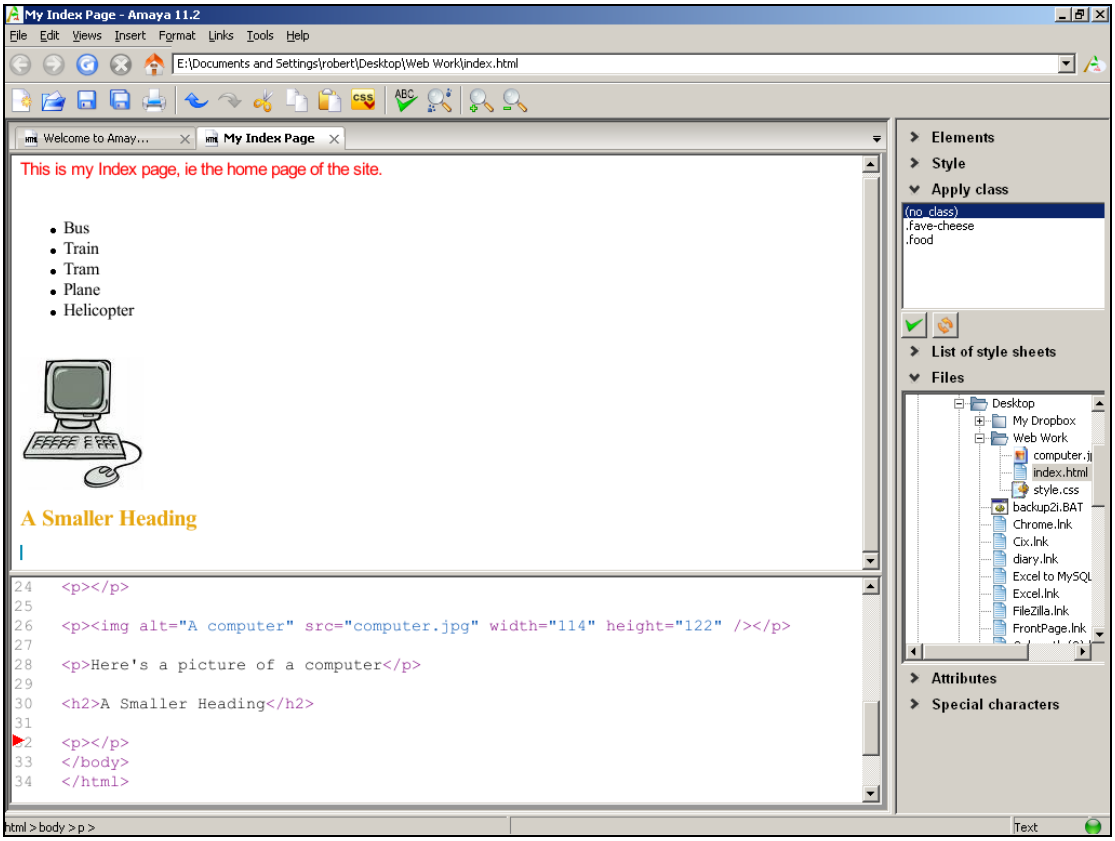
First we need to find a picture to use. If you have something to hand, copy it to your Web Work folder. If you don't have any images, visit your favourite web site, right-click one of the pictures on the page and choose "Save Picture As", and save it in your Web Work folder.

You might at this point be somewhat surprised to discover that stealing pictures from other people's web sites is as easy as right-clicking. Sadly, this is generally the case. Never assume that pictures on your web site are safe from others who want to use them. If this is a problem for you, eg if you're a professional photographer or artist and you want to show off your work online without others helping themselves to it, the best deterrent is to upload only small thumbnail pictures onto your public site. Anyone who wants to see full-size pictures can then be asked to pay before being allowed into a separate, subscribers-only area of the site. Another option is to use a graphics program to add a watermark or logo to your pictures, but this detracts from their overall look.

With your picture safely stored in your Web Work folder, open Amaya and load one of your HTML pages. From the Insert menu, choose Image. Then browse to your Web Work folder and select the picture you placed there earlier.

In the space provided, type some "alternate text". This is the text that a visitor's web browser will display if the visitor hovers their mouse over the picture. It's also the text that will be read out aloud to anyone who visits your site with the aid of some screen reader software. Therefore it's a good idea to always include some alternate text (or alt text, as it's known) for every picture you use on your site. In fact, Amaya won't let you leave this box blank so you have no choice in the matter.

Your chosen picture will now appear on your web page. Here's how it looks on my computer. Note that I've used the Show Source option to show both the standard page and the HTML source code:



Notice the <img> tag in the lower window, which specifies the name of the image file, the alt text for it, and the width and height of the picture.

If you want to see your finished page online, upload both the HTML page and the image file to the hosting space on your server. Don't forget to always upload all of the image files that your site uses. If you don't, and a visitor's web browser encounters a page where the <img> tag references a picture file which doesn't exist on the server, the picture on the page will typically be replaced by an empty box with a red cross in it. If you've ever seen a page on the web which looks like this, now you know why. The person who created the page forgot to upload the image file, or they renamed the image file on the server without updating the <img> tags on the page to match.

About Image Sizes

The dimensions of a picture file on a computer are generally measured in pixels, and the overall size of the file is, like any other file, measured in KB and MB (kilobytes and megabytes).

The typical person who visits your web site will have a screen size of around 1024 pixels wide by 768 pixels deep. So if you want a picture which fits on the screen alongside some text and

navigation, you'll probably want to ensure that the image is roughly 300 pixels wide and 200 deep. Equally, if you want the picture to take up almost all of the screen, a size of 1000 by 700 might suffice.

However, the typical image from a modern digital camera is often more like 3000 by 2000 pixels. Take something from your scanner in high-resolution scanning mode, and it could even be 3 or 4 times that figure. The picture will thus be far too big to fit on the screen. The web site visitor will have to scroll within their browser to see the full image.

Some web developers get around this by changing the width and height settings in the <img> tag. This forces the browser to resize the image to the required size. However, it doesn't change the overall size of the image file on the server, in terms of its KB and MB. A picture from your camera that's 3000 pixels wide, in a file that's 4 MB, will still occupy 4 MB of server space if you alter the <img> tag to display the picture at 200 pixels wide. It will therefore still download very slowly onto your visitors' computers.

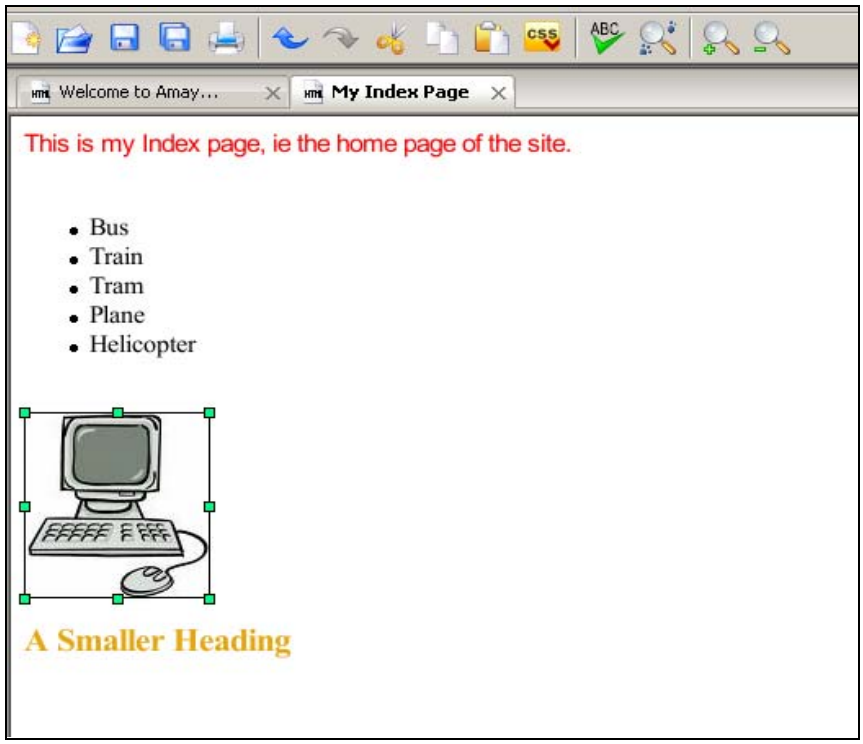
So, here's a vital rule about pictures on web sites that you should always follow. If the picture you want to use on your page isn't the right size, use an image editing program to resize it *BEFORE* placing it on your web page. ***Never*** resize a picture by changing the width and height settings in the img tag.

If you don't have a program that can resize pictures, upload the image to somewhere like **`www.piknik.com`** and use their online tools.

Pictures As Links

On page 48 we learned about hyperlinks, and how to turn a snippet of text into a clickable link which leads to another page or site. You can also turn an image into a hyperlink too, so that clicking on the image takes the user to another location.

First, select the picture from within Amaya by dragging over it with the mouse. It will be surrounded by a border like this:



Then simply go to the Links menu, select "create or change link", and enter the required page to which the visitor will be taken when they click the picture.

Note that some browsers add a border around any picture which is used as a link, and this may mean that the picture doesn't look quite as you want it to. To get around this, set the picture's border width to zero somewhere within the <img> tag for the picture. For example:

```
<img src="picture.jpg" alt="This is a picture" border="0">
```

Finding Images to Use on Your Site

Stealing images from web sites to use on your own is both morally and legally wrong. There's no harm in borrowing such a picture as a training exercise, which is what we're doing here, but it's certainly not advisable to do it on a real web site. If you need an image to illustrate your site but you don't have your camera to hand, or you don't have a suitable image, help is thankfully at hand. There's a variety of web sites which offer royalty-free pictures that you can use on your sites without payment. My favourite is at **www.sxc.hu**, which is registered in Hungary but is presented in English.

A Browser Icon for your Site

Have you ever noticed how some web sites have their own special icon that appears in your web browser's list of favourites, and on the browser's address bar? It also appears on the desktop if you create a shortcut to the site. An icon like this can help to make your site stand out on your visitors' computers, and also adds an extra degree of professionalism to your design.

For example, if I add the BBC News web site to my list of favourites, notice the small black BBC icon in the address bar and in my Favorites menu:

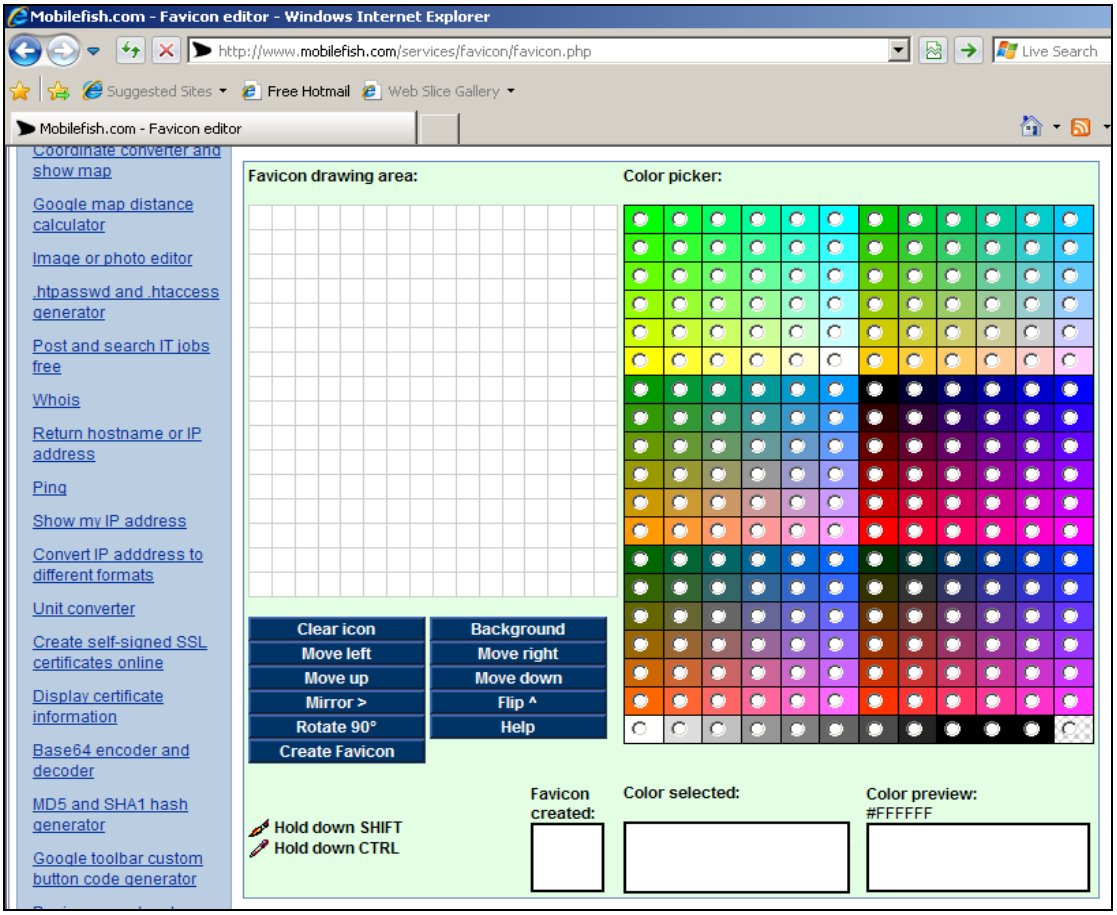
This sort of icon is known as a "favicon", because its most common purposes is to appear on your visitors' lists of favourite sites. If you want one for your site it's not hard to do. What you need is a graphics program which has the ability to create and save an image in favicon format.

While there are many such programs around, some of which are downloadable for free, it doesn't really make sense to install a program on your computer that you won't use very often. There are quite a few online favicon editors that you access via your web browser, and these work just as well. A Google search for "favicon editor" or "online favicon editor" will bring up lots of them, so just pick one that works for you.

As an example I'll use the one at:

http://www.mobilefish.com/services/favicon/favicon.php

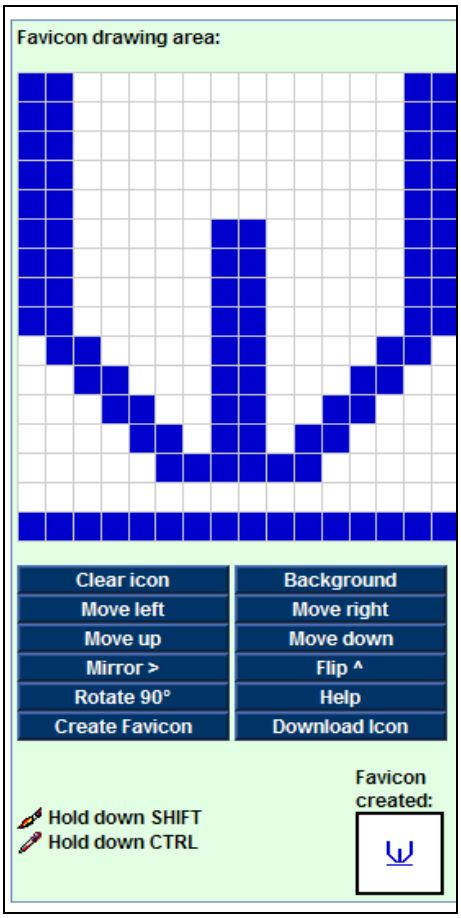
It looks like this:



To create your favicon, click the pixel boxes in the drawing area. The colour picker allows you to choose the colour of each pixel.

Note that a favicon is a fixed size of just 16 by 16 pixels (some operating systems and browsers support larger ones, but not all do, so stick to 16x16 if your editor offers you a choice of sizes). This means that your design will necessarily be fairly crude. Don't try to cram too much information into a favicon as it just won't look right. Equally, resist the temptation to use an editor which claims to be able to convert a digital photo or other full-size image into a favicon. You can try it, but the results won't be particularly appealing or comprehensible.

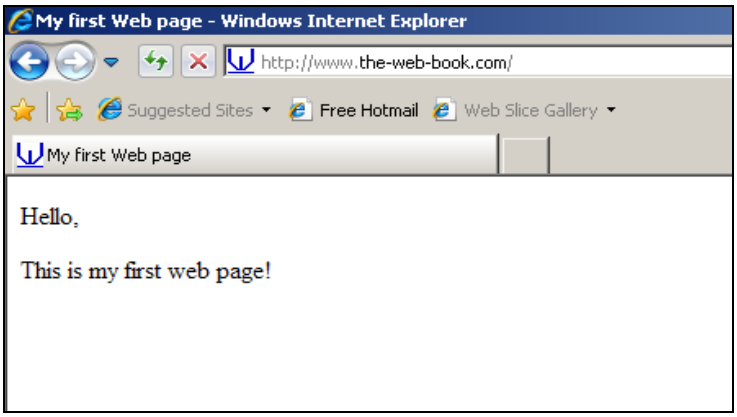
As you draw, click on the Create Favicon button to see how things are coming along. Here's my attempt at being a favicon artist. The design is a cross between a pen nib and a letter W, in case you're wondering!



With the design complete, click Download Icon and the site will download a file to your computer called favicon.ico. Don't change the name of the file or it won't be displayed properly by your computer or web browser.

All that remains is to upload the file to your web site using FileZilla. The rules about favicons state that the file must be uploaded to the root folder of the site. In our case, this means the public_html folder. If you put the file in any other folder on your site, it won't work.

Having uploaded the file, we can now test it. Open your web browser, surf to your site, and you should see something like this:



Notice how our new favicon appears in the browser's address bar and also on the Internet Explorer tab.

You can even get the favicon to display on your desktop. Just right-click on your Windows desktop and then choose New, Shortcut. For the location of your shortcut, type the web site's address. In this case **`www.the-web-book.com`**. For the name I'll just use TWB, to keep things concise. I now have, on my desktop, a direct shortcut to my web site, which even shows the proper icon:

If the favicon doesn't appear and you get a generic browser icon instead, double-click the icon to visit the site. This should force Windows to download the favicon, after which the display should be correct.

Short Cuts to Great Web Pages

Now that you know the basics of HTML and CSS, you're probably itching to create some real pages. However, you're probably also feeling rather daunted, having realised that creating a web page which looks good isn't going to be as easy as you might have hoped.

Thankfully, help is at hand. There are lots of ready-made page designs that you can download for free from the internet, and then customise as you wish. It's much easier to start with someone else's design and then adapt it, rather than creating something from scratch. However, even if you're starting with someone else's work, you will find it difficult to customise the page unless you know the basics of HTML and CSS first. Hence the previous chapters.

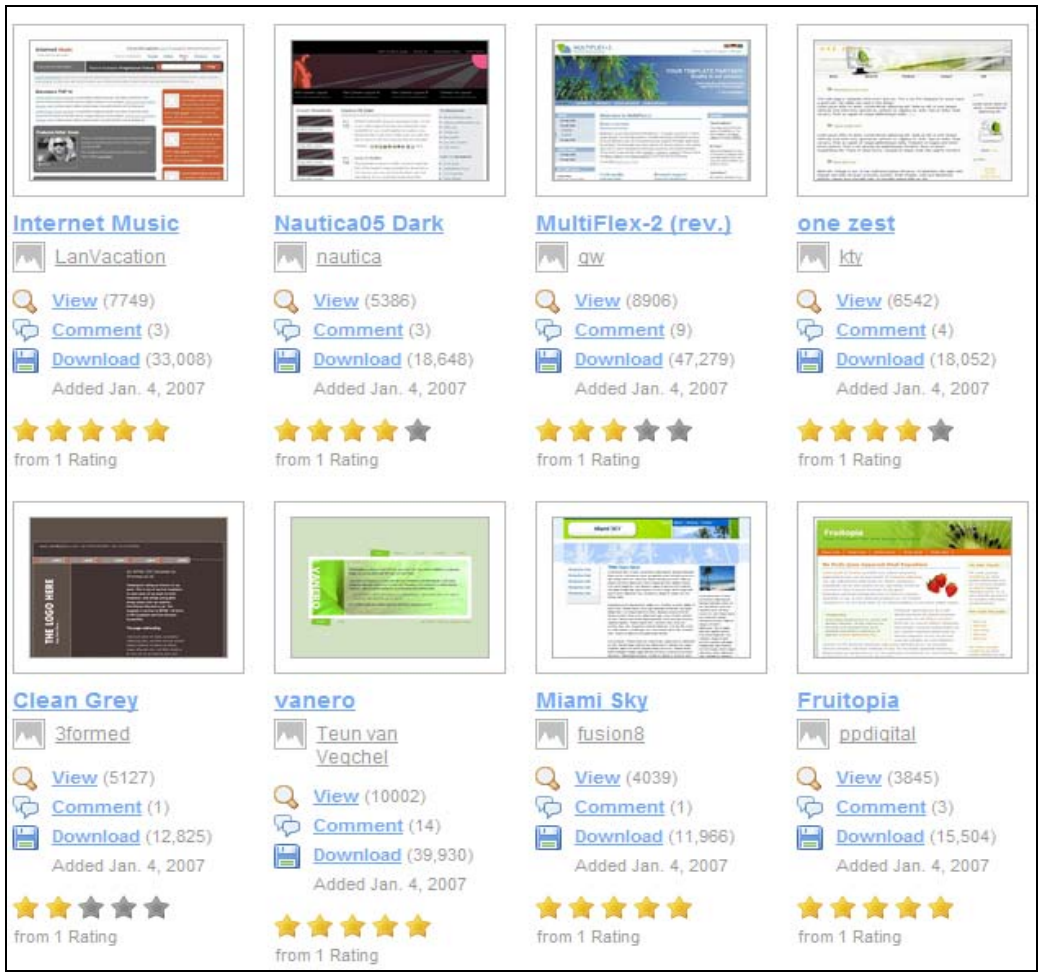
Two of the best-known sites that offer free, ready-made, CSS-based templates are:

Open Source Web Designs	**`www.oswd.org`**
Open Designs	**`www.opendesigns.org`**

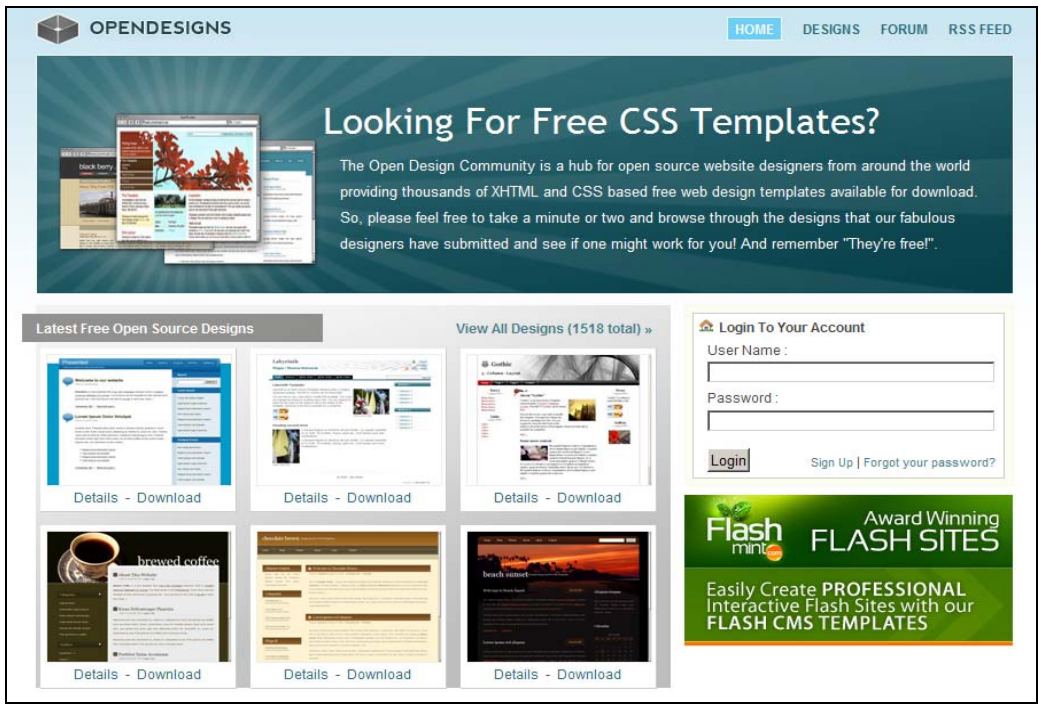
Both sites are very similar in nature. You browse the list of available designs, and then download any that you want to use. Each design normally consists of a single index.html file containing the sample design, an accompanying style.css file, and a collection of pictures that the site uses. Once you have download all of this, you turn it into a usable site by doing the following:

1. Tweak the CSS in order to fine-tune the look and feel
2. Tweak the index.html file too, if required
3. Copy the index.html file as many times as necessary, in order to create each of the pages for your site
4. Tweak or replace some or all of the image files, if required, to make the site more relevant to your audience.
5. Upload the finished set of files to your hosting space.

Here's an extract from OSWD, so you can see the sort of material that's available:



And here's a similar page from opendesigns:



Both sites are similar in what they offer, and each contains around 2000 separate designs. Remember, though, that what you get is not a finished web site. It's just a single page, nicely designed, that you can use as a building block.

Using an Open Source Design

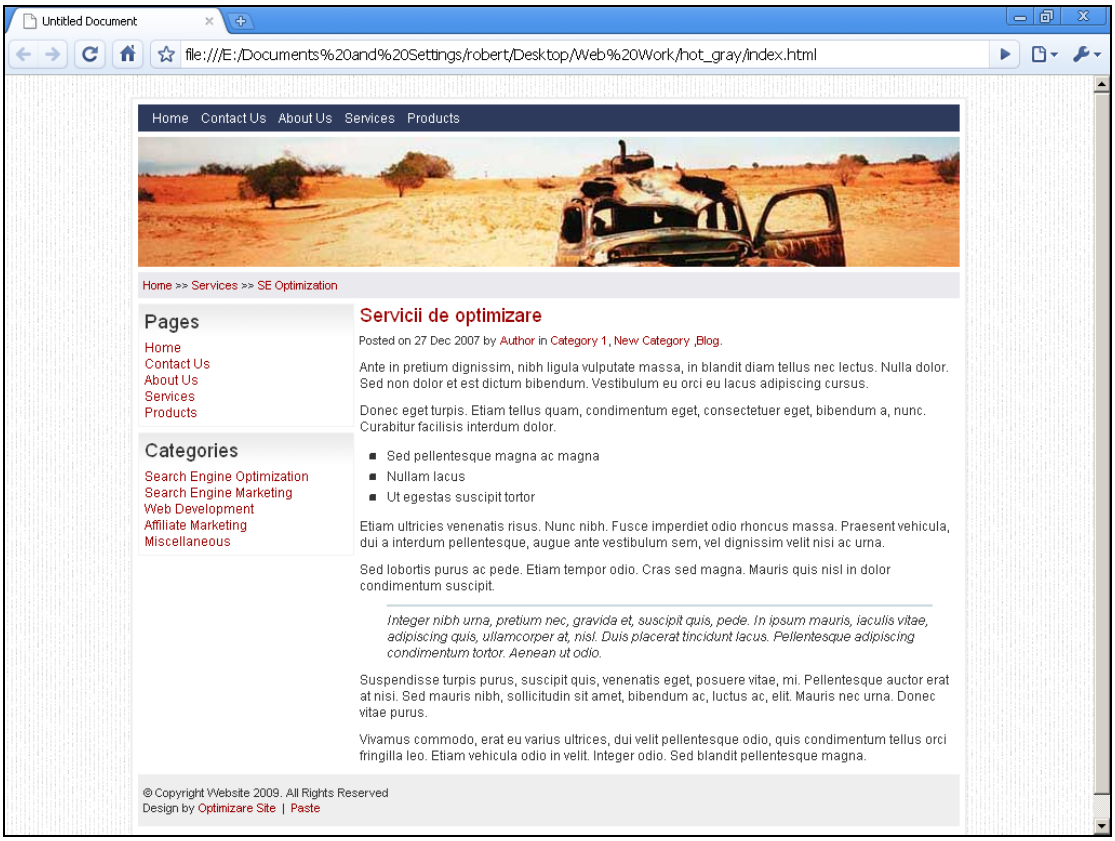
As an example of how to download and use a freely available web design, let's take a look at one of them in detail. In this case, the design comes from opendesigns.org and is called Hot Gray.

To start, locate the design on the web site and download it. It'll arrive on your computer as a .zip file, which you need to extract. Just right-click the file and choose Extract All. You can now click into the folder, within which will be another folder called Hot Gray. Within there will be files you actually need:

The license.txt file is a text file which spells out what you can and can't do with this particular design. Although most of the free web designs have licences and restrictions, the restrictions are rarely tough. In the case of Hot Gray, the licence file merely stipulates:

> Ok, I'm gonna be straight here, no legalese: this theme is free for personal and commercial use as long as the footer links are kept intact.
>
> You don't have to link from every page of your site if you don't want to, only the front page.

Sounds perfectly acceptable to me. So, copy the Hot Gray folder to your Web Work folder in order to keep all of your web-related projects in one place. Then double-click the index.html icon so we can see what the page actually looks like in a web browser:



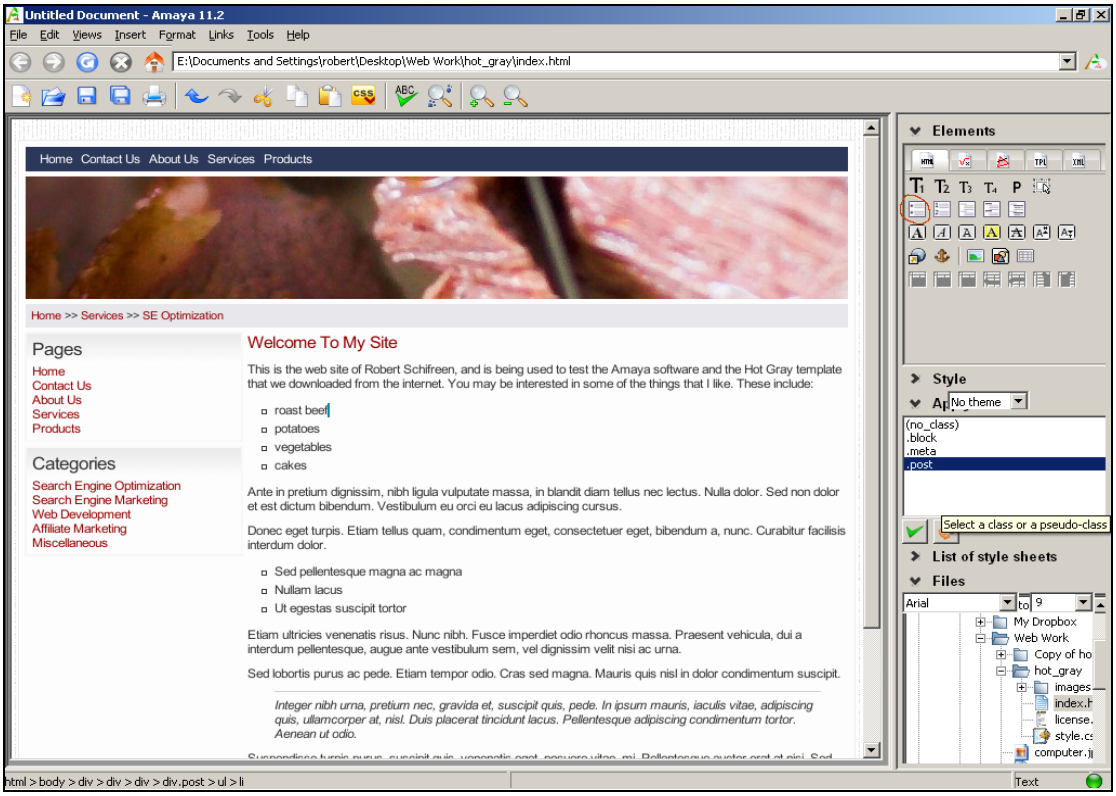
Looking pretty good so far, considering that we haven't had to actually do any design work at all. This is more like it!

Tweaking the Text

Changing the sample text that's on the index.html page supplied as standard is simplicity itself. Load the file into Amaya, and just type over what's already there. Delete anything that you don't want.

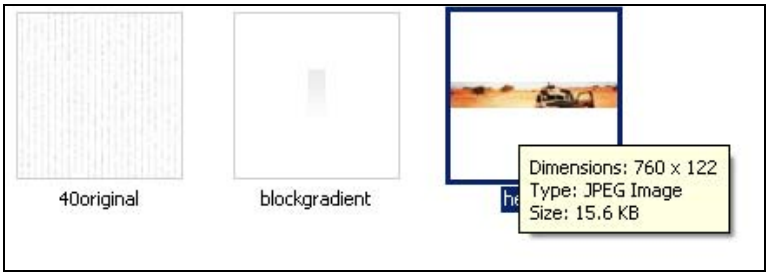
In the example below, I've replaced the main text with a paragraph of my own. In addition, I've created a short list of foods, and used the Unordered List option in Amaya (the button on the right hand panel that I have ringed in red) to format the list in the style as defined in the css file. In this case, for the Hot Gray design, the normal rounded bullets are replaced by small rectangles.

I've also deleted the "Posted on..." line under the main heading. This is useful for blog-type pages or online newsletters but isn't really necessary here. The relevant entries are still in the CSS file, so this particular style can still be used on other pages within the site if required later.



Changing the Pictures

You'll also see from the picture above that I've changed the header picture from a desert scene to some tasty roast beef. How was this done? Take a look inside the images folder of your Hot Gray design. Browsing in thumbnail mode within Windows, we find the following:



The 40original and blockgradient images are used to make up the border around the page, and the grey bars behind the menu lines. We can leave these alone for now. The file we're interested in is header.jpg. Hovering the mouse over the thumbnail brings up a summary box which tells us that the image is 760 by 122 pixels. So, in order to customise the design but without affecting the layout, we simply need to replace this image with something else that has the same name and is the same size. A quick visit to the free image site at **`www.sxc.hu`** that was mentioned earlier is all that's required.

You'll also need to crop the picture you download to a size of 760 by 122. If you don't have a program that can edit and crop pictures, search the web for The Gimp, which is a great image editor that's totally free. Or use the online image editor at **www.piknik.com**.

Changing the CSS Styles

If you want to change the text styles that the Hot Gray design uses, you'll need to change the CSS entries in its style sheet file. Loading the css file into TopStyle Lite, the first section looks like this:

```
body {
   margin: 0;
   padding: 0;
   text-align: center;
   background-color: #FFFFFF;
   background: url(images/40original.gif) repeat;
   font-family: Arial, Helvetica, sans-serif;
   font-size: 12px;
   font-weight: normal;
   color: #333333;
}

h1, h2, h3, h4 {
   margin: 0;
   padding: 0;
   font-weight: bold;
}

p {
   margin: 8px 5px;
}

a {
   text-decoration: none;
   color: #990000;
}

a:visited {
}
```

Scroll down and you get to the next set of styles, namely:

```
a:hover {
   border-bottom: 1px dotted;
}

ul {
   margin: 0;
   padding: 0;
   list-style: none;
}

/* Layout */

#container {
   width: 760px;
   margin: 20px auto;
   padding: 5px;
   border: 2px #eeeeee solid;
   text-align: left;
}

#links {
   width: 750px;
   padding: 5px;
   background-color: #2D3A5C;
}

#header {
   width: 760px;
   margin: 5px 0;
   height: 120px;
   background: url(images/header.jpg) no-repeat;
}
```

There are others, too, but this will suffice for the moment. Let's take a look at some of these style definitions, and decipher what they mean.

First is the style for the contents of the <body> tag, which governs some of the overall look for the entire page. We can see, for example, that the background image is specified here. The "repeat" bit means that, although the image itself is small, it will be tiled across the entire page as needed.

Next, the file specifies that headings 1 to 4 have no margin or padding, and are in bold type.

Next, we set margins around each paragraph (ie, the <p> tag). But why 8px and 5px? You can specify up to 4 values in instances such as this, to affect margins, padding etc all around an item. The 4 values start from the top and work clockwise around it. So, for example, a margin of **`4px 7px 10px 3px`** means 4 pixels at the top, 7 to the right, 10 underneath and 3 pixels to the left. In this case, only the first two values have been specified, so each paragraph will have an 8 pixel margin above it and 5 pixels to the right.

The <a> tag stands for "anchor", and is the one we use for creating hyperlinks. In the style definition for the <a> tag, text-decoration:none is how we remove the underline that

browsers otherwise add to links. Equally, we change the standard link colour from blue to red with the `color: #990000` line.

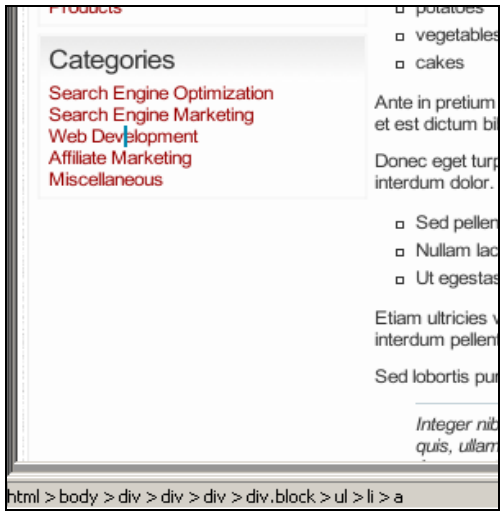
The a:visited and a:hover styles allow you to specify the style for links that the user has already visited. Most browsers normally display these in purple, but by setting a style for a:visited you can change this. Equally, a:hover is the style that a hyperlink has when the visitor has his or her mouse hovered over it. In this case, we use the <a> style to turn off the underline and then use a hover style to turn it back on. This gives the effect of making the underline appear only when the mouse is hovered over the link.

Which Style Is This?

One of the problems that you'll almost certainly encounter when you're trying to decipher how the sample index.html page and the style.css file work together, is how to tell which style or styles are governing the appearance of a certain item on the page. For example, look at the screenshot above, where I've added the "Welcome to my site" heading. Now look across to the Categories subheading on the left hand side of the screen, and particularly the "Web Development" line. If you wanted to change the appearance of that text, which entry in the css file would you need to change?

In an ideal world, the structure and content of the css file would tell you all you need to know. And indeed there's a section in the file called #links, which, if you remember our first lesson on CSS, means that this is the style for the item on the page which has an id of "links". There is indeed such an item, namely a tag that consists of `<div id="links">`, but this, if you look through the HTML code of the index page, clearly refers to the links under the Pages heading and not the entries under the Categories heading. So it's back to square one.

There are two techniques that can help you track down the css entry that affects an item on the page. Firstly, use Amaya's status line. Load Amaya, open the page in question, place your cursor somewhere within the text that you're interested in, and look at the very bottom of the screen. In the case of the page in question, you'll see this:



The status line shows the hierarchy of HTML tags leading up to the item on which the cursor is positioned. So in this case, it's the overall HTML tag, followed by body, then a selection of divs, then a div which is configured to use a class called "block". Within that div, we're looking for an unordered list, and list item within that, and a hyperlink (the <a> tag) within that.

The second technique is to look at the HTML code near the text you're interested in. View the HTML source with Amaya (or even load it into WordPad or Notepad, if you'd rather). Search for "Web Development" and what you actually see is:

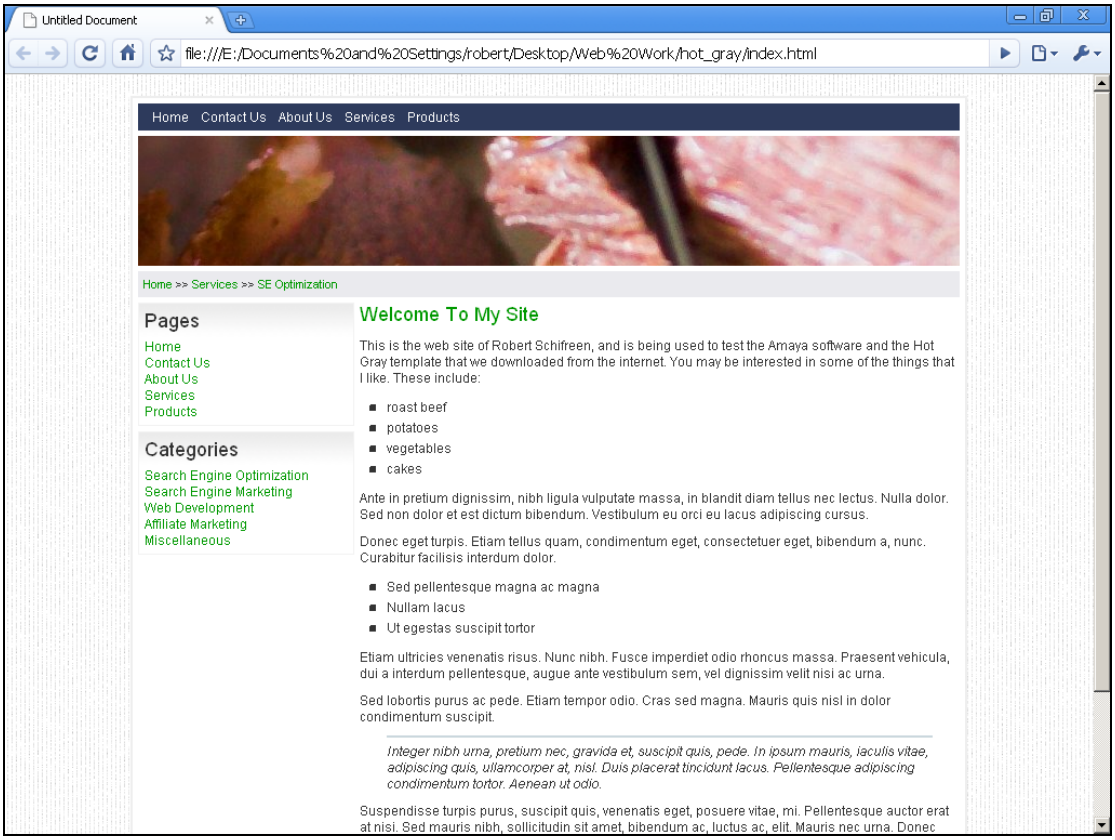
```
<div class="block">
<h3>Categories</h3>
<ul>
  <li><a href="/">Search Engine Optimization</a></li>
  <li><a href="/">Search Engine Marketing</a></li>
  <li><a href="/">Web Development</a></li>
  <li><a href="/">Affiliate Marketing</a></li>
  <li><a href="/">Miscellaneous</a></li>
</ul>
</div>
```

As we can see again the Web Development entry is a link within a list item, within an unordered list, within a div that is to be formatted with the css class called "block".

Although this looks cut and dried, it's not necessarily the end of the story. If you look further at the HTML source code, you will see that the above-mentioned div is actually contained entirely within another div that has an id of "menu", and that div itself is within yet another one called "contentwrapper". Any or all of which could be affecting our text, but at least we have some good clues as to where to look in the css file.

Let's start with the entry at the very end of Amaya's status line, and work backwards. It's an <a> tag. The css file has only one mention of the <a> tag, so that's a good place to start.

Let's change the entry from **`color: #990000`** to **`color: #009900`** and try viewing our index page again:



Success! Our links are now green rather than red.

Adding Pages and Navigation

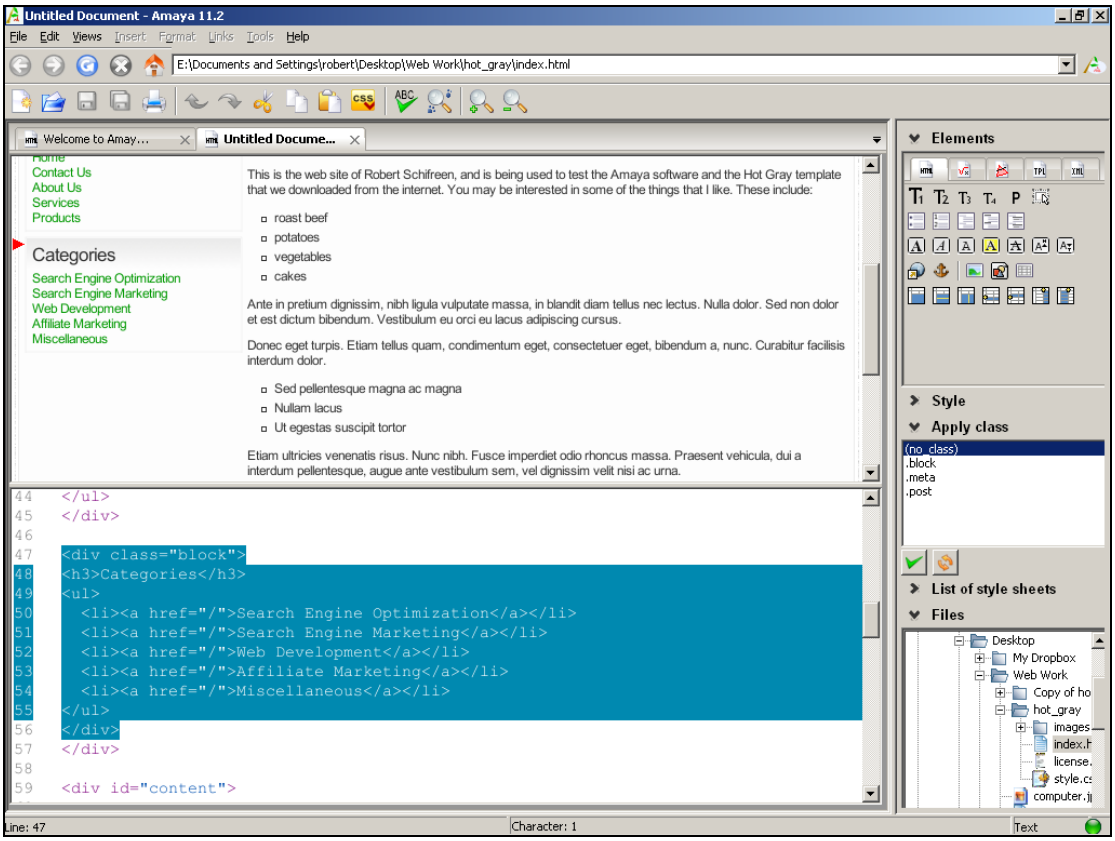
With our downloaded template now tweaked so that it looks how we want, all that remains is to create additional pages and get the navigation (ie, the set of links down the left hand side) working.

When you download a template that includes links, it's common for those links not to actually lead anywhere when clicked. If you look at the HTML source code for the page, and specifically the <a> tags which comprise the links, you'll see lines that look something like **`<a href="#">`** or **`<a href="/">`**. The entry in quotes after the href command is the destination for the link, and would normally comprise a complete URL or at least an HTML file name. Instead, these links are all empty so we need to populate them.

First, we need to decide on the structure of our site. We don't need the Categories section, so that can be deleted. In the Pages section, we'll keep the first 3 (Home, Contact Us and About Us). We'll delete the last 2, and replace them with a new entry called Blog.

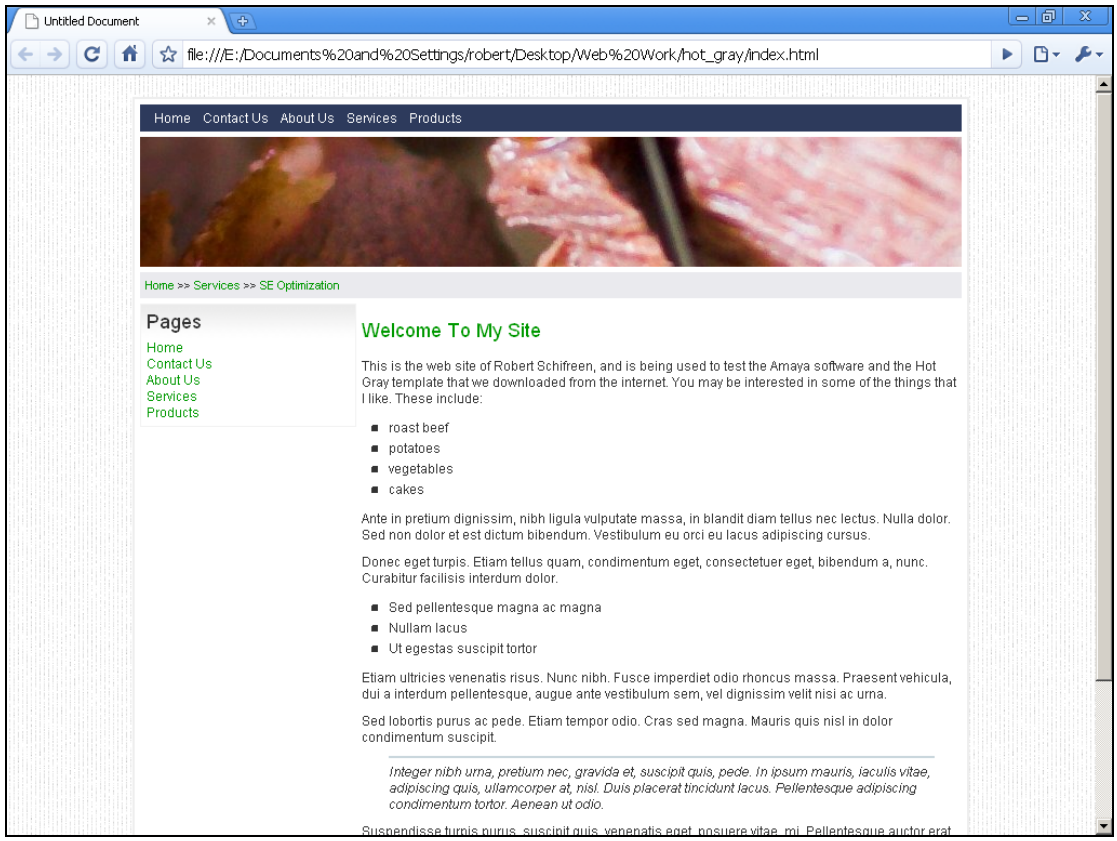
So, to start, load the index.html page into Amaya. First look for the section of code that comprises the Categories block. It's safer not to just select the whole block with the mouse and hit the Delete key, as you risk leaving a stray HTML tag in place. Much better to switch to HTML code view and do it properly.

Here's the page loaded into Amaya, with the relevant block of code selected:

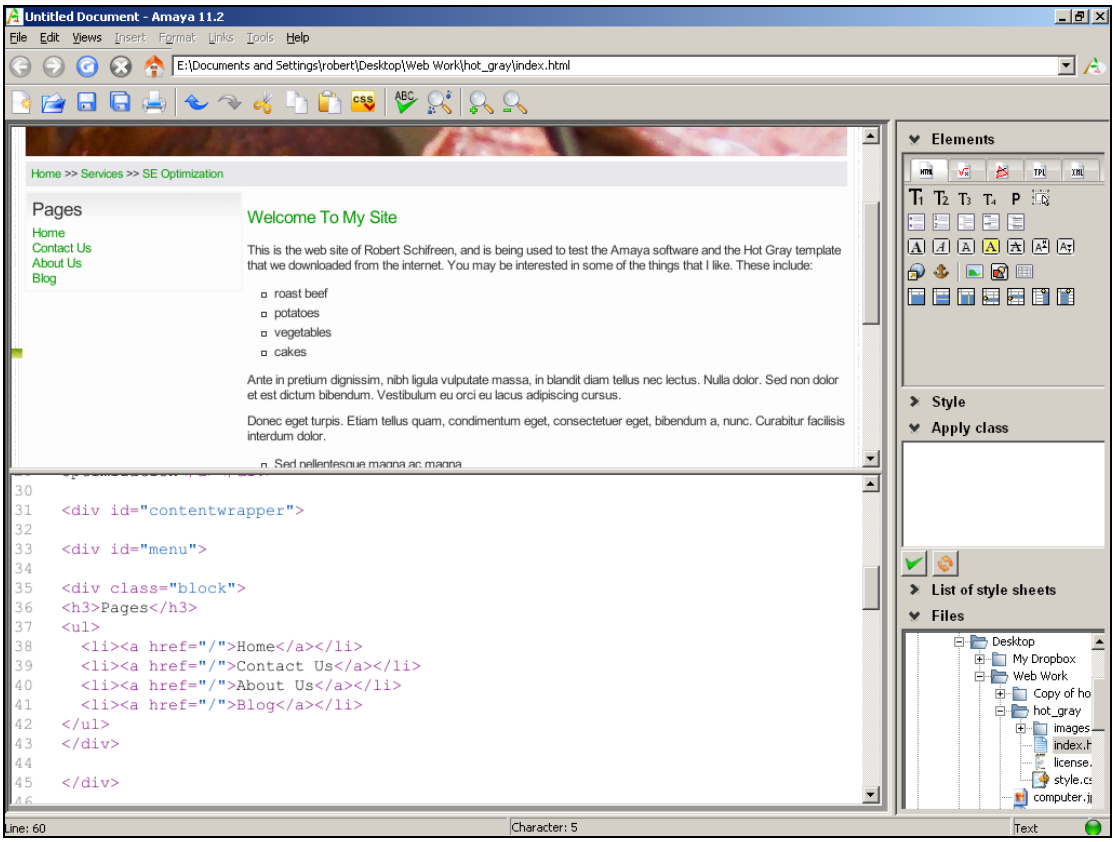


We're about to delete an entire section of the page, which starts with a tag and ends with . Don't delete the second of the two tags, as that closes a div which is opened earlier, and deleting it would lead to an imbalance. And a page that won't validate correctly.

Press Backspace to delete the selected block of code, then save the HTML file. Open the page in your web browser, rather than Amaya, and the Categories section should now be gone:



To remove the final two entries in the Pages section, and add a link to an as-yet-nonexistent blog page, search for the Pages section in the HTML code and, using Amaya, change it to:



That's the text of the links. Now we just need to set up the destinations. So, in Amaya's page view (you can close the source code view for now), click somewhere on the green Home link and then, from the Links menu, choose Create Or Change Link. In the box, change the destination to index.html, as that's our home page.

Change the Contact Us link to point to contact.html in the same way, and the About Us link to point to about.html too. Finally, set the Blog link to blog.html as well.

All that remains is to actually create the pages that we've now started referring to. So use the Save As option from Amaya's File menu to save 3 more copies of the index page, called contact.html, about.html and blog.html. You can then edit those new pages, again with Amaya, to contain the relevant information according to the name of the page. Remember not to change any of the navigation elements because, if you do, you'll need to make the same changes across all of your pages. That's why we made sure that the page looked correct before we started to make the additional copies.

Uploading the Finished Files

With everything done, it's time to test your site. Upload your 4 HTML pages plus the CSS file to your hosting space, using FileZilla. Then, using your Web browser, surf to the index page

and try clicking on the links. You should find that you can navigate around your brand new site quickly and easily.

Rules, Tables and Image Maps

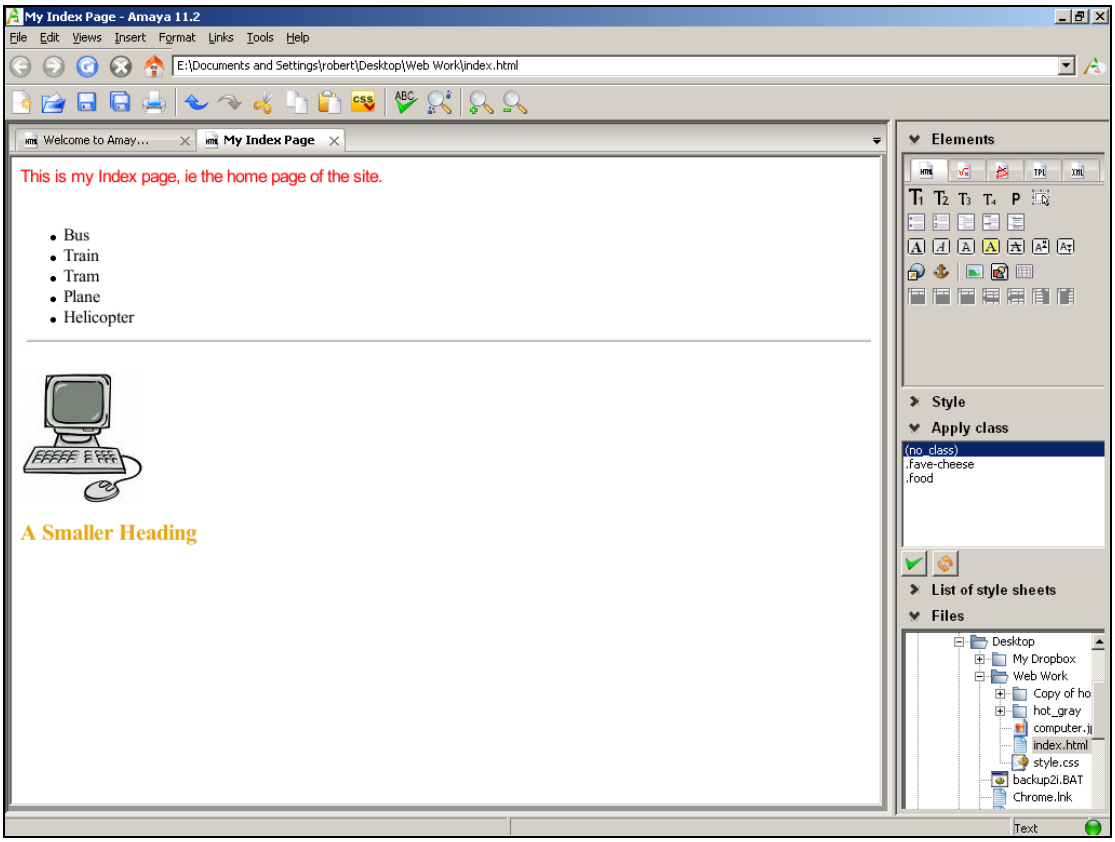
Now that you're building up your knowledge of HTML in order to create web pages, and the use of CSS to specify how each HTML tag looks, we'll cover three more features of web pages that you might find useful.

Horizontal Rules

A horizontal rule allows you to create a line across your web page. This is useful as a way of underlining headings, or to separate areas of a large page. Although, if a page is becoming particularly large, it's always better to split it into multiple pages in order to save your visitors' time and bandwidth. For example, if you're putting together the web site for your school's sports day and there's a huge list of race times that requires visitors to scroll down the page in order to find their name, split the page into separate parts. "Click here if your surname begins with A-K, or here for L-R, or here for S-Z". After all, Smith is unlikely to be interested in Brown's results, so don't waste his time including them on the same page.

The standard HTML tag for a horizontal rule is just <hr>. Because this tag doesn't officially need to be closed, and yet modern HTML coding standards say that every tag needs to be closed, it's good practice to use the version which includes a closure. Namely **<hr />** rather than a plain **<hr>**. It'll help ensure that your pages validate correctly.

Amaya has the built-in ability to create rules. Just load one of your pages, then go to the Insert menu and choose Horizontal Rule. Your page will now look something like:

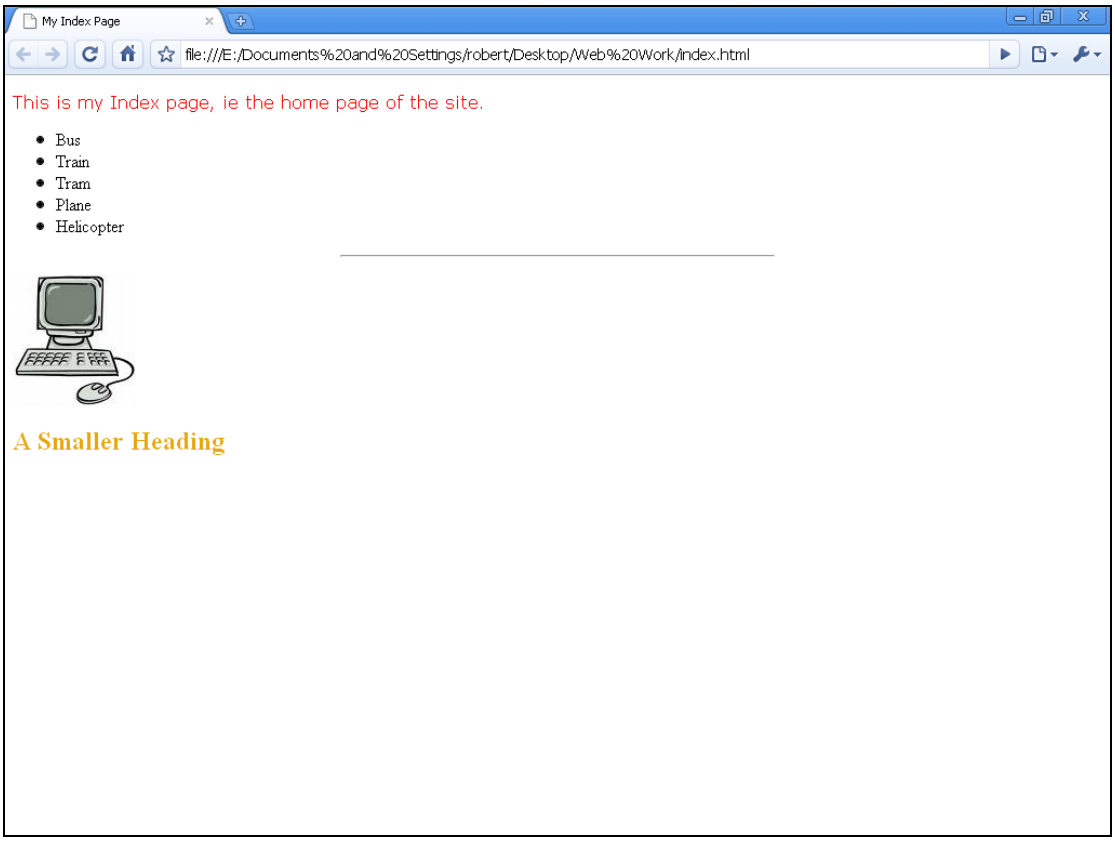


In a browser, such as Internet Explorer, Chrome or Firefox, it will look similar. Possibly not identical, because each browser shows the standard <hr> rule in a different way, but the differences will only be minor. For example, the line may be a slightly different colour or thickness.

As with every HTML tag, to stamp your own design on it you need to use CSS. For example, add this to your CSS file:

```
hr {
width: 400px;
}
```

and your horizontal lines now become 400 pixels wide, rather than filling the entire width of the screen. Like so:



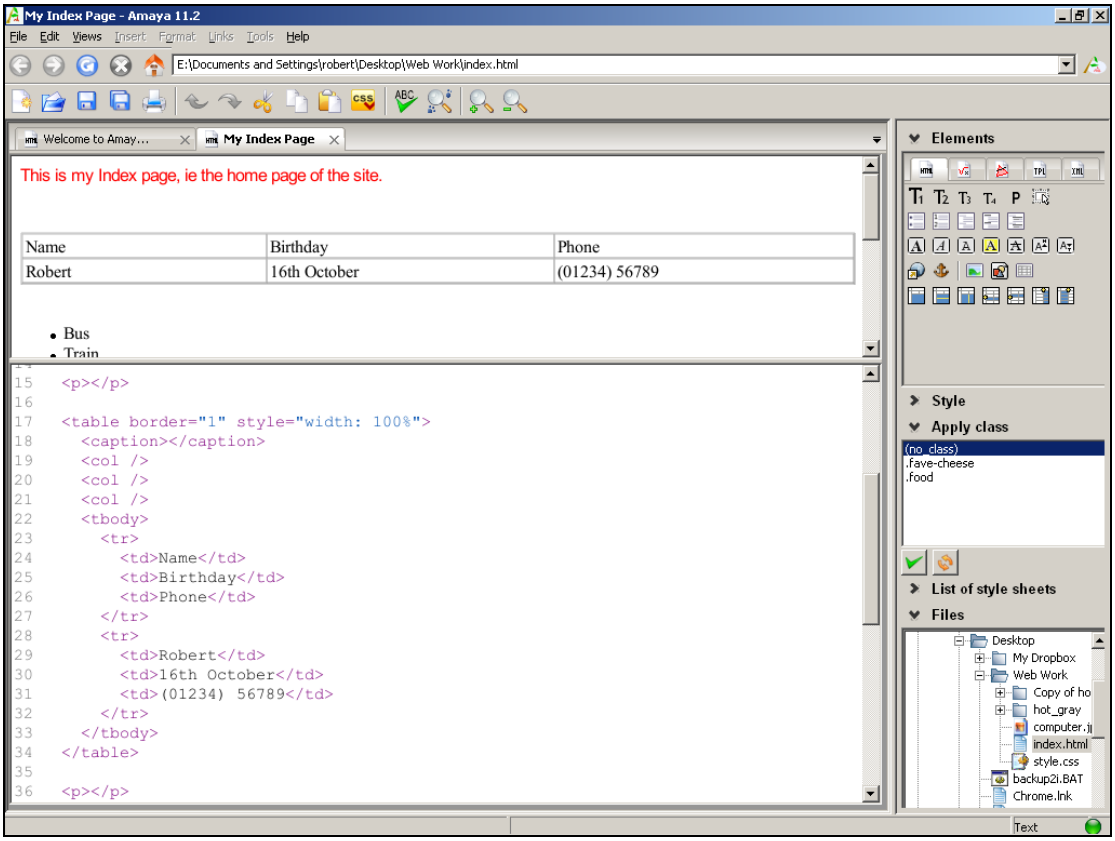
As always, if you want one particular line to be a different width, use an id-based selector. Eg, create a line like this:

```
<hr id="special_line">
```

and then create a style called **`#special_line`** which contains the specific information for this line.

Tables

A table is useful if you want to present tabular information on a web page, such as a timetable, price list and so on. To create a table on a web page using Amaya, go to the Insert menu, choose Table, then choose Insert A Table. In the box that appears, choose the number of rows and columns that you want, and the border thickness. Here's what Amaya looks like, after I inserted a table of 3 columns and 2 rows, with a border thickness of 1:



I've used Amaya to split the screen to show the table as it would normally appear in a visitor's browser, and also the HTML code that Amaya has generated. You can now see the HTML tags that are relevant to tables, which you need to know in order to be able to control the appearance of your tables with CSS.

The table starts with a <table> tag and, of course, ends with </table>. The table's caption, had I chosen to include one, would have been placed between the <caption> and </caption> tags. By applying CSS to the caption tag, you can thus choose the particular font etc that you wish to use for your table captions.

The <col> tags are used to specify various characteristics for each of the columns of the table. The tags above, as generated by Amaya, don't actually have any effect because there are no additional parameters specified. It's easiest, and perfectly safe, to ignore them.

Next comes the main body of the table, which is enclosed within a <tbody> and </tbody> tag. Within the table body, each table row goes between a <tr> and </tr> tag, and each cell is created with a <td> tag, which stands for Table Data.

To change the overall look of all tables on a page, use your CSS file to apply one or more styles to the <table> tag. This is where you change things such as overall border line colours and thicknesses, the margin and padding between and around each cell, and so on. Styling

the <tr> tag, meanwhile, lets you specify various aspects of table rows, and specifying the <td> tag in your CSS file lets you style the table cells themselves.

As always, you can use id-based styles if you want to change the appearance of just one row or cell. For example, change **`<td>Robert</td>`** to **`<td id=robert-cell>Robert</td>`** and you can now specify a unique style for the particular HTML item whose id is robert-cell. Just create an id-based style with TopStyle Lite and specify the required id, as explained on page 75.

Another useful table-related tag is <th>. This stands for Table Header, and is often used in place of <td> tags for each of the cells in the top row of a table. By applying different styles to <th> and <td>, you can easily create a different look for your header row. For example, add a background colour, make the text slightly larger, or make it bold.

Image Maps

On page 84 we saw how to add a picture, or image, to a Web page using the <img> tag. In the same chapter, we saw how images can also be used as hyperlinks, so that a visitor to your site can click on an image, just like some text, to be taken to another part of the Web.

Now that you know about images and links, we can introduce another, advanced feature of HTML which is known as an image map. This allows you to divide an image into a number of separate areas, and specify a different link destination for each area.

For example, let's say that you are creating a web site that offers various items of clothing for sale. You place a picture on your home page that shows someone dressed in shoes, trousers, t-shirt, scarf and hat. You want to allow your visitors to click on part of the picture and be taken to different pages of your web site, namely shoes.html, trousers.html, tshirts.html, scarves.html or hats.html.

You might think that achieving this feat involves loading the picture into an image editing program and dividing it into five separate files, but actually you'd be wrong. The picture can remain intact. What you need to create is an Image Map, which is a special collection of data items which specify the boundary co-ordinates of each clickable area of the image, as well as details of the link destination. For example, "if someone clicks within the square that starts in the top left-hand corner and whose lower right hand corner is 38 pixels down and 29 pixels across, take them to trousers.html".

You then include this image map data within the HTML code of your Web page, and the image can now be clicked in different areas for different results.

Unfortunately Amaya can't create image maps, but there are a few free programs on the internet that can. One such program is called Handy Image Mapper. There's a link on **http://silveragesoftware.com/handytools.html** which allows you to download it, and the good news is that, like all of the software mentioned in this book, it's free.

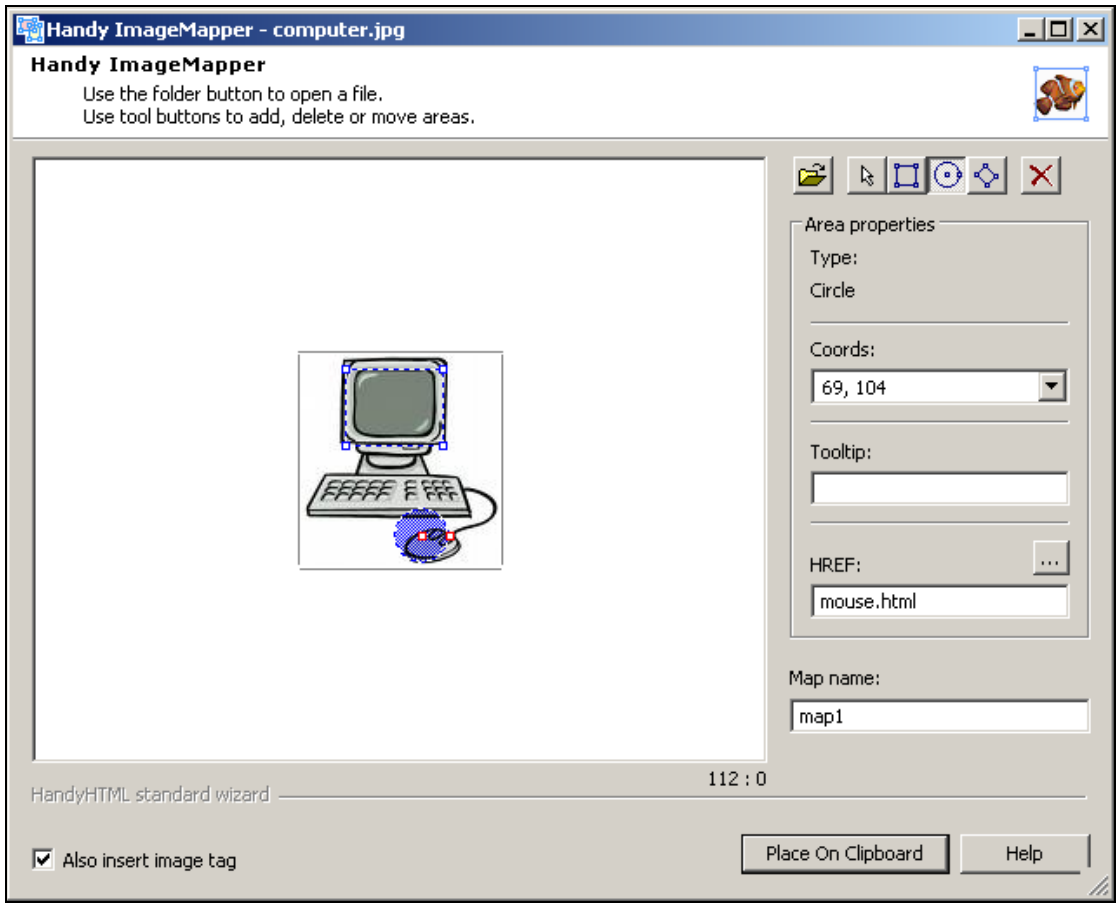
Once you've download the file and installed it, you can run it. Start by clicking on its folder icon to open one of your pictures. Then click on the shapes to define areas of the screen, and enter the URLs that correspond to the intended web destinations. When you're finished, click the Place On Clipboard button, which creates the map data and copies it to the Windows clipboard. You can then paste it into your web page.

The HTML code for using an image map looks like this:

```
<IMG SRC="computer.jpg" USEMAP="#computer">

<map name="computer">
Paste your image map data here
</map>
```

As you can see, it's all pretty straightforward. Add a USEMAP parameter to the <img> tag, which names the image map to be used. Then place the map data somewhere on the page between a <map> tag and a </map> tag. The map data can simply be pasted from the clipboard, having been generated with Handy Image Mapper as described above.

Here's how Handy Image Mapper looks during use, once you've loaded a picture:



In this example, I've loaded computer.jpg and, using the controls at the top of the right-hand column, created a square area that links to screen.html and a round area that links to mouse.html.

With this done, I can paste the map data into Amaya. The map data, in case you're interested, consists of:

```
<MAP NAME="map1">
<AREA SHAPE="RECT" COORDS="25,8,81,52" HREF="screen.html" TITLE="">
<AREA SHAPE="CIRCLE" COORDS="69,104,16" HREF="mouse.html" TITLE="">
</MAP>
```

Obviously you'll need to change the name from map1 to match whatever you have specified in the picture's <img> tag.

Password-Protecting your Web Pages

Most web sites are openly available to anyone who knows, or finds, the URL. Most of the time, that's the way you want it to be. But occasionally you will want to password-protect some or all of your pages. For example, if you run the web site for a club or a school, you'll want to restrict certain areas to members or students only. Or perhaps your site contains information that has a value, such as documents or pictures, and you need to ensure that it's only viewable to your paying customers.

Password-protecting one or more areas on your web site is actually a fairly easy task to accomplish, because there's a facility built into most web servers. You don't even need to do any programming.

The key to protecting your pages in this way is to know about two special text files called .htaccess and .htpasswd. The first file, .htaccess, contains some special commands for the web server which basically say "please don't show anyone the pages in this folder unless they enter a correct username and password. And here is where you'll find the list of usernames and passwords". As you've probably guessed, the second file, .htpasswd, contains the actual list of usernames and passwords. All that you need to do in order to protect your site, therefore, is to create these two files and then use your FTP program to upload them to the web server.

Almost all web servers support this feature. If you're not sure whether yours does, you've nothing to lose by trying. If your web browser pops up a username and password box at the appropriate point, you're in business. If it doesn't, and simply shows you the pages without asking you to log in, it's back to square one. Or you need to ask your hosting company to enable the facility, as some of them don't turn it on by default.

The remainder of this chapter shows you how to create the .htaccess and .htpasswd files. Note, by the way, the dot at the front of the filename. You're probably used to seeing dots in filenames, such as letter.doc or finances.xls, but these two files are special as there's nothing before the dot. Web servers generally won't object to filenames in this format, but some desktop computers might. If yours does, and you have trouble creating files with these weird names, just call them something a little less weird when you first create them on your PC. Then, once they're uploaded to the web server, use your FTP program to rename the uploaded versions *in situ*.

The .htaccess File

The .htaccess file is used to provide special instructions to a web server. It can be used for many purposes. For example, you can use a .htaccess file to customise the error messages

that visitors see if there's a problem with your site. If a visitor attempts to surf to a page that doesn't exist, or if they click on a link that doesn't actually lead anywhere, they'll normally see an "Error 404" page. This is generated by the visitor's web browser. But by using a .htaccess file you can create and specify a customised error page instead. Just create it as a standard web page, upload that page to the server, and then use a .htaccess file to tell the server where the page is.

The other main use for a .htaccess file is password protection. For such purposes, the file needs to look something like this:

```
AuthType Basic
AuthName "test"
AuthUserFile "/home/thewebbo/.htpasswds/.htpasswd"
require valid-user
```

The first line tells the web server that you want to use HTTP Basic Authentication, which is the official name of the password protection feature that is the subject of this chapter. The second line is where you specify the text message that appears in the password entry box that pops up on visitors' screens. You'll see it in action shortly. The final line, **`require valid-user`**, tells the web server that it should deny access to everyone except those who can provide a valid username and password.

The most important line is clearly the third one. This contains the location of the .htpasswd file, which is where the list of valid usernames and passwords is stored. It's important to get this location right, or people won't be able to access your protected pages. However, the precise format varies greatly and is generally specific to each hosting provider. You'll need to search your host's online support pages for terms such as "htpasswd path" in order to find out the details.

In the example above, I've used the version that is correct for Hostmonster, the company that's hosting my **`the-web-book.com`** site. This starts with /home, then comes my username, then a folder called .htpasswds and then the file name itself (.htpasswd).

It doesn't normally matter where you put the .htpasswd file, so long as you use the .htaccess file to tell the server where you put it. But in the case of our hosting company, hostmonster, they impose a rule that all such files have to go in the .htpasswds folder. If you look at the hierarchical tree of folders that FileZilla displays, you'll see that .htpasswds is not within the public_html folder. Although you can still access the .htpasswds folder with your FTP program, as owner of the site, visitors can't access it with their web browser.

Only files that are within public_html can be accessed with a web browser. Technically, public_html is known as the web root folder. It means that you can gain added security by

ensuring that files which visitors don't need to surf to are not put in public_html. And for obvious reasons, you really don't want visitors to try surfing to, and attempting to download, your .htpasswds file. Hence Hostmonster's insistence that you don't put it in public_html. If you do, the feature won't work.

To start password-protecting your web site, use FileZilla to create a new folder on the site, within public_html, called private. In that folder, create and upload a simple test page. Then type the example .htaccess file shown above, into a text file, call it .htaccess, and upload that into your private folder too.

The .htpasswd File

We've created a private folder, placed a sample page in there, and used a .htaccess file to tell the web server where the list of usernames and passwords is. All that remains is to actually create that list of usernames and passwords. In other words, the .htpasswd file.

A .htpasswd file is merely a text file of usernames and passwords, with each pair on a separate line and with a colon between the usernames and passwords. Here's an example (but don't bother trying it yet, for reasons that will be explained shortly):

```
fred:flintstone
open:sesame
```

In this case, it would create two valid users (fred and open) with passwords of flintsone and sesame respectively. However, it's not *quite* as simple as that!

If you were allowed to create the file as shown above, it would represent a significant security risk. Anyone who managed to hack into your web site and download the file would now have a list of every username and password on the system. Clearly not ideal, or remotely sensible.

So for added security, here's how the feature actually works. Instead of creating a line that consists of, say, **`open:sesame`**, you encrypt the password in advance, thus the line becomes:

```
open:$apr1$kZy1b/..$BM8fLekWkKblQ6pynXeM/1
```

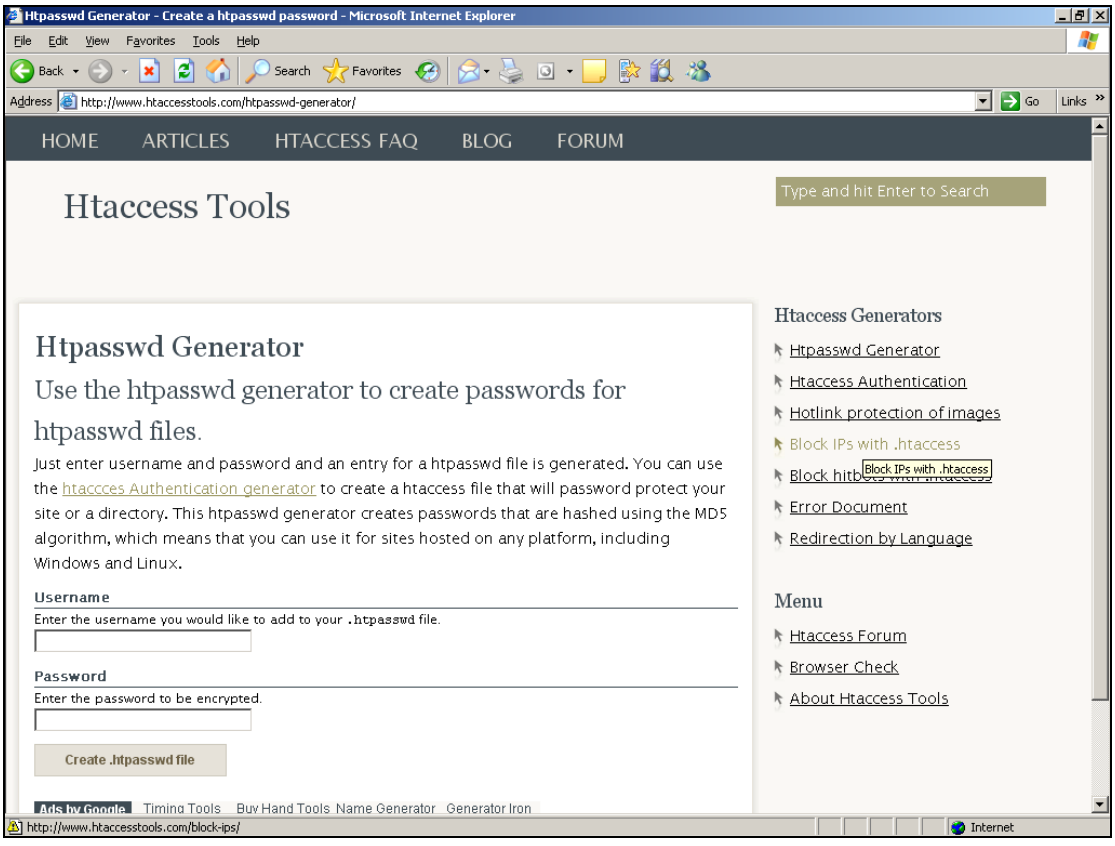
When the user types in their password, the web server encrypts it and compares it to the version in the .htpasswd file. If both of the encrypted passwords match, then the original, unencrypted ones must also match too (don't worry, they just do, trust me).

The encryption algorithm (formula) used for encrypting .htaccess passwords is a standard feature that almost every web server supports. It works because the encryption is a one-way

process. Turning sesame into $apr1$kZy1b/..$BM8fLekWkKblQ6pynXeM/1 is very easy. Trying to reverse the process, however, is much more difficult. Not impossible, but certainly difficult enough to provide another level of security for your protected pages.

All that remains, therefore, before you can create a proper .htaccess file, is for you to find a way of encrypting passwords in the correct way, ie the way that web servers expect. There are lots of programs that are capable of doing it, which you can download from the internet for free. Even easier, there are websites that will do the encryption for you. They're quite safe to use, as they won't know what web site you're creating the passwords for.

A Google search for "htpasswd generator" will lead you to a choice of sites, which will look something like this one:



Type in the username and password of your choice, and the site will generate the line that needs to be added to your .htpasswd file. Then just copy and paste it in, using a simple text editor like Notepad or WordPad.

When you're done, use FileZilla to upload .htpasswd to the .htpasswds folder of your web site (or wherever else you decide to put it, according to what your hosting company allows). Then open your web browser and surf to your protected folder. In my case, that's **`www.the-web-book.com/private`**. As if by magic, this appears in my web browser:

Notice the "test" message, as specified in the .htpasswd file. We could (and probably should have) just as easily specified a more friendly message, such as "Please log into the site".

Keep typing wrong usernames or passwords, and nothing happens. The password entry box keeps re-appearing. But as soon as you supply one of the correct username/password pairs, the protected page is displayed. Once the user is "logged in" like this, access to all pages within the protected folder on the server (ie, the one which contains the .htaccess file) will be allowed without the user needing to re-enter their password.

Protecting Multiple Folders

If you want to protect more than one folder on your site, you can do that in 2 different ways. If you want to use the same set of usernames and passwords for all of the protected folders, just put a copy of the same .htaccess file in each of the folders, which all point to the same .htpasswd file. If you want to use different usernames and passwords, then each protected folder will need its own .htaccess file, which points to a unique .htaccess file. In the case of Hostmonster, those files must all reside in the .htpasswds folder, but the name of the file itself can vary. So you could, for example, have separate files called .htpasswd1, .htpasswd2 and so on, pointed to by complementary .htaccess files.

CMSes and Other Software

By now, assuming you've read every page of this book, you'll know pretty much everything that's required to create simple Web pages from scratch. This includes registering a domain name, renting some space on a web server, and creating pages using HTML for the content and CSS for the layout.

With this basic knowledge under your belt, it's time to make an important decision. Will you continue to create web sites in this way, where each page is an HTML file that has to be edited and uploaded? Or will you do things The Other Way?

To understand what we mean by The Other Way, consider modern web sites such as Facebook, WordPress, LinkedIn, Bebo, Twitter, MySpace and Ebay. These are often referred to as Web 2.0 sites, but what does this mean exactly?

When you want to update your profile on Facebook, you don't have to create an HTML file with Amaya and then use an FTP program to upload it. You just fire up your web browser, connect to the site, log in with a username and password, and manage your content directly within the site itself. It's the same whether you're posting a new page to your blog on wordpress.com, creating an item for sale on eBay, or using any of the other sites I mentioned. How do they do that? And more importantly, can anyone do it? Can you create sites that work in a similar way, that you can update directly from a browser without the need for an HTML editor or an FTP tool?

You may be surprised to know that creating a Web 2.0 site isn't actually difficult. Mainly because the software that allows you to do it is available from various places. All you need to do is obtain it, install it on your hosting space, and start using your Web 2.0 site. Even better, many of the most popular programs are what's known as FOSS (free, open-source software). This means that, not only are they completely free of charge, you also get the program source code so you can modify it if it doesn't quite fit your needs.

The key to creating a Web 2.0 site, ie one that you can edit and maintain from within a web browser without the need for programs like Amaya and FileZilla, are two technologies known as server-side programming languages and server-side database engines. The former lets you write programs which run on the Web server. One of the things that these programs can do is to retrieve text from a database and present it in a web browser, which is where the second technology comes in.

We'll learn how to use these technologies in much more detail during the forthcoming chapters in this book. There, we'll learn how to write programs using the PHP language, and access databases that use the MySQL database engine. This will allow us to create our own

web-based applications, including those that work in the same way as Facebook and all the other Web 2.0 sites mentioned above.

But if you want to create a Web 2.0 site that performs a common function, such as a blog, or a discussion forum, or a simple web site, or a picture gallery, then here's the good news. There are lots of ready-made open source applications. Just install one or more of them on your hosting space and you're ready to roll.

A Web 2.0 application that performs the basic functions of creating and maintaining a standard web site is known as a Content Management System, or CMS. Sometimes the term used is WCMS, or Web Content Management System. As I said, we'll leave the precise details, including how to write one using PHP and MySQL, until later. For now, let me show you just how useful the existing, free, ready-made applications are, and how easy they are to get up and running.

Note that this chapter isn't an in-depth tutorial on how to use the products mentioned. If you decide that any or all of them would be a useful addition to your site, use what follows as a starting point and then search online for documents and forums that help you get your system working (and properly secured) the way you want.

For the purposes of this chapter we're going to install and configure 4 different products:

- Joomla - One of the leading free WCMS products
- Wordpress - Another leading CMS, especially good for blog sites
- phpBB - A discussion forum
- Plogger - A picture gallery

There are, of course, many other free, open source web applications in addition to those detailed here. There are literally hundreds available, in many other categories. If you want to sell things via your web site, for example, there's a free e-commerce suite called oscommerce. If you want to set up your own version of wikipedia, there are dozens of free wiki applications available. One great place to find Web software is **`www.hotscripts.com`**, which offers ready-made applications in a number of categories. Here's just one part of one of their menu screens, showing the huge number of programs available, all for free.

CMSes and Templates

CMS products like Joomla and WordPress are template-based. You start with a basic template, or page layout, which contains common elements such as a logo, the site's title, the page footer and so on. The template also contains markers (called placeholders) to indicate where the unique content for each page should be inserted. For example, the main heading, the intro paragraph, the main body text, the name of the author, and so on.

Key to using a CMS is to create a template. Thankfully, there are lots of free ones available, which can change the look of your site instantly. But if you don't like any of the standard templates, you can edit the existing ones or design your own from scratch. Any HTML page can be turned into a CMS template – it's just a case of looking through the CMS documentation for details on how to insert those special markers at the point where you want each page's content to appear.

If you like designing HTML pages, perhaps to be used as CMS templates, one product that deserves a mention is Xara Web Designer. It's not free, but is only around £39 or $60. It contains more than 700 ready-made page templates, easily adaptable for use with any CMS,

and it's incredibly easy to use. It allows you to do things that no other HTML editor does. For example, here's one of the program's standard designs:

Now, just type over any of the text or buttons in order to customise the page. Or click one of the coloured diamonds to change the entire colour scheme. You can also click on any object (a paragraph, a picture, a button etc) and, with your mouse, move it, enlarge it, or rotate it, just like any PC-based drawing program. When you're done, just save the file and you've got yourself an HTML page which can be used as-is on a site or converted to a CMS template.

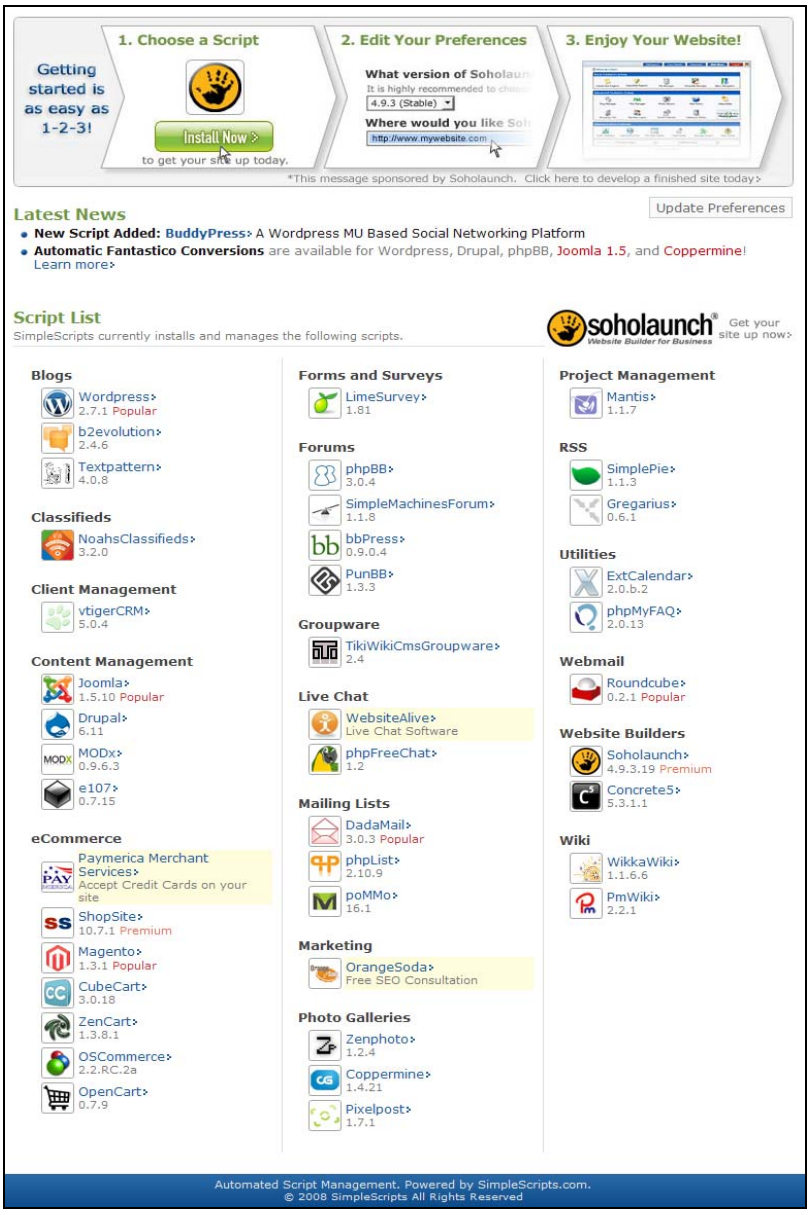
Also, if you're using Joomla or Wordpress, check out a commercial product called Artisteer, which makes it easy to create great-looking template files. You can read more, and see a demo video, at **`www.artisteer.com`**.

Automatic Installers

In the chapters that follow, I'll show you how to obtain, install and configure Joomla, Wordpress, phpBB, and Plogger. But before you read these sections in detail, check whether your web host offers you the use of an automatic installer such as Fantastico or SimpleScripts. If so, then your life just got even easier. These systems let you install many of the leading free web applications in seconds, with just a single click.

If the facility is available, and if you want to go down the easy route rather than learning how to install and configure the programs from scratch, then it makes good sense to use it. Just check on your web hosting control panel to see if the feature is available.

As an example, here's the main SimpleScripts menu, which shows all of the applications that it can install. You can then install any of them into your hosting space with just one click, and be up and running in just seconds:



Try Before You Install

Although it's easy to install applications into your hosting space, and even more so if you use SimpleScripts or Fantastico, the process still takes time. Plus, if you don't like the application, you have to spend more time uninstalling it. The best way round this is to find a site that already has a copy of the application installed, and which allows you to log into it and use it for test purposes. Then, when you know it's what you want, you can set up your own copy in your own hosting space.

My favourite online resource for testing open source Web applications in this way is **`www.opensourcecms.com`**. Despite the name, it doesn't just cover CMSes. It also includes forums, wikis, and more. The site is completely free to use, and offers links to installed demo copies of dozens of products. You even get supplied with the admin passwords, so you can experience the installed product as an owner rather than just a user. Which means, for example, that you can create new pages on the CMS rather than just reading what's already there.

For obvious security reasons, a few features are automatically disabled, such as sending email or changing the admin password. Also, the entire system is automatically deleted and reinstalled every couple of hours so any malicious content won't be around for long. But opensourcecms.com should definitely be your first port of call if you're considering installing such a product into your hosting space.

If the product you want to install isn't listed on opensourcecms.com, all is not lost. Check out the product's home site on the Web. Many offer a test version that you can log into (look for a Demo or Online Demo link), and this often includes the admin passwords so you can check out the management facilities too.

A Word about Patching

The key to security on any Windows PC is to ensure that automatic updates are enabled. Microsoft issues around 100 security fixes each year for Windows, many of which are for bugs that are so serious they could allow everyone in the world to access your files via the internet. By ensuring that these patches are downloaded and installed automatically, you prevent your PC from serious attack.

But Windows is not unique. Every program has security bugs, and every team of developers issues regular security patches for its products. The people who develop products such as Joomla, phpBB, and all of the other web-based software mentioned here, are no exception. The difference between Windows and these products, however, is that there's no Automatic Updates feature. Therefore, if you install a product on your Web server, it's vitally important that you check the product's web site every couple of months to find out whether any important security updates have been issued. If they have, you'll need to download them and

install them on your server. Instructions on how to do this are always available, and normally amount to nothing more than uploading the files to your server via FTP to overwrite the versions already there.

Failure to carry out this essential maintenance task will mean that your entire site is potentially liable to attack by hackers. In some cases, this could mean that hackers can wipe or corrupt the content database which holds all of your CMS web pages, or all of your forum messages.

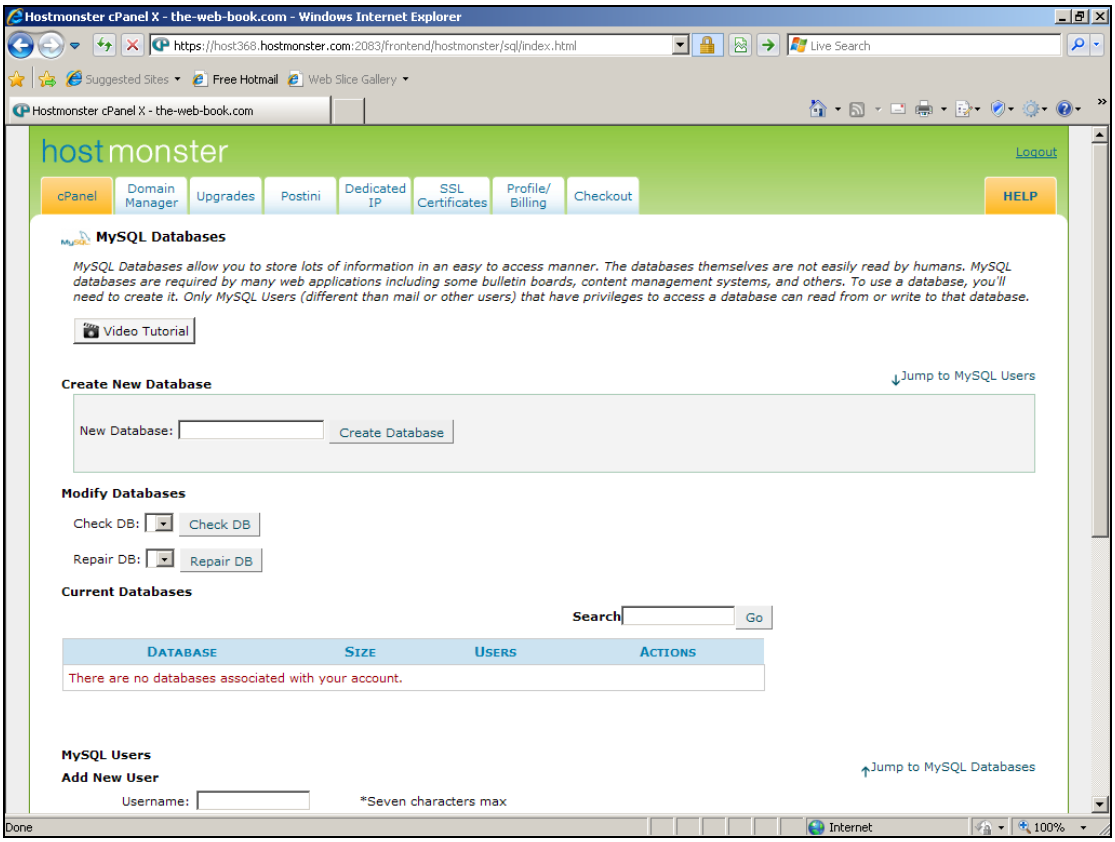
Setting Up A Database

All of the products that we're about to install require a database on the server in order to store content such as web pages, forum postings, and information about images. When you come to install and configure the products, you will be asked to provide details of a database that can be used. Therefore, if you intend to install any of the products, you MUST first create a new database on your web server.

There are various web database technologies, but one of the most popular is MySQL. In a later chapter I'll show you how to create your own programs that can store information in server-based databases. But for the moment, all we need to do is to create an empty database for our products to use.

To start, log into the control panel of your web host. In our case it's the Hostmonster control panel, a picture of which appears on page 30. Remember, this is your hosting control panel and not your domain name control panel.

Scroll down to the Databases section and click on MySQL Databases, and you'll see a screen like this. Of course, if you're not using Hostmonster, or if your host doesn't use the cPanel control panel, your site won't work in quite the same way as this, but the principle is always the same.



All that's required is to type a name for our new database, and click the Create Database button.

Rather than creating a database called joomla, plogger, or anything else that's specific to one particular product, we're going to call it **`cms`** instead. We can then use this database not just with Joomla, but with all our other CMS, forum and gallery products too (WordPress and so on). Don't worry – the data won't get mixed up, for reasons that will become apparent later. But by having just one database for all of our CMS testing work, we gain a couple of useful benefits:

1. If your host only allows you a small number of databases, this is a great way around the limitation.
2. When you're finished testing CMSes, there's only one database that you need to delete

So, go ahead and create the cms database. cPanel does the job, and displays a confirmation:

Notice how the database that's been created is not actually the name we chose? We chose cms but we've actually got one called thewebbo_cms instead. Why? Because, like many hosting companies, Hostmonster adds your username or some other unique identifier to each database you create. Don't worry about this, as it won't be visible to your visitors. But it's very important to know about this, because it will affect what you need to tell the product when it asks for details of the database that you have set up for it to store its information in.

Here's a useful tip. Whenever you need to connect to a MySQL database from your own PHP programs, or when you're configuring third party products such as Joomla, you'll need to know the database host name, the database name, and the connection username and password. So once you know these, and they work correctly, make a note of them somewhere. You'll need them often so it's handy to have them in a convenient place. But make sure you keep them secure – anyone who finds them could connect to your database via the web from anywhere in the world and delete or corrupt any web site that stores its data there.

The database connection settings that will work for us in the following examples are:

Host: localhost
Database: thewebbo_cms
Username: thewebbo
Password: xxxxx

Setting the host name to "localhost" tells the program that the database server and the web server are the same machine. In most cases, that's the way it works. If your hosting company's database server uses a different name, you'll be informed.

General Installation Procedures

Although there are minor differences, the basic procedure for installing an open source product on your web server is as follows:

1. Check that your server supports the main features (PHP and MySQL, for example) that the product requires.

2. Download the product from a web site, normally as a .zip file
3. Check any README file which will contain basic details of the installation procedure and anything else that it's important for you to know.
4. Extract the zip file to your computer
5. Create a folder (directory) on the web server
6. Upload the product's files to the new directory
7. Ensure that you have a MySQL database set up on the server for the product to use
8. Surf to the product's installation page, using a web browser, and fill in the details of the MySQL database. Also choose an admin username and password.
9. Test the product by surfing to its main home page on your server.
10. Occasionally refer back to the web site from where you downloaded the product, to check for important updates and security patches that you might need to install.

Once you understand this basic procedure, you should have little trouble installing any open source product such as a CMS, blog, image gallery, forum, wiki, and so on.

Uninstalling

If you no longer need access to an open source product that you've installed on your web server, uninstalling it is rarely difficult. However, the process is almost always a manual one – there's no "Add/Remove Programs" button on web servers, as there is in Windows. At least, not unless you installed the product using an automated installer such as Fantastico or SimpleScripts.

Thankfully, uninstalling a product such as a CMS, forum, gallery etc is normally just a case of doing the following:

1. Check the product's documentation or official website to find out whether there are any special actions you need to take in addition to the steps below.
2. Use your web hosting control panel to delete the database that the product uses. If you wish, export the contents of the tables to a backup file first.
3. Delete the product's directory on the server.

In the majority of cases, that's all you need to do.

Joomla

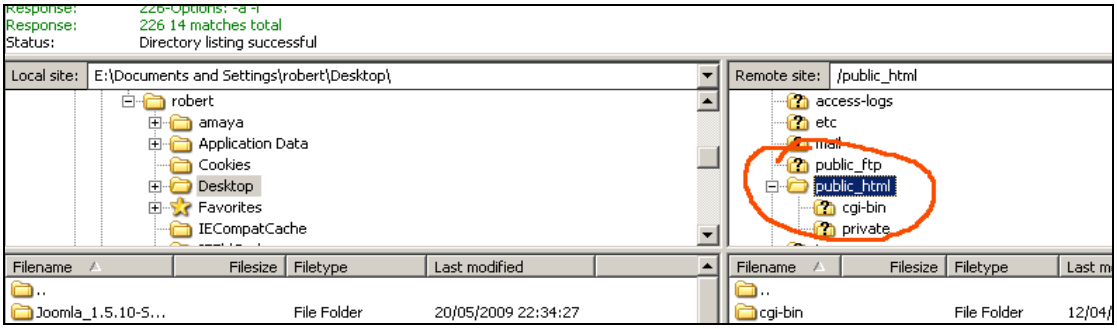
Joomla (officially known as Joomla!, complete with exclamation mark) is the best-known and most widely used of the open source CMS products. Its widespread adoption means that the Web is awash with helpful documents, free add-ons, discussion forums and other resources to help you make the most of it. For now, we'll just create a basic installation so that you can appreciate the steps involved and decide whether your site should use a CMS rather than consisting of separately-crafted HTML pages.

The first step is to download the Joomla software from **`www.joomla.com`**. This example uses version 1.5.10, but a later one may be available by the time you read this. Make sure you download the latest available version. You'll need the full installer rather than any upgrader that's offered.

You'll end up with a zip file, which you need to unpack. If you use WinZip or similar, follow your normal procedure. If you don't have such a program installed, Windows can unpack Zip files for you. Just right-click the zip file and choose Extract All, to unpack the file into a separate folder. You can then delete the zip file itself.

Uploading the Files

You now need to upload the entire unpacked Joomla folder to the web server, using FileZilla. Start the program, and connect to your server. In the folder list for the remote site, double-click the public_html folder to open it:



In the same area, right-click public_html and choose Create Directory. Name it /public_html/joomla and press OK. You should now have a folder (directory) within public_html called joomla. Double-click on joomla to open the folder, which will be empty.

Note that we created a folder called joomla rather than Joomla. It's accepted as best practice, whenever you create files or folders on a web site, to stick to lower case (small letters rather than capitals) throughout. This is because, to a web server, capital and small

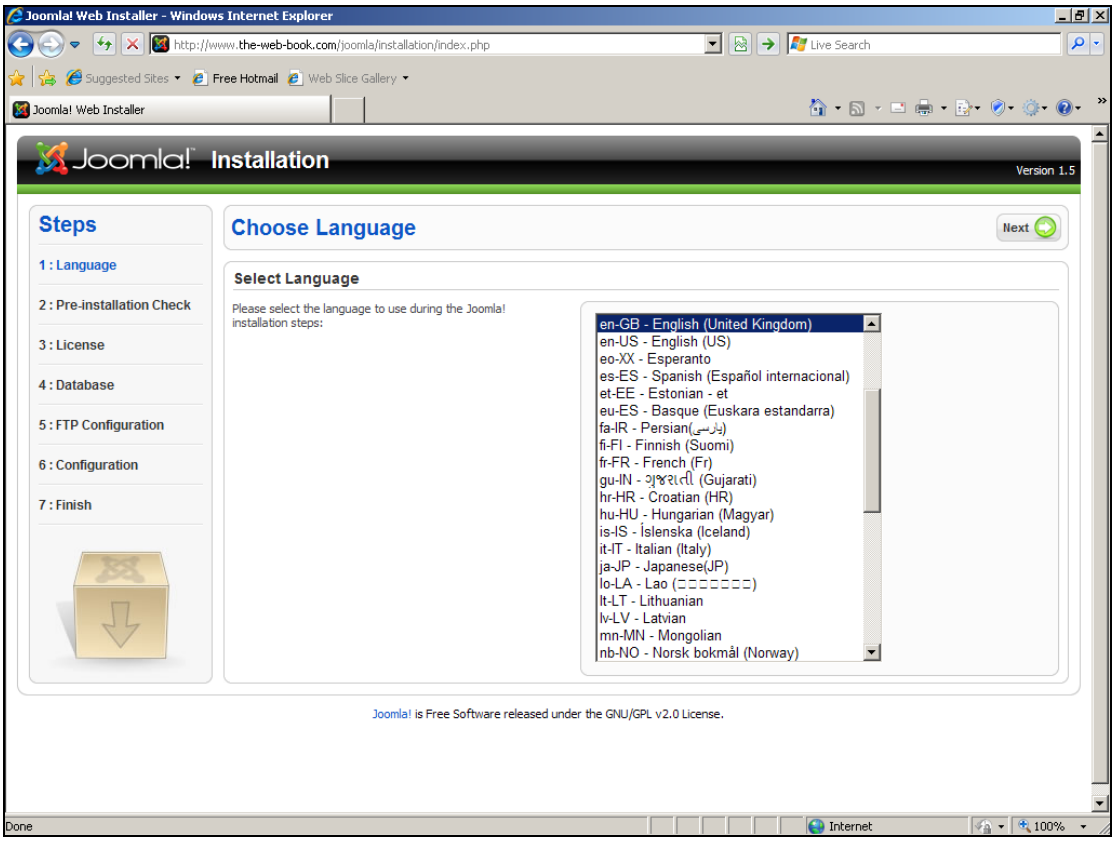
letters are different. Thus a folder called Joomla is not the same as one called joomla, which could easily lead to errors if visitors were to type the wrong thing. So to keep things simple, we stick to lower case throughout.

With your joomla folder created and open on screen, we can now upload the system files into it. On the left hand size of FileZilla's window, in the local site section, navigate to the Joomla folder that you unpacked earlier, so that all of the Joomla constituent folders and files (administrator, cache, components etc) are displayed.

Click inside the window where they're displayed, then press Ctrl-A to select everything. Then press Return, and everything should be transferred from your computer to the server. This will take a good few minutes, as there are around 4000 separate files to transfer. When it's done, you can delete all the files from your PC as they're no longer required.

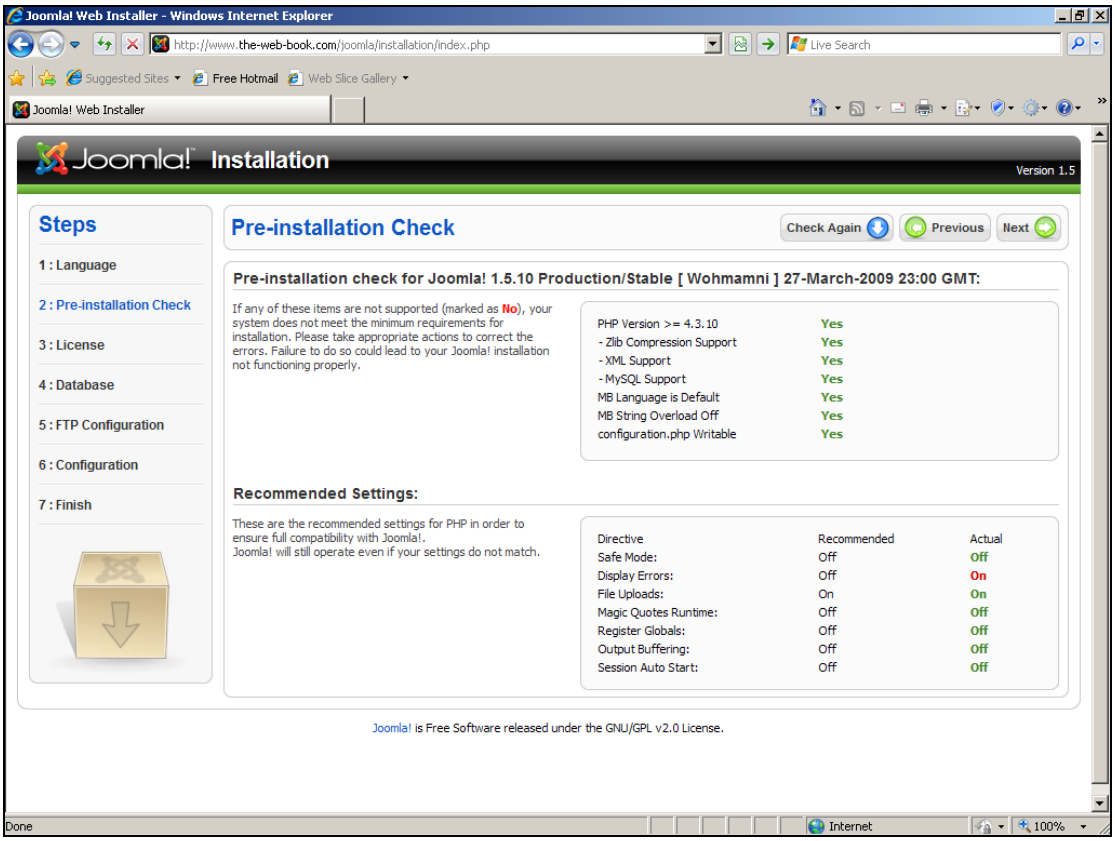
Configuring Joomla

We can now go ahead and configure Joomla. Using your web browser, surf to the Joomla installation that you copied to the server earlier. Assuming you uploaded it to a folder called joomla, the address will be **`www.yoursite.com/joomla`** or something similar. In my case, it's **`www.the-web-book.com/joomla`**, which brings up a page like this:



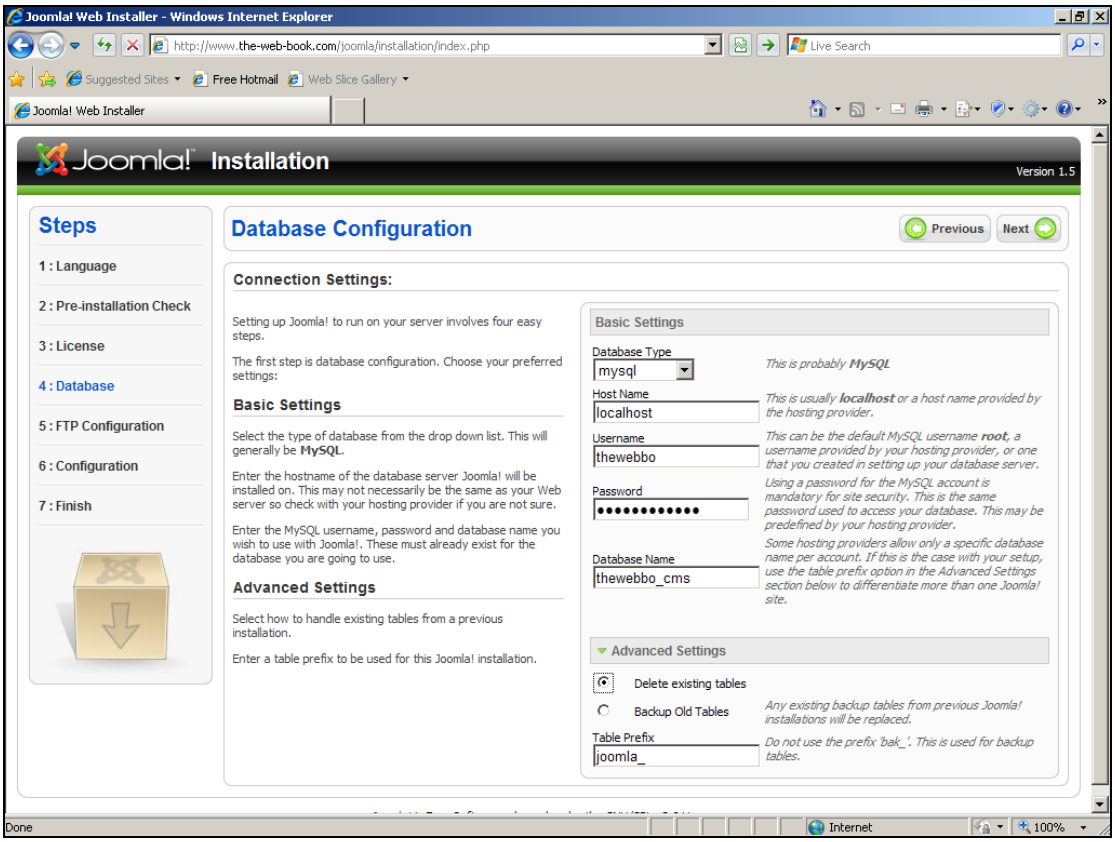
Choose a language (English, presumably), and press the Next button, to begin the pre-installation check. Here, Joomla checks various things on your server, to ensure that you have the right versions of PHP (the programming language in which Joomla is written) and MySQL. It also makes a number of other checks. If any of these fails, check the support pages on your web host's site in order to find out how to resolve the problem.

Here's the page from my Joomla pre-installation check:



The only warning is that the setting for Display Errors is turned on, where Joomla recommends that it's turned off. This is relatively minor. It means that, if Joomla crashes, someone who's visiting the site at the time might see an error message. This is unprofessional, and a minor security risk. We can live with this, so we'll click the Next button and move on to the licence agreement screen. Read it, then click Next.

Now comes the most important screen, which is to do with our database configuration. Fill it in as follows:



In the Database Type box, choose MySQL.

For the host name, type localhost. This tells Joomla that the database server is the same machine as the web server. This is not always the case – it depends on your web hosting company. If you have problems making Joomla work, check your host's support site, or the email that was sent when you signed up, to find out the address of your MySQL server.

For the username and password, use the same ones that you use to connect to your hosting control panel.

For the database name, use the one that you created on page 124 (thewebbo_cms in this case).

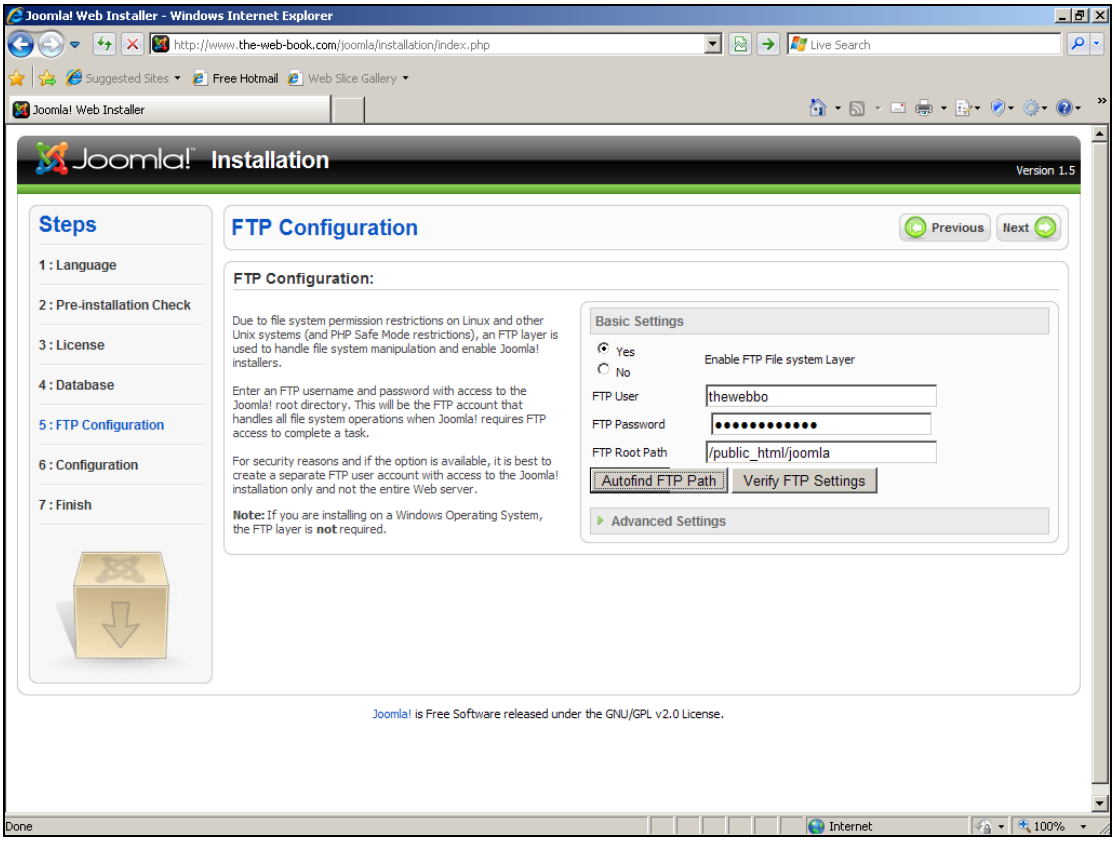
Finally, go to the Advanced Settings box. Tick the "delete existing tables" entry, just to be neat.

Choose joomla_ for the table prefix. Within a MySQL database there can be more than one table, just as there can be multiple Excel spreadsheets within one workbook. Joomla creates various tables for itself, to store sets of data such as web pages, user details, page counters and so on. By specifying a table prefix, all of the Joomla-related tables within the database will be clearly identifiable because they have joomla_ at the start of their name. When we

come on to install other products later, we'll use a separate table prefix such as wordpress_ or plogger_. This is the way that we can allow multiple programs to share one database – they all have their own tables with their own prefixes. So even if both Joomla and Wordpress wanted to create a table called users, they'd actually end up being called joomla_users and wordpress_users, thus avoiding any conflicts.

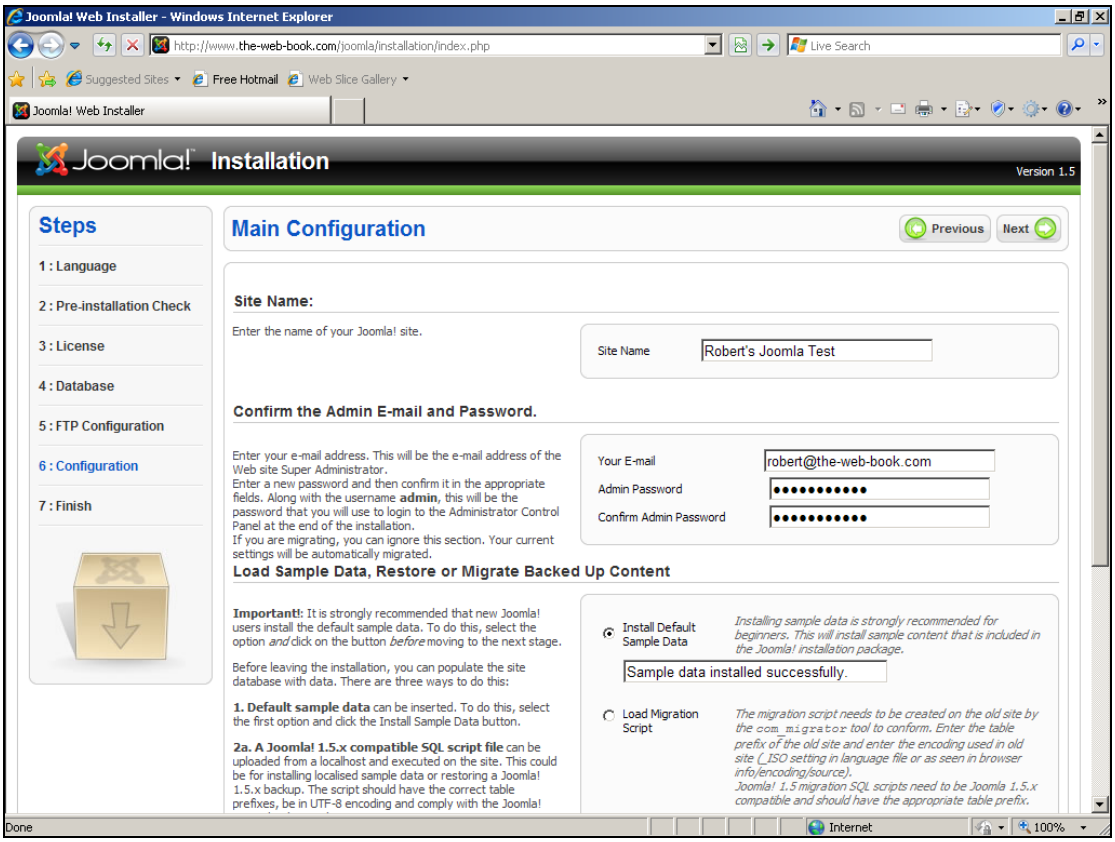
With everything entered, press the Next button.

Joomla now needs to know a username and password that it can use to communicate with your web server via FTP. For the sake of simplicity, we'll use our site-wide account that we use for accessing the control panel and the MySQL database:



Tick the box to enable the FTP layer, then enter your site username and password again. Then, rather than filling on the FTP Root Path entry, just click the Autofind FTP Path button and Joomla will do the hard work for you. In this case the result is /public_html/joomla, which is what we'd expect.

Almost there. Just one more screen to go. Click the Next button, and this appears:

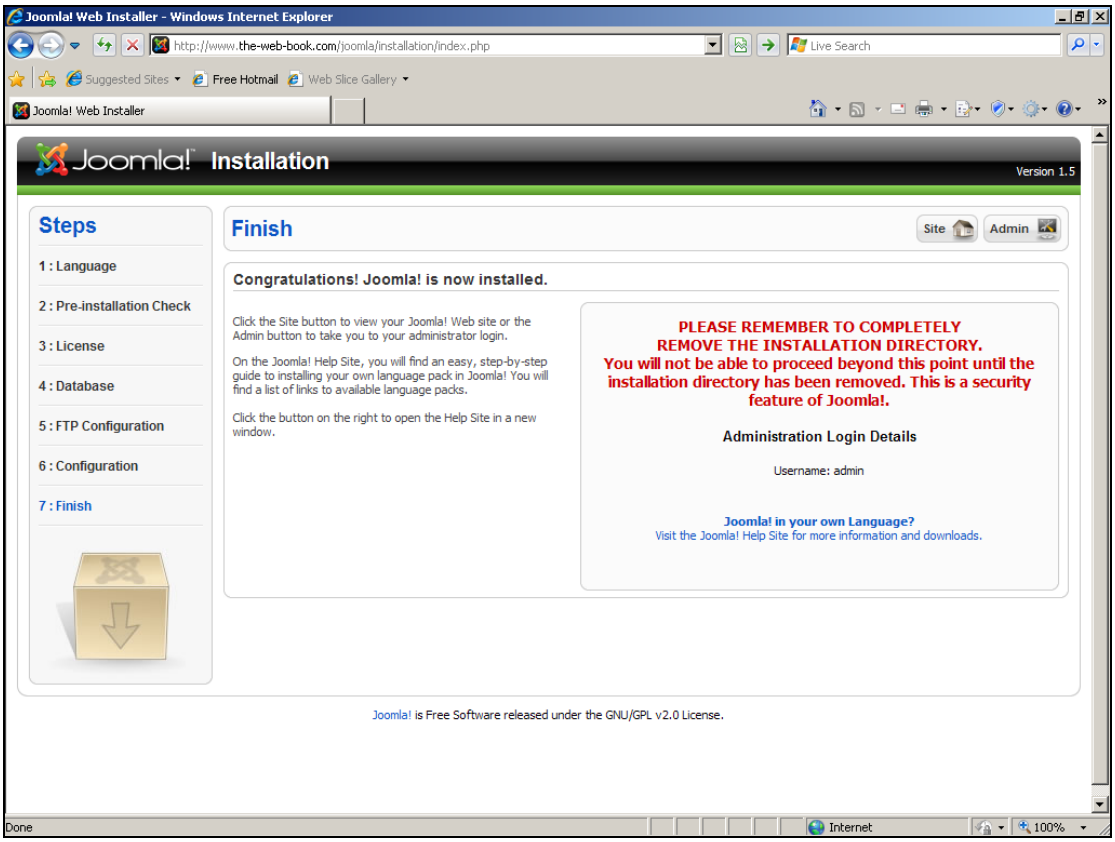


Choose a name for your web site. It doesn't matter what you choose, as you can always change it later. For now, I'll call my site Robert's Joomla Test.

Enter your email address where asked, and then choose an admin password for your Joomla site. The username will be admin (you can't change this). Make sure you remember this, as you'll need it to log into your Joomla CMS to maintain or create pages. In this demo site I've chosen a password of poltergeist.

Before you go any further, click the Install Sample Data button. This will create some sample content in your CMS, rather than leaving you with an empty CMS that contains no pages at all. Having done so, and with the "Sample data installed successfully" message displayed, click the Next button.

Hopefully, you should now have a working Joomla installation. The final screen of the configuration section looks like this:

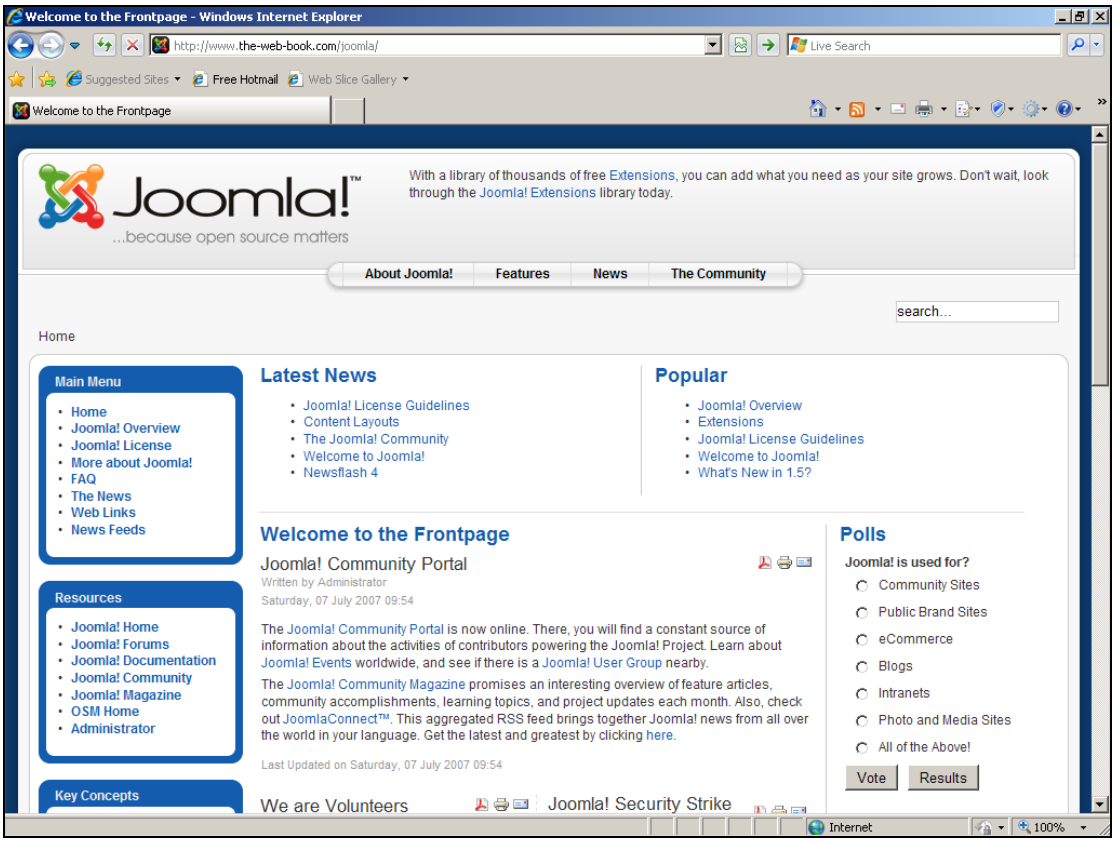


Before you can use Joomla, you need to heed the message and delete the installation folder from your web server. This is a security precaution. If you don't do this, someone could surf to that folder and re-install Joomla from scratch, just like we have done. By choosing their own admin password, they could also lock you out of your own CMS.

So, close your Web browser and open FileZilla. Connect to your server. On the right hand side of the screen, which shows the remote site, double-click on the public_html folder to open it. Then open the joomla folder. Then right-click on the installation folder and choose Delete. Wait for a minute or two until FileZilla has finished deleting all of the files from the installation folder on the server, then close the program and return to your desktop.

Your New Joomla Site

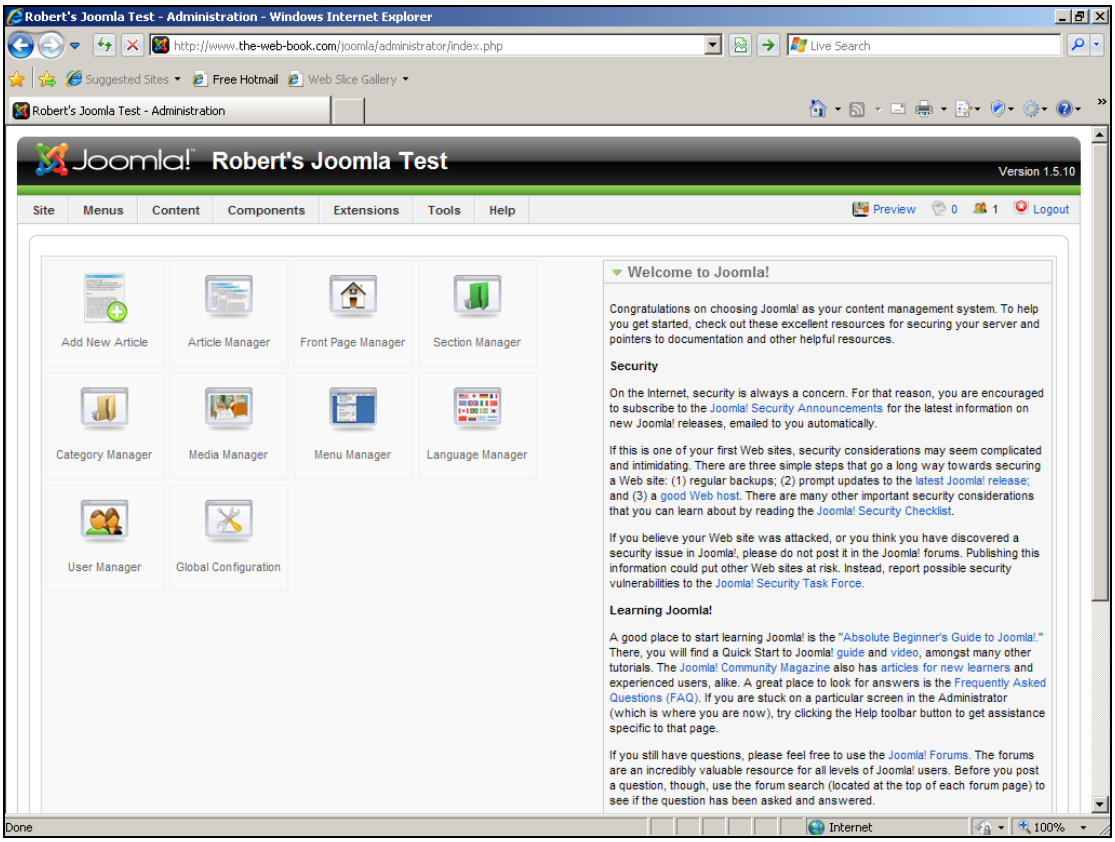
Right, hopefully our Joomla installation is now ready to use. Surf to the folder where you installed it (eg, **`www.the-web-book.com/joomla`** in our case) and you should see it in action:



You now have a fully-working, functional web site based around the best-known open source CMS. Click some of the links, and see all of the features that your new site has. Remember that you can change all aspects of the site's design and features. You don't have to stick with the colours and design that are shown. Equally, if you want to remove certain features such as the poll, you can do so. And of course, the Joomla logo can easily be replaced with your own.

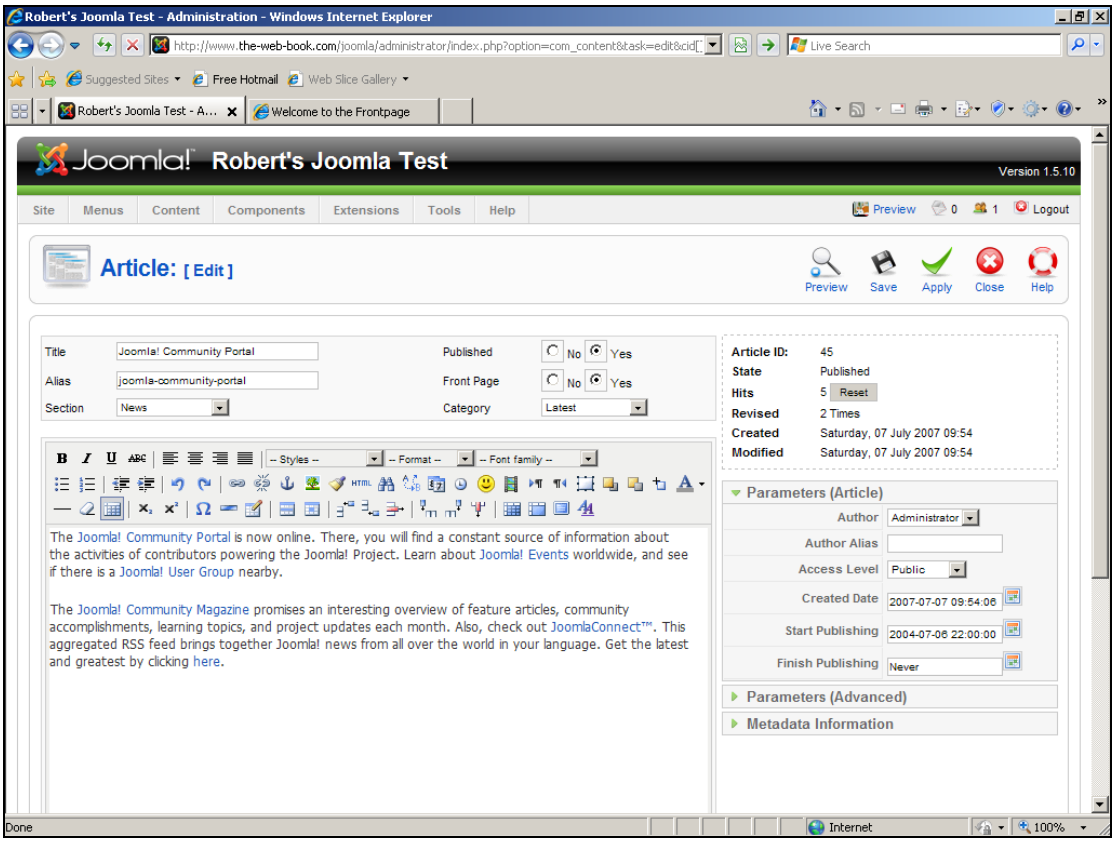
This chapter does not pretend to be an exhaustive guide to creating smart-looking web sites with Joomla. There are plenty of online tutorials that can help you do this, as well as lots of templates and skins that you can download in order to change the visual appearance of your site. For now, let's finish this brief look at Joomla by logging into the administration system, so you can see how easy it is to create content via a web browser without the need for an HTML editor or an FTP program.

On the Resources menu on the left hand side of your screen, click the Administrator link. You'll be asked for your username (admin, if you remember) and password (poltergeist for our example here). The main admin menu looks like this:

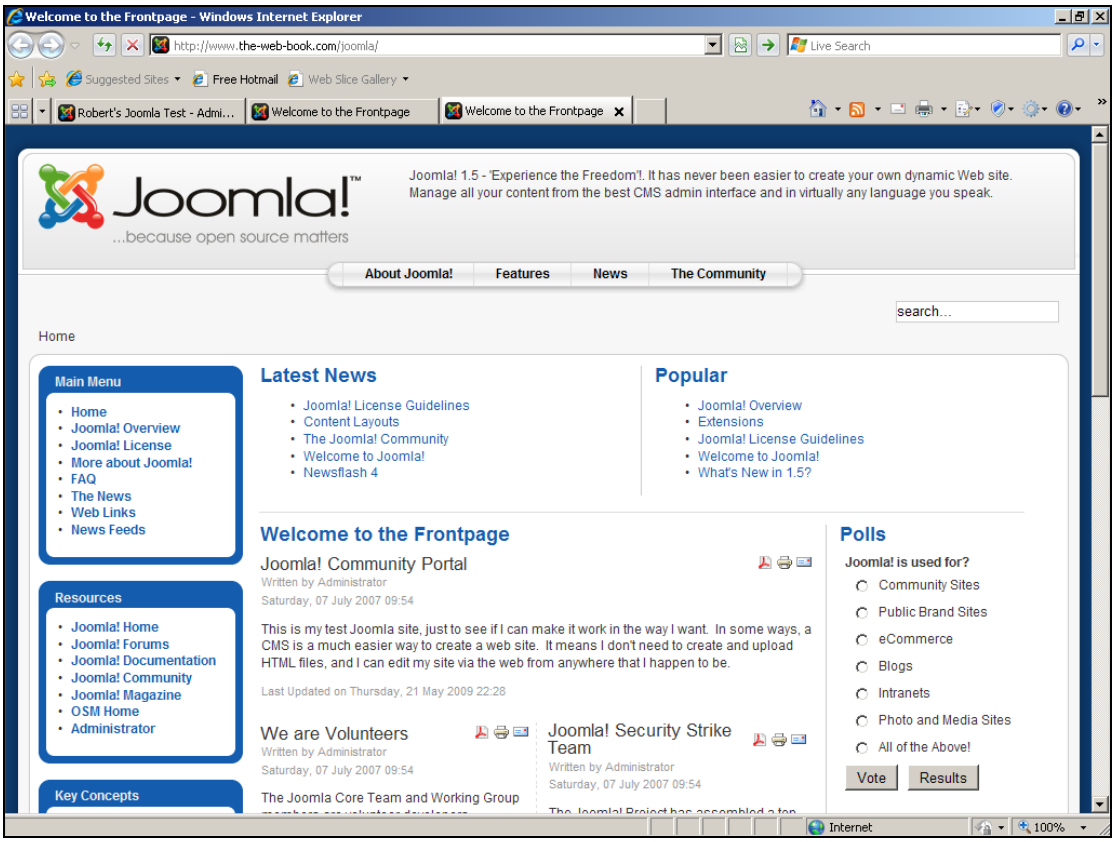


This is only a test installation, so why not just click on various features and see what they do? You won't break anything. If you do, and everything goes horribly wrong, just use FileZilla to connect to the server, delete the entire Joomla folder, then upload the files again and start afresh with a new installation. So long as you click the option to delete any database tables that already exist, you won't even have to create a new database.

Let's change something in the site. Click the Front Page Manager icon, and you'll see a list of all the items that appear on the front page of the site. Click the title of the first article (Joomla community portal) and an editing screen appears:



Type some new text in place of the old story, then click the Save button at the top of the screen. Now go back to your site and you'll see that your front page has been updated:



As you might be starting to realise, a CMS is a hugely flexible way of working but it's still not particularly simple. The sheer number of options and menus on the main admin screen look daunting at first. But it's definitely worth persevering, because it will save you time in the long run. Once you've sorted out your design templates, you can create new pages in a flash and add them into the main body of your site in just seconds. Plus, because Joomla is very much based around templates, skins and CSS, it's easy to roll out a new design across all of your site without having to change each individual page. So go ahead and explore Joomla.

WordPress

Like Joomla, WordPress is a free, open source CMS. Although it is used nowadays for creating all sorts of general-purpose web sites, it was originally designed for use in blog publishing and it is in this area that it functions particularly effectively.

As with Joomla, getting an installation of WordPress up and running isn't particularly difficult. The remainder of this chapter will take you through each of the required steps in turn.

Downloading the Software

To obtain your copy of the WordPress software, visit **`www.wordpress.org`** and download the .zip version of the software. At the time of writing, the latest release is version 2.7.1. If there's a later version available, use that one instead.

Once you've download the file, you need to unpack the zip file. If you don't know how to do this, refer to page 128. Once you've unpacked the zip file into a separate folder, you can delete the zip file itself.

Make a Database

Next, you need to ensure that there's a MySQL database on your web server. If you've already followed the instructions on page 124 for creating the thewebbo_cms database then it's safe to proceed. If not, go back and do that now.

Configure WordPress

Unlike Joomla, which you configure via the web after uploading the product to your server, WordPress is slightly different. The configuration data is held in a text file which you need to edit before uploading anything.

Browse through the folder of files that was created when you extracted the zip file. In the wordpress folder you'll find a file called wp-config-sample.php, which you need to open. Use a text editor such as WordPad or NotePad. DO NOT use a word processor like Microsoft Word, as it will save the file as a document rather than plain text, which won't work. If you really must use a word processor, save the file by choosing Save As from the File menu, and ensure that you choose the plain text option.

With the wp-config-sample.php file open in your text editor, there are 2 sections you need to edit. First, look for this section in the file:

```
// ** MySQL settings - You can get this info from your web host ** //
/** The name of the database for WordPress */
define('DB_NAME', 'putyourdbnamehere');

/** MySQL database username */
define('DB_USER', 'usernamehere');

/** MySQL database password */
define('DB_PASSWORD', 'yourpasswordhere');

/** MySQL hostname */
define('DB_HOST', 'localhost');
```

Change 'putyourdbnamehere' to the name of the MySQL database you created earlier. In our case this will be 'thewebbo_cms'. Note that the single quotes need to remain in place.

Change 'usernamehere' to your web hosting username. In our case it's 'thewebbo'.

Change 'yourpasswordhere' to your web hosting password.

You can probably leave 'localhost' as it is, unless your web hosting company tells you that its MySQL database server is located on a different computer, in which case you need to enter its details here in place of localhost. It might be something like 'mysql2.yourhost.com'. But in our case, 'localhost' is what we need.

Finally, scroll down the file and locate this section:

```
/**
 * WordPress Database Table prefix.
 *
 * You can have multiple installations in one database if you give each a
unique
 * prefix. Only numbers, letters, and underscores please!
 */
$table_prefix  = 'wp_';
```

Note that our WordPress installation is configure to add wp_ at the start of each of the MySQL database tables it creates. That should suffice, so you can leave it alone. Just make sure that a table prefix is set, and that the WordPress prefix isn't the same as the prefix you're using for any other product that you have installed on your server. Otherwise there will be problems when table names clash and two different programs attempt to access them.

With the configuration file edited, save it to your hard disk as wp-config.php. Your initial WordPress configuration is now complete. Remember, you MUST save the file as wp-config.php, without the word *sample*.

Upload The Software

You can now upload the WordPress folder from your computer to your web server. Using FileZilla, connect to the server, double-click on the public_html folder to open it, then right-click to create a new directory (folder) within it called wordpress (no capital letters). Then upload the entire contents of the WordPress folder. It will take a minute or three to transfer all of the 600 files, after which you can delete the WordPress files from your PC and exit FileZilla.

Final Configuration

To finish setting up your installation of WordPress, open your web browser and surf to /wp-admin/install.php within your WordPress installation folder. So in this example, where we uploaded the files to the the-web-book.com site in the wordpress folder, the full URL will be:

`http://www.the-web-book.com/wordpress/wp-admin/install.php`

Note that the public_html folder is never part of the URL of your site. That's where all the other folders start from.

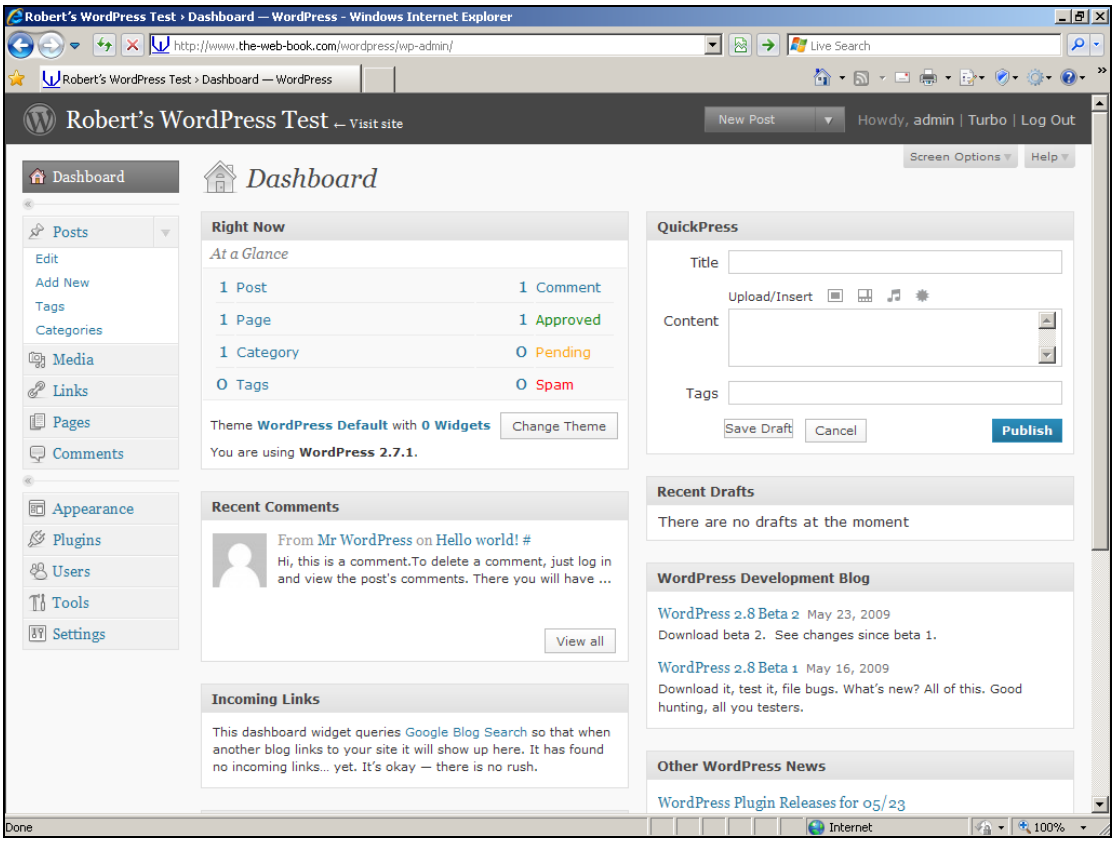
You should now see the following screen:

Choose a title for your blog, and enter your email address, then click the Install WordPress button. Assuming that you edited the config file correctly, and your database host/username/password were correct, this second configuration step should only take a couple of seconds. After which, you will see a username (usually admin) and a password displayed. You'll need these in order to log into your blog to create and edit content, so make a careful note of the password.

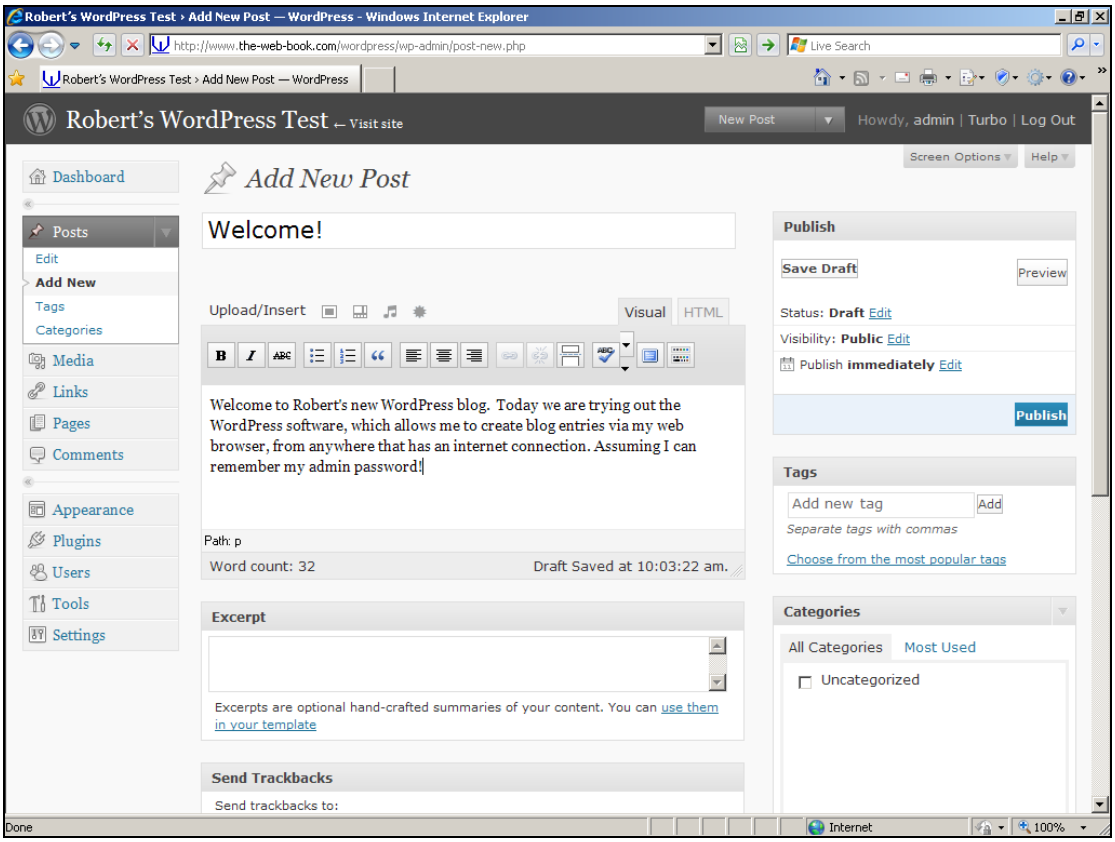
You can now browse your blog (even though there's not much in it yet), or log into the editing system to start creating content.

To view your blog, the URL is **`www.yoursite.com/wordpress`**. To log into the editing system, use **`www.yoursite.com/wordpress/wp-login.php`** instead.

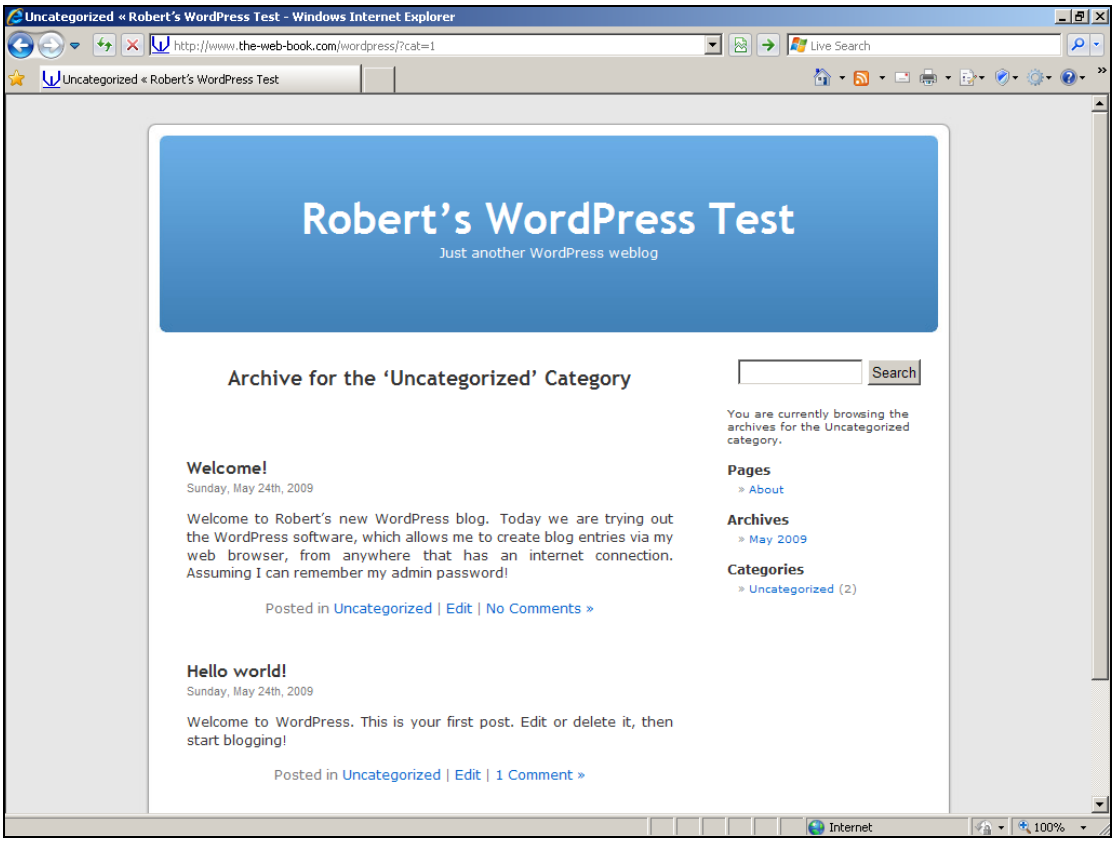
Let's create a first blog post. Log into WordPress with your username and password, and you'll see this screen:



Click the New Post button at the top of the screen, and then type a title and some text for your blog post, like this:



That'll do for now. Click the Publish button, and you're done. Here's the finished post, online for the world to see:



Obviously there's a lot more to explore with WordPress. For a start you'll want to delete the handful of sample postings, and create some categories to which you can assign your posts. Feel free to experiment with the various features, including the appearance of your blog. After all, this is your installation of WordPress running on your own hosting space, so you have total control over what you do with the system. Just remember to keep your admin password safe, so that no one but you can create or edit pages.

phpBB

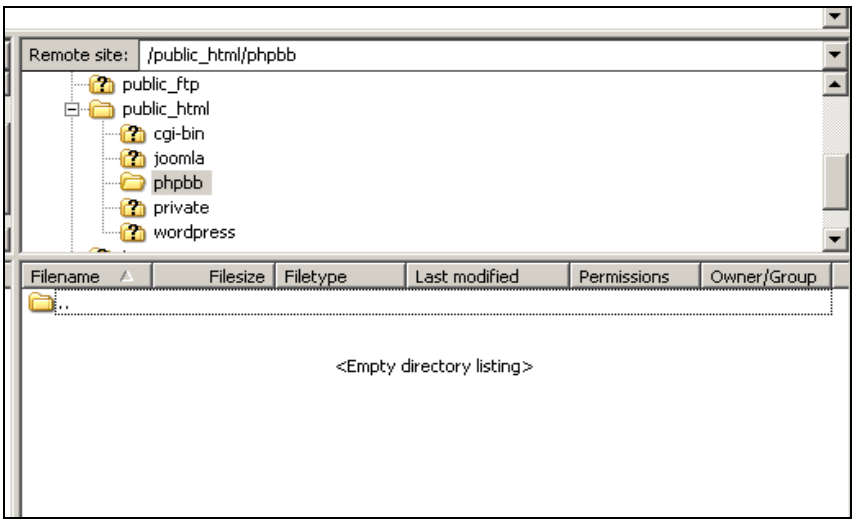
As part of your web site, you may want a discussion forum or two. There are many open-source forum programs that you can download for free and install on your web server in order to provide this. Possibly the best-known is phpBB, so that's what we'll use here.

Like a traditional CMS (Joomla, Wordpress etc), forum software needs access to a database (MySQL is just fine) to store forum postings, user data and so on. So before you go any further, make sure you have followed the procedure on page 124 to create the empty MySQL database.

Once you've done this, go to **`www.phpbb.com`** and download the software. You'll need the zip file version. In this example we're using release 3.0.4 but there may be a later revision available by the time you read this. If there is, use it.

Having downloaded the zip file, extract it to your computer. If you have a program such as WinZip installed, follow your normal procedure. If you don't, Windows should be able to extract the file for you – just right-click it and choose Extract All. Once you've extracted it, you can delete the zip file from your computer. All you need is the folder containing the extracted version.

Next, we need to connect to our server with FileZilla and upload the extracted files. Once you're connected, double-click on the public_html folder to open it. Then right-click on the opened public_html folder and choose Create Directory. Make a directory (ie, a folder) called phpbb. Once you've done this, double-click the phpbb folder and make sure that a) you're in it, and b) it's empty. FileZilla's remote site window should look like this:



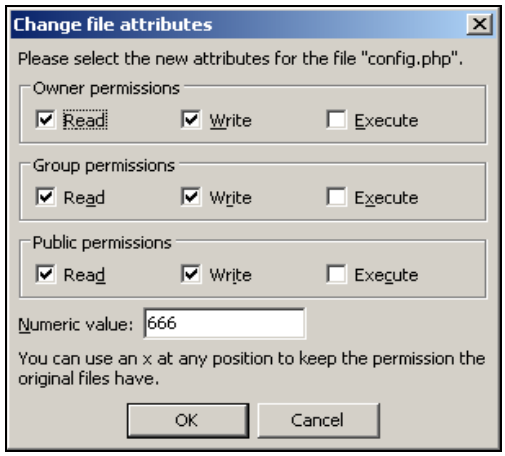
In the left hand Local Site window, navigate to the extracted phpbb folder on your PC. Select everything except the top ".." entry, and press Return to start the upload. There are around 900 files to transfer so it'll take a good few minutes, depending on the speed of your internet connection.

File Permissions

Before you can proceed, you need to adjust some of the access permissions on the files you have just uploaded. When we start to configure phpBB, the installation pro gram will need permission to create its config file on our web server. To allow this to happen, we need to change the access permissions of the config file so that every user on the server, including the phpBB program, can write to that file.

In theory, this means that other people whose web sites are hosted on the same server could also access your config file. In practice, however, almost all web hosting companies configure their servers in such a way as to lock each customer to within the confines of their own area, so there's nothing to worry about. But if you're particularly concerned, check with your hosting company.

To change the permissions of the config file, scroll down the list of uploaded files in the Remote Site window of FileZilla until you find config.php. Right-click this file and then choose File Permissions. You'll see a box like this:



Tick all 3 of the Read and Write boxes, so that the Numeric Value of the permissions reads 666. Then press OK.

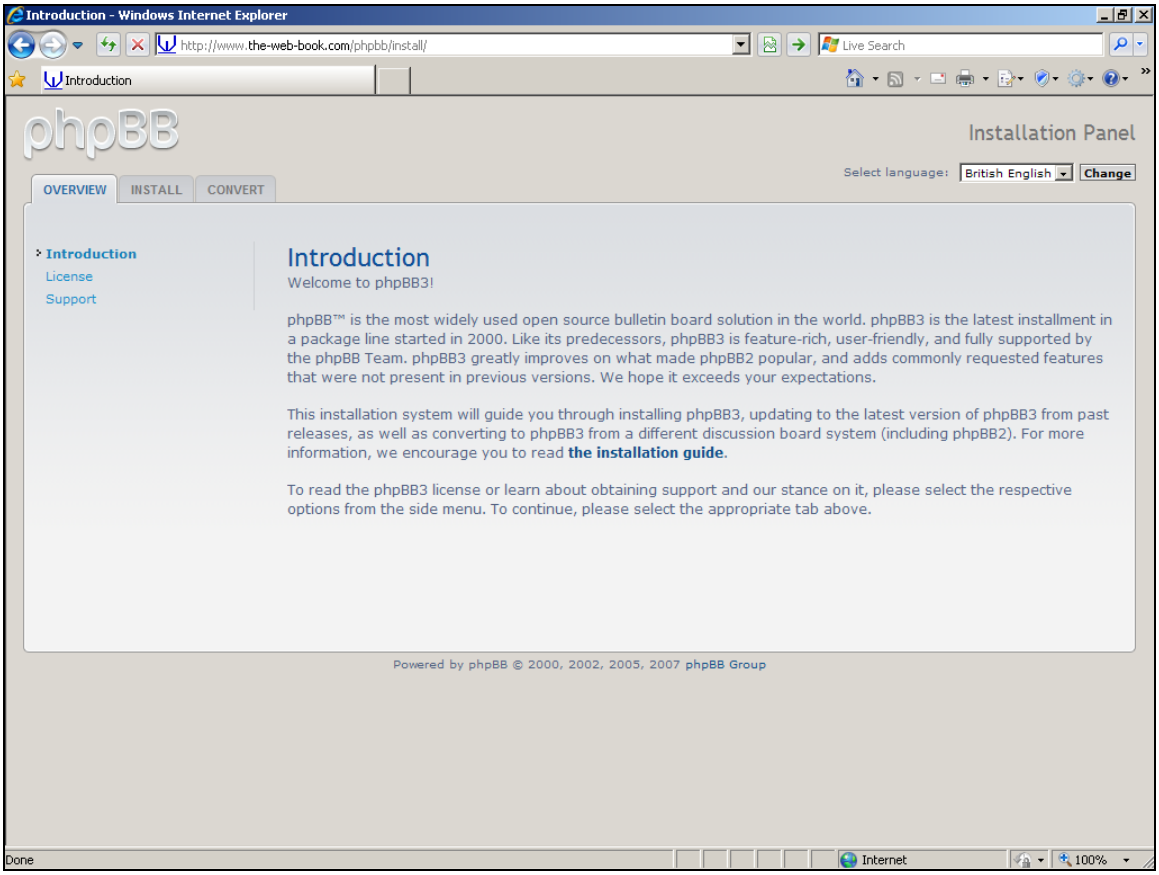
Next, find the store folder. Right-click it, choose File Permissions, and tick all 9 boxes so that its permissions are 777. Again, click OK.

Do the same thing with the cache and files folders. IE, change their permissions to 777.

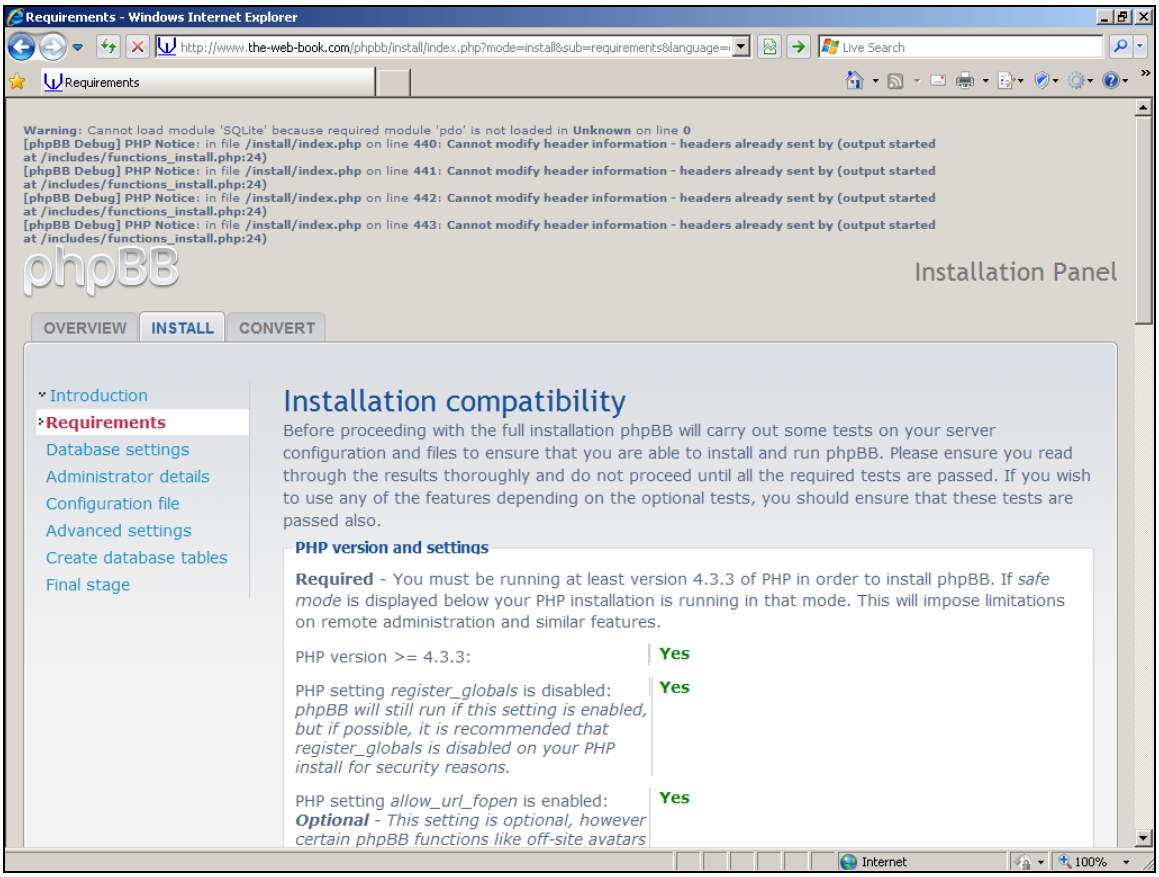
Finally, double-click the images folder to open it. Then, within it, open the avatars folder. Within there, right-click the upload folder and change its permissions, too, to 777.

That's the file permissions changed. You can now exit FileZilla.

Now we need to run the phpbb installer on the web server. We've already uploaded the program files, so open a web browser and surf to the install folder within your phpbb folder. In this example, where we uploaded the files to **`www.the-web-book.com`** in the **`phpbb`** folder, this means **`www.the-web-book.com/phpbb/install`** and it brings up the following screen:



Click the Install tab to get started, and you'll see the following:



Something is clearly not right. There are some PHP error messages at the top of the screen. However, these are actually notices rather than actual errors. A notice is more friendly advice than a warning of a major problem. Although it could signify an error in the program, most notices happen because the configuration of the server isn't precisely what the program was expecting. In many cases, you don't get to see notices because the server is configured not to display them. But clearly this one is happy to show them.

We'll persevere for now. This is only an installer. If the phpBB program appears to install and function correctly, we can stop worrying. Obviously, if those same notices are shown to the users of our forum then this will be unacceptable, and we'll need to try some different software instead, rather than phpBB.

Click on Start Install, and enter the information that's requested. For the database type, choose MySQL. The database server should be specified as localhost unless your hosting company tells you otherwise. Leave the database server port blank, unless you specifically know that it needs to be set.

For the database name, enter the name of the MySQL database that you created as per the instructions on page 124. This is thewebbo_cms in our example. The database username

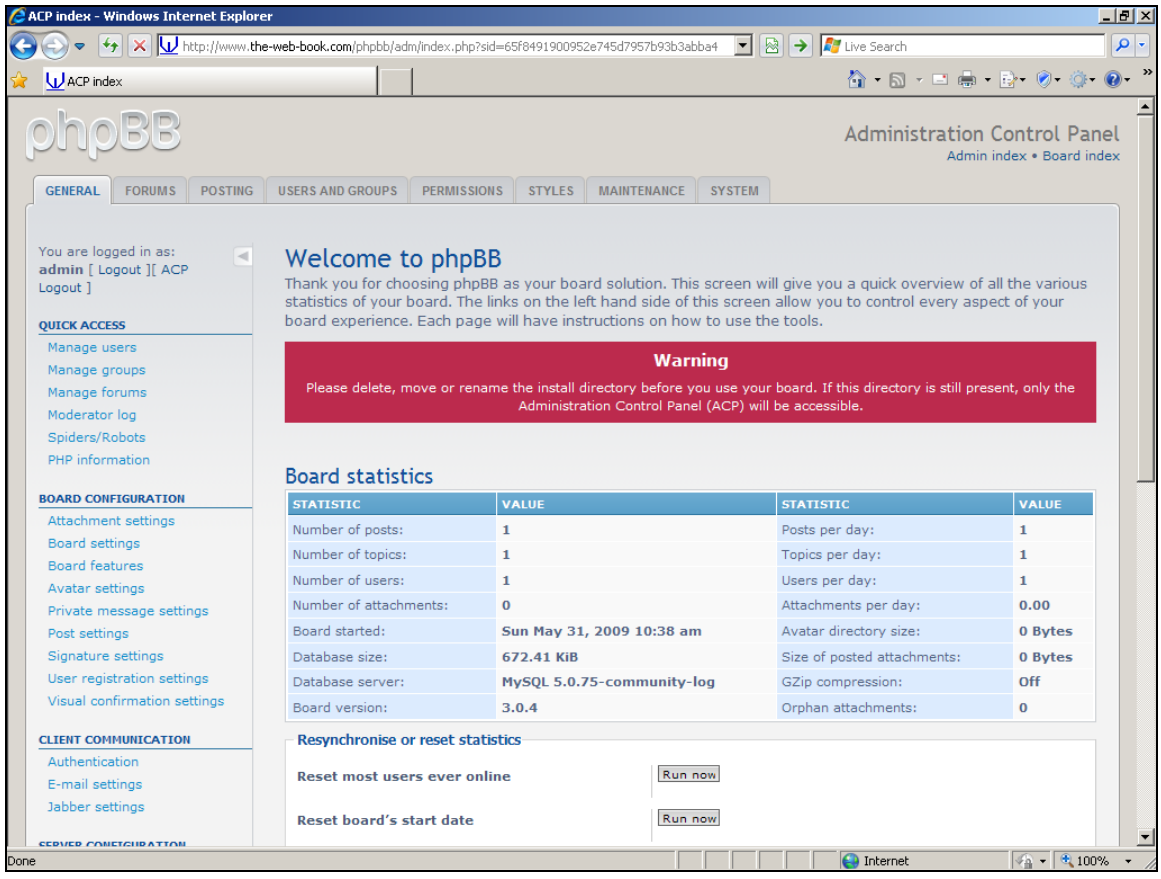
and password should be the same as you use for connecting to your web hosting control panel.

You need to select a database table prefix which is not used by any other programs that are configured to use this MySQL database. The installer suggests phpbb_, which is fine, but you can change it if you want.

Click on "proceed to next step". You'll then be asked to choose an admin username and password for your forums. Choose something non-obvious for both. The last thing you need is for someone to guess the combination and wreak havoc on your board.

We're almost done configuring phpBB. You can ignore all the options on the Advanced Settings screen for now.

Keep clicking "proceed to next step" until you see a login button. Click that button and you'll see:

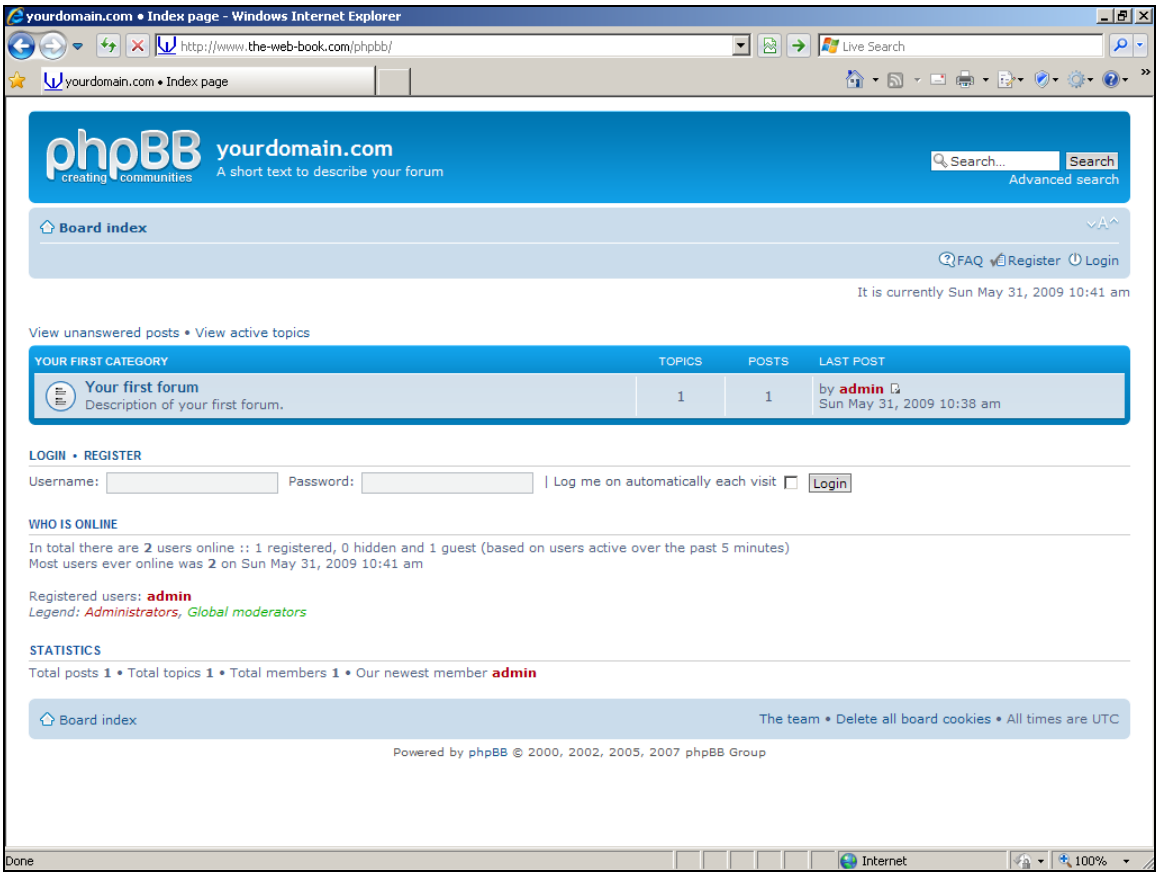


Before you can start using your forum, we need to tidy up a few things. First, remove the zip file and the extracted files from your PC as these are no longer required. All that matters are the files on the web server.

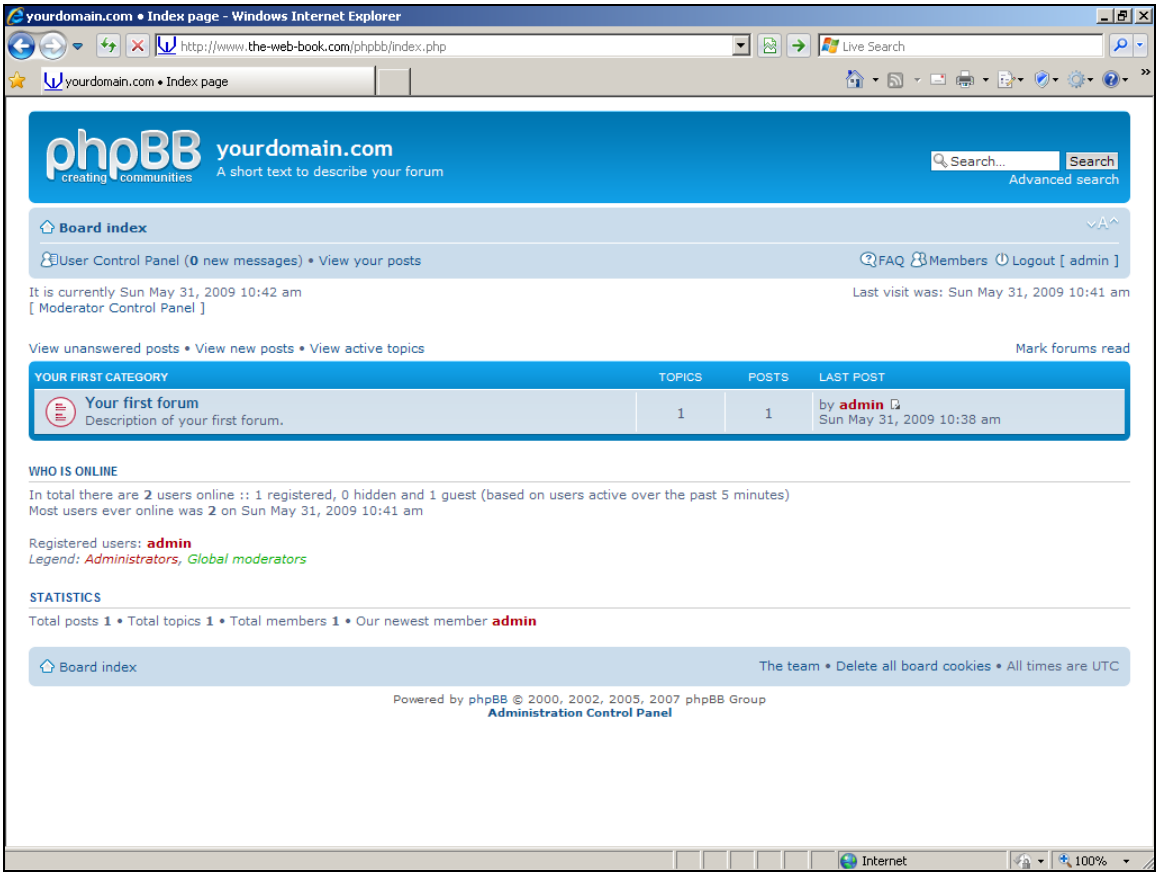
Next, open FileZilla and connect to the server. Go to the public_html folder, then the phpbb folder, and find the config.php file. Now that the installer has done its job, and created the config file, we need to change its permissions again so that no one can change it. Right-click config.php, choose File Permissions, and change the 3 sets of permissions to read/write, read, and read. The numeric value should say 644. When it does, press OK.

Just one more task. Right-click the install folder and choose Delete. If you don't do this, someone else could run the installer again, and wipe out all of your settings.

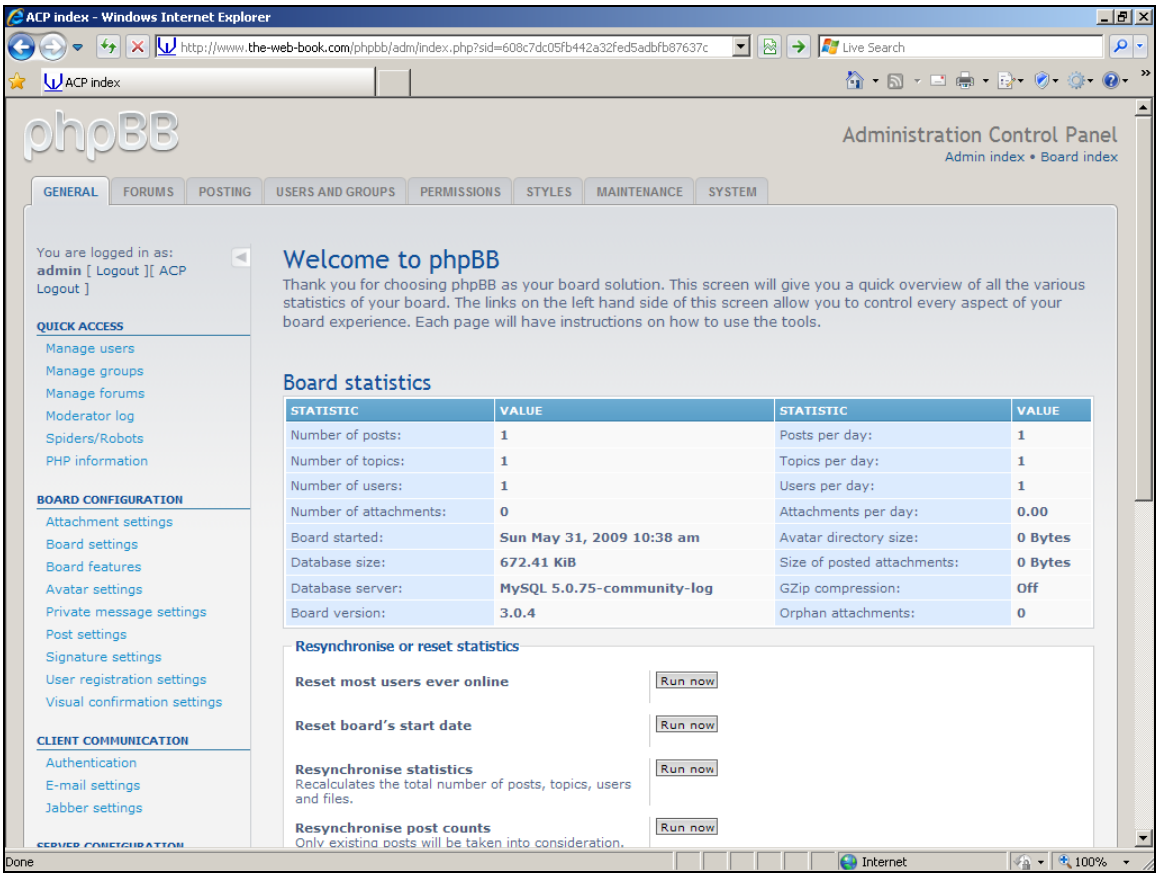
When FileZilla has finished deleting the folder and its contents, close the program and return to your web browser. Surf to your forum, which will be in the phpbb folder on your site. In our case that means **`www.the-web-book.com/phpbb`** and the screen will look like this:



We now have a fully working forum on our web site. Click the Login link and log in with your username and password that you just chose. The screen will look like this:



Click the Administration Control Panel link at the bottom of the screen, and enter the admin username and password that you chose earlier. You'll see:



You're now all ready to start creating forums and customising your phpBB system. To get started, click the Forums tab and explore. Remember, however, that the web is full of people who get their kicks from finding poorly-protected forums that they can vandalise or fill with spam. So ensure that you set the security options accordingly. This means, at the very least, forcing people to sign up and log in before they can post messages, and verifying their signup by asking them to reply to an email message.

You may also want to consider, at least in the early stages, configuring the forums so that new members can't post until their account information has been verified by you. This will add to your workload in administering the system, but does help to ensure that spammers don't attempt to sign up with multiple accounts that are merely used to flood your forums with advertising.

Plogger

So far, the products we've installed on our web server are primarily text-based. These include a Content Management System, a blog, and a forum. Plogger is an image gallery application, allowing you to upload pictures to your server and create online albums for people to browse. However, it still requires a MySQL database, as it uses this to store details of images and some information about them. It doesn't actually store the uploaded pictures in the database, though – these are uploaded to a folder on the server within the main Plogger directory.

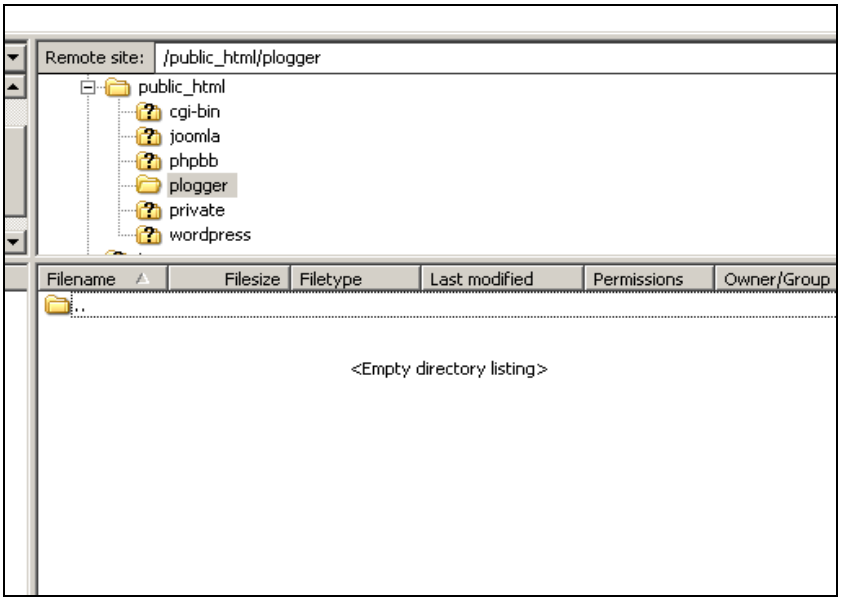
Web servers aren't generally good at handling image files. Programs such as Plogger often boast the ability to manipulate pictures, such as resizing them for display, generating thumbnails for browsing, and compressing them to save space. However, such functionality generally relies on the availability of specific features that are not always available on all web servers. Such features include the ImageMagick graphics library. In the absence of such features, certain parts of the program may not work properly, or at all. You should therefore take special care to read the list of server requirements before deciding whether to install an image-based product such as Plogger on your web site. If you have problems with a specific product, there are always plenty of others to choose from.

Getting Started

To obtain the Plogger software, go to **`www.plogger.org`** and click on the Download Plogger link to get the zip file. Right-click the downloaded file, and choose Extract All to unzip it.

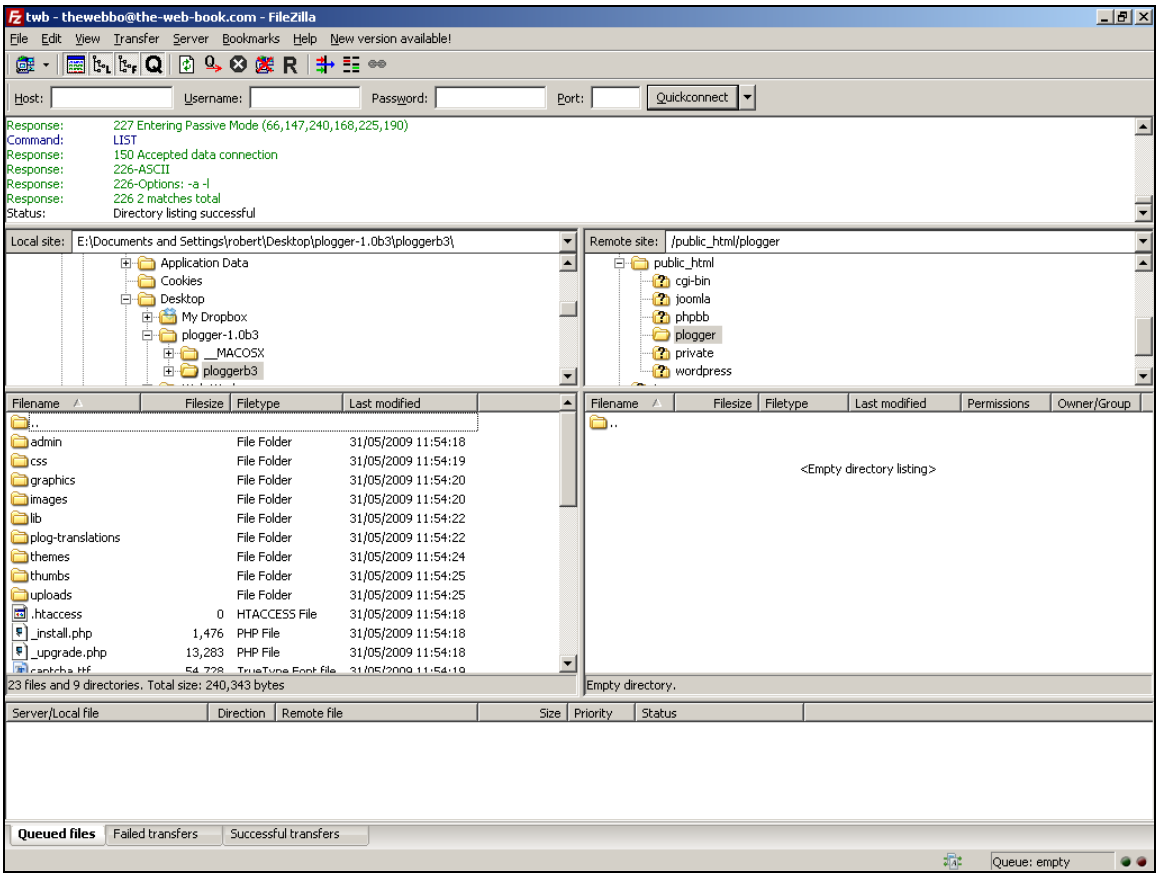
Next, connect to your server with FileZilla. Double-click the public_html folder to open it and then, within it, create a new folder called plogger (it's best to stick with lower case letters rather than creating Plogger).

Ensure that the plogger folder is open and that it's empty. Assuming this is the case, the Remote Site window in FileZilla should look something like this:



On the left hand side of the screen, in the Local Site window, browse to your desktop or wherever you extracted the zip file to. Then double-click the Plogger folder to open it (you don't need the MACOSX one).

Select every file and folder in the left hand window except the ".." folder, and then press Return to begin uploading the Plogger system to the web server. FileZilla should look like this:



When the upload has finished, you can close FileZilla and also delete the extracted zip file from your PC.

Next, make sure that you have created the test MySQL database on your server, which we have been using for all of our product installations. If you haven't done so, refer to the procedure on page 124.

The Installation Process

With the Plogger system uploaded to your server, you can now surf to it and begin the configuration. Open a web browser and go to the _install.php page within your Plogger system. In the case of our example site, that means going to:

`www.the-web-book.com/plogger/_install.php`

You should see a screen like this:

Fill in the connection details for the MySQL database, so that Plogger can access it. The host should be set to localhost unless your hosting company has advised you otherwise. The username and password will normally be the same as you use for logging into your site's hosting control panel. In our examples the database name is thewebbo_cms.

When you've finished, click the Install button and Plogger will finish its setup process. Once it has done so, it will create an admin account for you, so that you'll be able to log in and upload pictures. The admin password is randomly generated and will be displayed on the screen, so make sure you don't close your browser until you've made a note of your admin username and password!

The screen should look like this, assuming the installation and configuration succeeded:

When you've made a note of your admin username and password, click the Proceed button. Plogger will then tell you the URLs for browsing your gallery and for logging in as an administrator. These are, in the case of our example site:

`www.the-web-book.com/plogger`	for public access
`www.the-web-book.com/plogger/admin`	for the admin login screen

You may find you also see a few error warning notices displayed at this point. If this happens, check that you can browse and administer the site correctly. Assuming you can, there's nothing to worry about. If the problems persist, check your hosting company's online FAQ or support section to see if there's any reason why these error message might be displayed. Also (and this is a good tip for any similar situation), try typing or pasting the full text or the error message into a search engine such as Google, to find out if anyone else has reported similar problems. Hopefully they'll also have reported how they solved them.

Uploading Your Pictures

With Plogger now installed on your web server, you'll need to upload a picture in order to test that it's working properly. Surf to the admin page, as listed above, and enter your admin username and password when prompted. You may also wish to try entering an incorrect

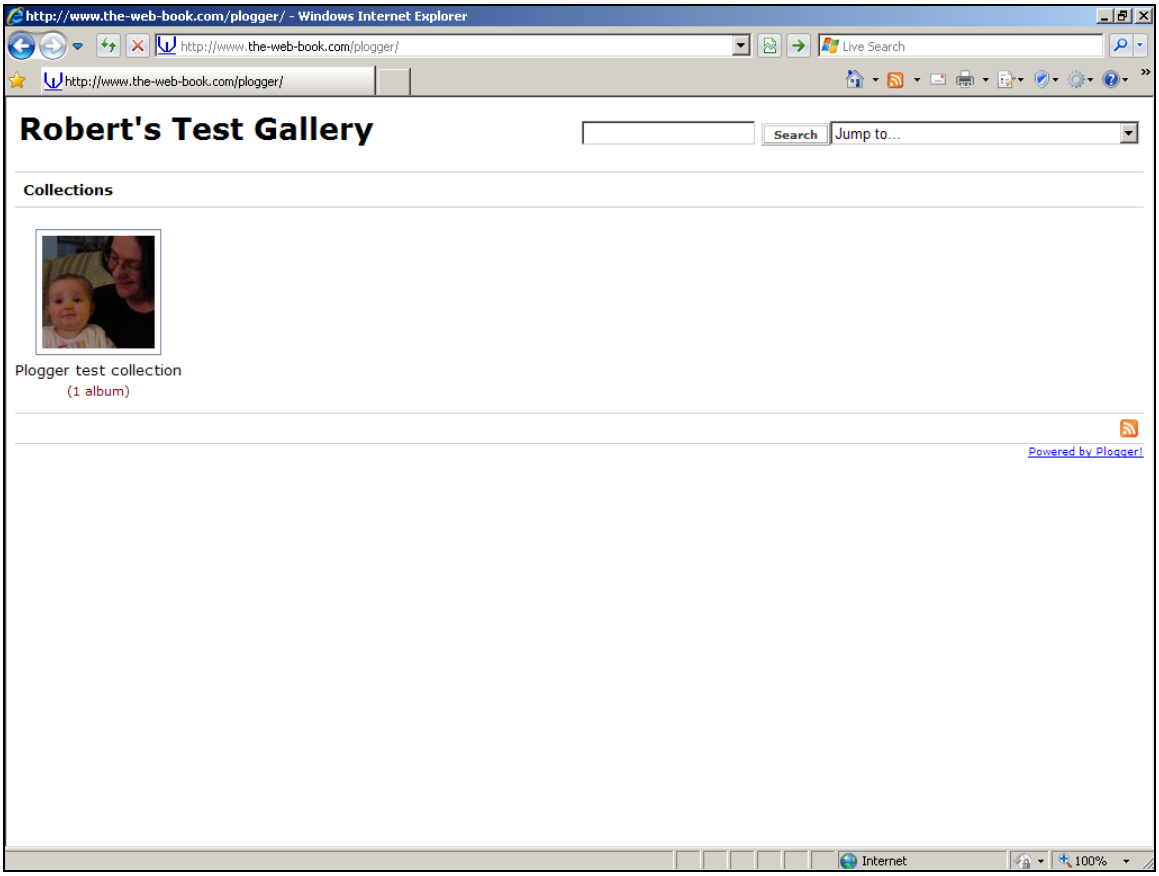
password, just to make sure that the security is working properly. Performing such a test each time you install any new product on your server is always a good idea.

Once you're logged in, you'll see this screen:

Click the Browse button and navigate to an image file on your computer. Add a caption and a description (this is important, as you'll see later). Then choose which album you want to add your picture to. You'll probably want to create a new one rather than putting it in the test collection.

When everything's ready, click the Upload button and wait for the message that says your upload was successful.

When it's done, log out and surf to the public part of your site rather than the admin section. You should see your gallery displayed. To check that Plogger is correctly storing information in your MySQL database, use the search box. Just type a word or two from the caption or description, and check that Plogger finds your picture.



With everything working properly, you can now proceed to make your site look as you want it to. Log into the admin section again and explore the various features that are available.

Avoiding Data Overload

When you sign up for a web hosting package, your account comes with a certain amount of disk space on the server to store your information. If you're storing mostly text, you don't need to worry about hitting your limit. But if you intend to store lots (as in, thousands) of pictures, be aware that they eat server space like nothing else. Especially if they are high-resolution multi-megapixel images, which can typically occupy up 10 MB each.

If you think this might be a problem for you, check the small print of your web hosting package to find out just how much server space you get. Remember, too, that people who view or download your pictures will use up your bandwidth allocation, so you may want to check that as well. For example, if you have a 10 MB picture on your site and 100 people download it, that's 1 GB of bandwidth traffic from your server.

Equally, if you upload 100 pictures to your gallery, at 10 MB each, that's 1 GB of traffic from your own broadband connection. Which, if you are paying for a broadband service that only allows you 2 GB a month, is a fairly large chunk. If this is the case, check whether your ISP

allows free uploads overnight that don't count towards your monthly limit. If so, upload the pictures then, or downsample them (reduce their resolution and thus the size of the files) before uploading them.

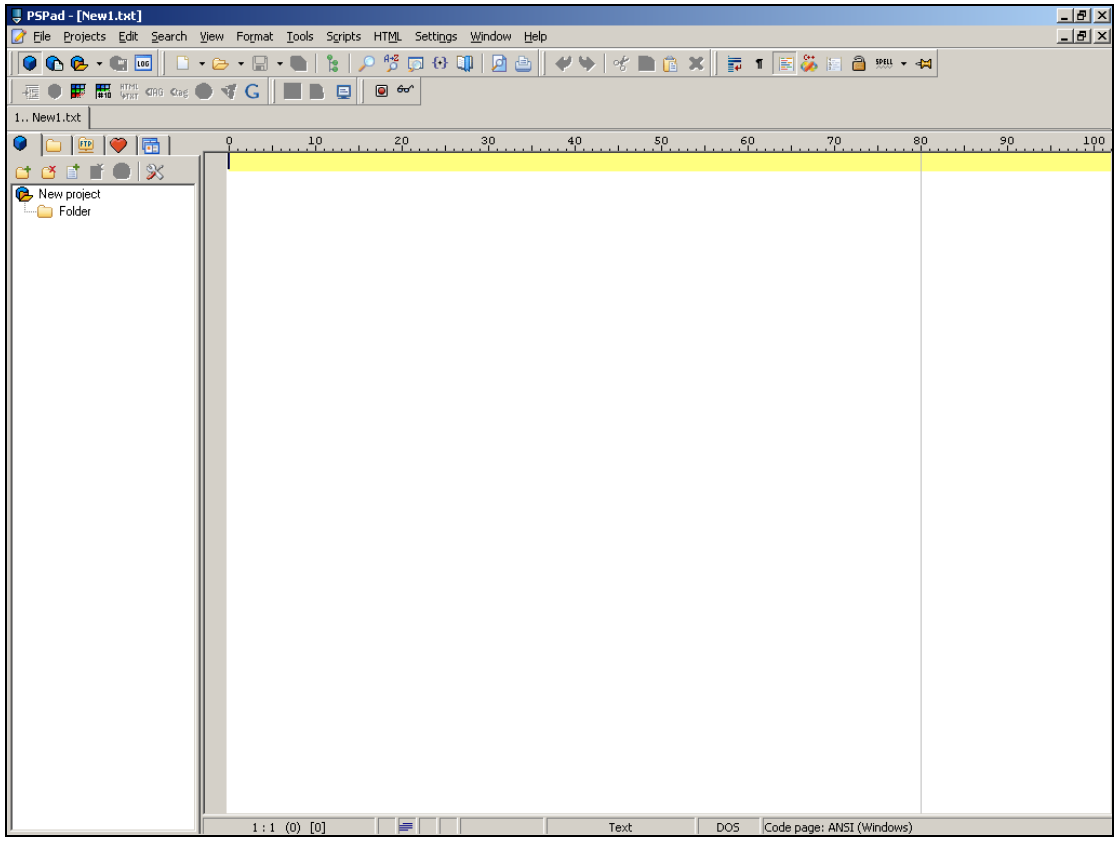
Installing the PSPad Editor

While Amaya is great for editing HTML files, it's not so good if you're writing JavaScript or PHP. The solution comes in the form of an excellent program called PSPad. It's a text editor with two features that make it ideal for writing programs. First, it understands the syntax of JavaScript and PHP, so it can alert you if you make a mistake. Second, it has a built-in FTP facility, thus making it easy to upload your programs to the web server if you need to do so. And like everything else we've used throughout this book, it's downloadable free of charge.

The following chapters cover how to write programs in JavaScript and PHP so, if you intend to follow them, you need to install PSPad now.

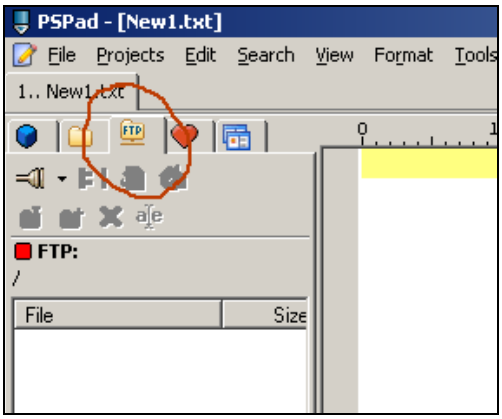
To obtain the program, visit **`www.PSPad.com`** and download the full installer. This will copy a small (only 4 MB or so) .EXE file to your computer. Double-click that file to begin the installation, accepting all of the default options (though you'll probably want to reject the offer of additional language support unless you really do want to have the option of seeing the PSPad menus in Indonesian and other non-English tongues). With the program installed, you should now see a PSPad icon on your desktop.

Double-click the PSPad icon to start the program, if it's not already started automatically after being installed. You'll see the main screen, which looks like this:

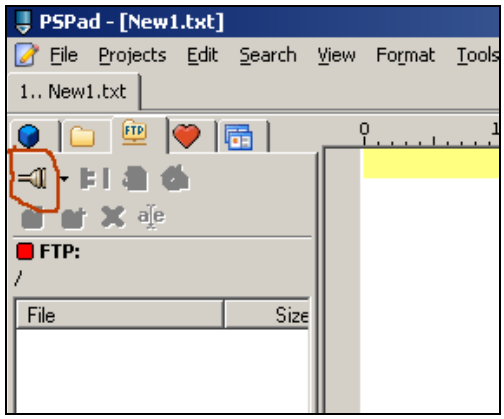


We now need to configure the program before we can use it. Mainly, this means setting up the FTP details so that PSPad knows how to copy files from our PC to the web server. First, though, you can turn off that confusing menu of assorted icons at the top of the screen, as you don't need them. To do this, just press the F2 key.

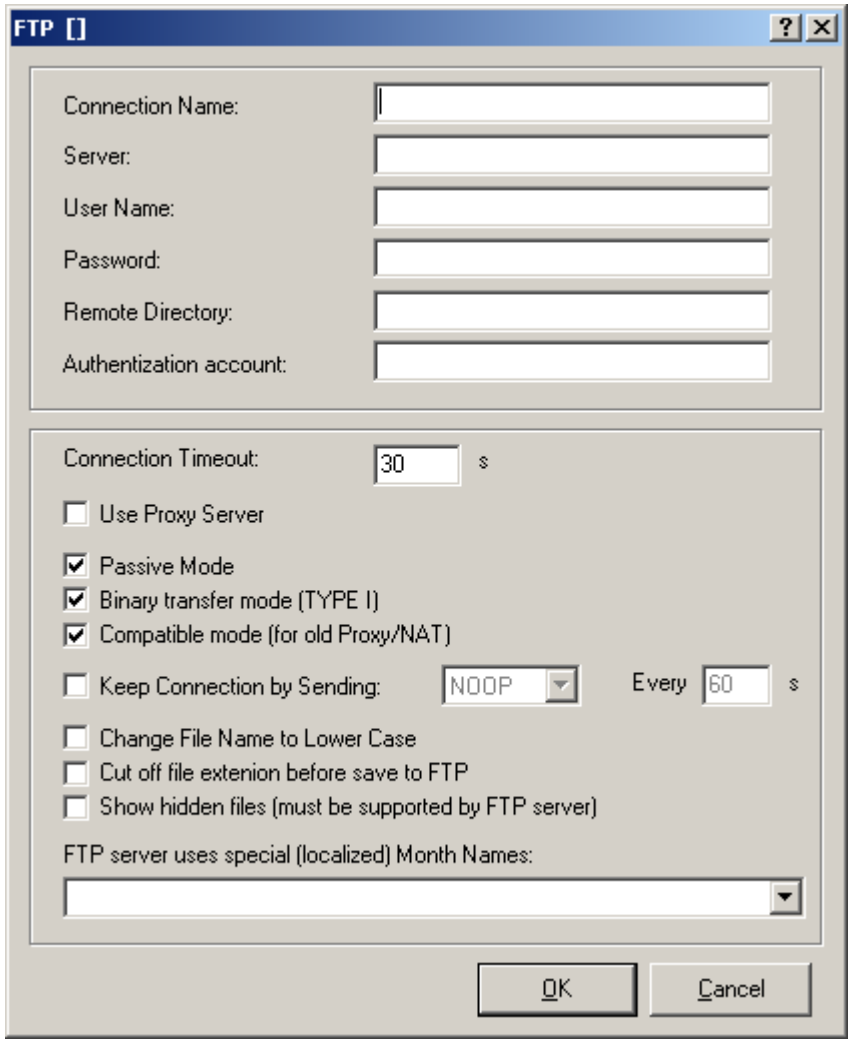
Now to configure the FTP site of things. Click the FTP tab on the left hand side of the screen, to get to PSPad's FTP facility:



Then click the connection icon:



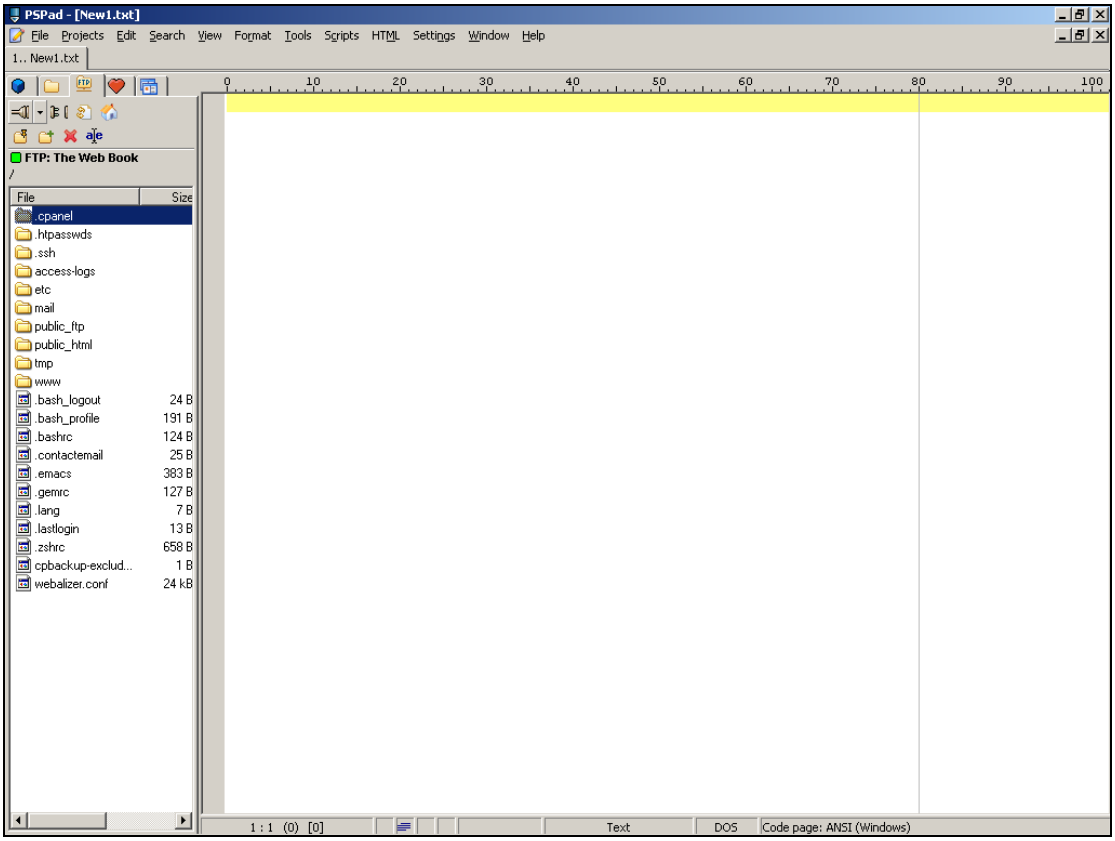
A dialogue box will appear, into which you can enter the details of your FTP connection to the server. This is very similar to the process that we went through right at the start of this book, for configuring FileZilla.



For the Connection Name, just choose a name for this connection. This is purely for your benefit so that, if you configure PSPad to connect to multiple servers, you can choose one from a list. I'll call mine The Web Book.

Following the server-specific examples as per the earlier examples in this book, for the site at the-web-book.com which is hosted by hostmonster, the required details are the-web-book.com for the server, thewebbo for the username, and the password that I normally use for connecting to my web hosting space.

Enter just a / symbol for the remote directory. Leave everything else as it is, and press the OK button to finish. You'll return to a list of options that shows your list of available connections (just the one so far, namely the one you just created) and some buttons. Click the top button, labelled Connect, to connect to your web server. Assuming it works, you should see something like this, with the contents of your server listed down the left hand side.



That's all you need to do for now. With everything working, you can exit PSPad for the moment.

Javascript

HTML is a very full-featured language. When coupled with CSS, it's capable of producing just about every design of web page you could ever desire. But HTML is only a markup language, for specifying how pages should look. Which is just fine if you intend your web site to act as an electronic version of a newspaper, magazine, book or brochure. But sometimes you'll want to create web sites that exploit the power of computers, including features that go above and beyond the capabilities of electronic publishing. For example animation, online databases, and so on.

To add such features to our sites, we need to leave the safe environment of HTML markup and enter the world of web-based programming. Once you know how to write programs, just about anything becomes possible.

The following chapters explain the basics of web programming using Javascript and PHP. I'll show you how to create basic web applications (often known as Web 2.0 sites) using Javascript and PHP. But be aware that this is a complex subject which takes a couple of years to master, and this is best done by practising rather than reading. So now might be a good time to think of a Web 2.0 application that you've always wanted to write, in order that you can apply what you're about to read to a real situation.

Choose Your Side

There are two ways to write web-based programs, and understanding the difference between them is vital because each method is used for specific purposes. As a reminder, the following explanations use the term "visitor" to describe a person who visits your web site, ie who views it online by using a web browser such as Internet Explorer or Firefox.

The first programming method is called client-side programming. This means writing programs that will run on the visitor's computer, within their web browser. Just about all modern web browsers support this, via a programming language called JavaScript. To write a JavaScript program, you include it within the body of a web page along with the HTML code. Note that JavaScript is not the same as Java, and you should never refer to it as such.

The second programming method is called server-side programming. Here, you write programs that run on the web server. The browser never gets to see the program code. It merely sees the results, which normally consist of HTML code that is sent to it. The fact that this code was generated by a program running on the server, rather than being stored in a pre-written .html file, is unknown to the browser. The most common languages for writing server-side programs are PHP, Perl, Python, Ruby and Java, as well as Microsoft's proprietary languages ASP and ASP.NET.

So how do you decide which type of program you need to write? Should it be client-side or server-side? It depends entirely on what you want to achieve, and this should become clearer as you read the chapters that follow. It also means that, to develop full-featured sites, it helps to know a little about both types.

In the next chapter, we'll cover server-side programming using PHP. For the remainder of this chapter, we'll cover client-side JavaScript programming. First, though, if you haven't already done so, follow the instructions on page 162 and install PSPad on your PC.

Start PSPad and connect to your server, so that the list of files and folders appears down the left hand side of the screen. The list should look familiar from when you used FileZilla. These are the contents of your web server, brought to you via the FTP component in PSPad.

Double-click the public_html folder to go to the place on the server where your web-accessible files are stored. If you previously installed products such as joomla, wordpress, phpbb and plogger, you'll see their folders here.

We're going to create a new file with which we can start exploring JavaScript. So right-click on the top folder icon (the one that just consists of 2 dots) and choose New File. Enter jstest.html as the new of your new file and press OK. Verify that PSPad has created the file on your server, which should appear at the bottom of the list.

Now let's create a very simple web page in that file. Double-click jstest.html to open it, and verify that the blue title bar of your PSPad window changes from new1.txt to jstest.html. This tells you that the correct file is open, and ready for editing.

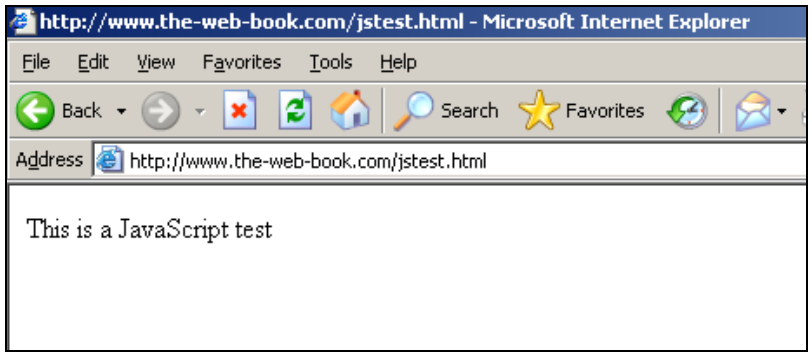
Into your new jstest.html file, type the following:

```
<HTML>
<body>
This is a JavaScript test
</body>
</HTML>
```

When you're done, go to the File menu and click Close. Click Yes to confirm. The PSPad program has now created your jstest.html file directly on the web server.

This is a different way of working, compared with how we did things previously. There's no need to upload our test page with FileZilla as it's already there. There's no local copy in our Web Work folder.

Let's make sure it worked. Quit PSPad and open your web browser. Surf to your site and page. In this case it's **`www.the-web-book.com/jstest.html`**, which brings up the following:



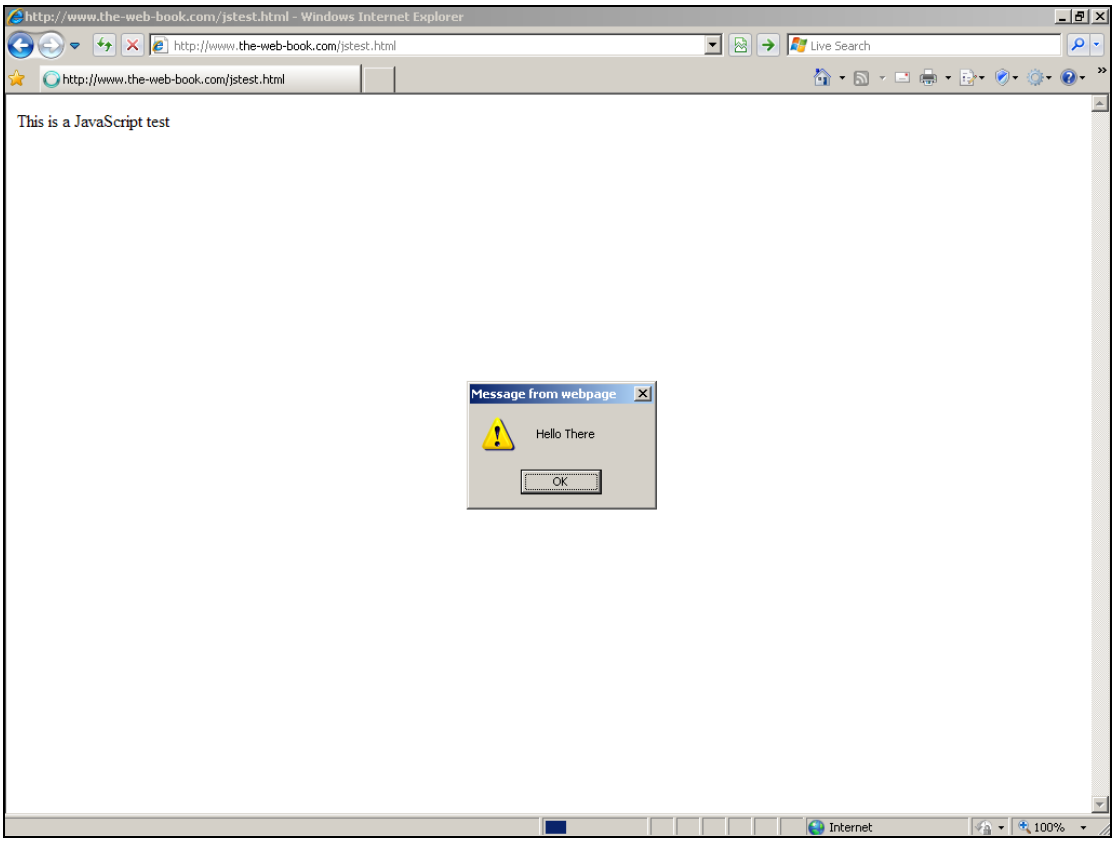
So far so good. But that's plain HTML rather than JavaScript, and nothing we haven't seen before. So let's remedy that.

Fire up PSPad again. Connect to the server by clicking the FTP tab, and then the first icon in the row beneath it. Go into your public_html folder then double-click your jstest.html file to open it.

Now, before the **`</body>`** tag, add the following:

```
<script type="text/javascript">
alert("Hello There");
</script>
```

When you're done, save the file to the server (File, Save should do it, as PSPad has its handy built-in FTP program). Then quit PSPad, open your browser, and surf to your jstest page again. You should see the following:



In addition to the text on the page, there's now an alert box with a message in it. Not something that HTML can do on its own, but our little JavaScript program enables us to do clever things like this.

Our 3 lines of JavaScript aren't particularly clever. You should be able to decipher how it works without too much trouble. The line which does all the hard work is:

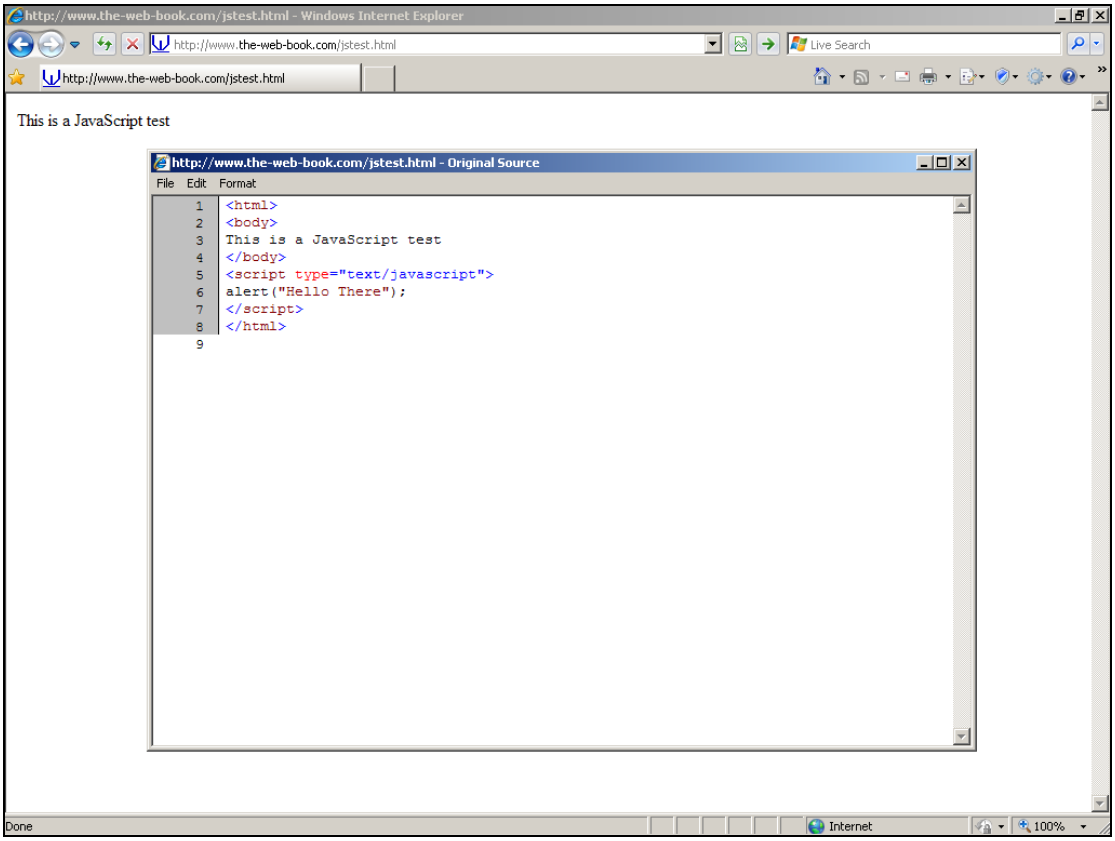
```
alert ("Hello There");
```

The "alert" command is the one which creates the alert box.

As I mentioned earlier, JavaScript code co-exists with HTML code in the same .html file. The **<script type="text/javascript">** tag tells the Web browser that you're switching from HTML into Javascript, and the **</script>** tag then switches back into HTML again. Note that these tags don't require semi-colons as they themselves are HTML tags rather than JavaScript commands.

As already discussed, JavaScript is a client-side language rather than server-side. I'll repeat this again, as understanding the difference is vitally important. The most vitally important point to note is that, because the JavaScript code goes in the HTML file, it's visible to anyone who uses the View Source option. If you surf to your jstest.html page again and choose the

View, Source option in your web browser, you'll see what this means. The screen will look like this:



Although JavaScript is very useful, and great fun, the fact that anyone can view your JS code in the browser means one important thing: never use JavaScript for anything related to security or authentication on your site. For example, it's perfectly possible to create a password system in JavaScript whereby the visitor is asked to enter a password and is only shown the "secret" page if he enters it correctly. But if you implement this in JavaScript, it's trivial for a hacker to look at your source code, find the bit which effectively says **"if password = 'sesame' then let him in"**, and thus deduce the password.

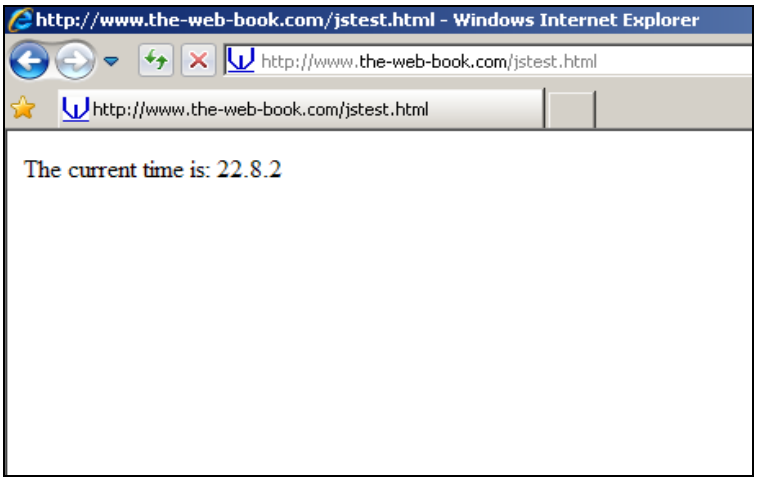
The way round this conundrum is to use a programming method whereby the code for the program doesn't appear in the page. That's what server-side programming with languages such as PHP is all about, which we'll cover later. For now, though, back to some more Javascript examples.

Edit your jstest.html file again, and replace the entire contents of the file with the following:

```
<HTML>
<body>
The current time is:
```

```
<script type="text/javascript">
var d = new Date()
document.write(d.getHours())
document.write(".")
document.write(d.getMinutes())
document.write(".")
document.write(d.getSeconds())
</script>
</body>
</HTML>
```

Save the file, surf to your jstest.html page, and you'll see something like this:



Let's example this in a little more detail, and you'll hopefully realise that Javascript isn't as complex as it might first appear. First, here's the code again, with line numbers to aid the explanation that follows.

```
1  <HTML>
2  <body>
3  The current time is:
4  <script type="text/javascript">
5  var d = new Date()
6  document.write(d.getHours())
7  document.write(".")
8  document.write(d.getMinutes())
9  document.write(".")
10 document.write(d.getSeconds())
11 </script>
12 </body>
13 </HTML>
```

Lines 1, 2, 3, 12 and 13 should be familiar to you. This is standard HTML code, as explained in the previous chapters. This includes the "the current time is:" message, which, as you can see from the screen picture, gets displayed on the finished web page just as you'd expect it to.

Line 4 tells the browser that we're now in JavaScript mode, and that everything which follows comprises a JavaScript program rather than normal HTML. Up until line 11, which ends the program and tells the web browser that we're back with standard HTML again.

Incidentally, you can include as many JS scripts in an HTML file as you wish. Just use a <script> tag at the start of each one, and end with </script>.

Now let's look at how the JS code works. Line 5 creates a new variable called d, which is set to the current date and time. That's what the date() function in JS does – it returns the current date and time.

You'll remember that, at the start of this chapter, I pointed out that JavaScript is a client-side programming language. That is, it runs on the web client (the visitor's browser) rather than the web server. By using the date() function, this becomes evident and significant, because the date and time which get returned are those of the visitor's computer, wherever that happens to be. If it's 7pm on the user's computer, the page will display the hour as 19.

Contrast this with what would happen if we had retrieved the time from the server. 7pm in the UK is 2pm in New York. So if the server is in New York and a UK-based visitor checked the page at 7pm, he'd be told that the time is 2pm. Server-side programming has its uses, as we'll see later, but retrieving the date and time isn't necessarily one of them. At least, not if you intend to display it for the visitor.

Line 6 introduces the document.write statement. Here's yet another fundamental concept that you need to understand if you're getting into web programming. This statement inserts content into the web page, just as if you'd written it yourself. But in this case, we're not inserting literal text that we know in advance. Instead, we're inserting the "hours" component of variable d.

The remaining lines, which insert the minutes and seconds, as well as a couple of full stops, should be self-evident.

At this point, it's vital that you understand how document.write works. So take a look at these examples, which both produce exactly the same results:

```
<HTML>
<body>
```

```
<script type="text/javascript">
document.write ("Hello")
</script>
</body>
</HTML>

<HTML>
<body>
Hello
</body>
</HTML>
```

Before we leave this example program, there's one slight problem that we need to fix. The minutes and the seconds don't look right, because there's a zero missing from the front of each of them. Which is just as we'd expect if, for example, the current time is 8 minutes past the hour. The date() function is doing what it's supposed to do, and returning a value of 8 for the minutes. If we want to display it in a cosmetically-enhanced way, it's up to us to add the necessary code.

So, edit the jstest.html file and add the following after line 7:

```
if (d.getMinutes() < 10 )
{
document.write("0")
}
```

And add the following after what's shown above as line 9:

```
if (d.getSeconds() < 10 )
{
document.write("0")
}
```

The purpose of this code should be obvious. Before we display the minutes, we check whether the value is less than 10 and, if so, insert a zero into the document (ie, the web page). Ditto for the seconds.

This also shows you how to use the "if..." statement in JavaScript. Specifically, note the way that everything which should be executed if the statement is true, goes in curly brackets.

The ability to perform date-based calculations in web pages is very handy. For example, you could easily write a web page which automatically alters its appearance according to the

season. Just create 4 images that correspond to spring, summer, autumn and winter. Then use some Javascript to find out the current month and perform one of 4 different **<img src=xyz.jpg>** operations, where xyz corresponds to the name of one of your image files. Alternatively, you could automatically add a "Merry Christmas" footer to your home page if the month is December and the date is the 25th.

Javascript and Semicolons

A word is in order about Javascript and semicolons. You'll have noticed in the examples that some lines end with a semicolon while others don't. In Javascript, each line of code is considered to end either at the end of the line or at the semicolon. So if you write each statement on a single line, you don't need a semicolon. But if you're including multiple statements on a single line, they need to be separated with a semicolon. For example, these are all valid (I've coloured the semicolons for clarity):

```
document.write ("Hello")
document.write ("Hello");
document.write ("Hello"); document.write ("Hello again")
document.write ("Hello"); document.write ("Hello again");
```

But these are not:

```
document.write ("Hello") document.write ("Hello again");
document.write ("Hello") document.write ("Hello again")
```

Email Address Obfuscation

Those who send spam email for a living need to find a constant source of new email addresses to which they can send their annoying, offensive advertisements. They do this in various ways, one of which is to scour the internet looking for web pages that contain email addresses. It's actually not difficult to do. All that's required is a program to randomly connect to millions of web pages and scan the content for an "@" symbol. Then look for the nearest space to the left, and space to the right. Everything between those two spaces is likely to be an email address, ready to add to the spammer's mailing list.

Because of this, one of the best ways to ensure that your email address remains relatively free from spam is to never mention it on a web page. However, you probably also want people to be able to contact you by email. Thankfully, Javascript provides a great solution to the problem.

Instead of including your email address literally on the page, such as r, write a small Javascript program which uses one or more document.write() statements. For example, whenever you want to quote your email address, use the following on the page instead:

```
<script language="javascript">
document.write("robert");
document.write("@");
document.write("the-web-book.c");
document.write("om");
</script>
```

Such behaviour is enough to confuse the vast majority of the so-called automatic email address harvester programs that are continually scanning the web in search of fresh addresses.

If you want something even smarter, search online with your favourite search engine for "javascript email address obfuscation". This will lead you to a variety of web sites which allow you to enter an email address and which will then generate a complex impenetrable Javascript program just for you which, when run by a visitor's web browser, will show your email address.

Why Upload?

You may be wondering why, if JavaScript is a client-side language that gets executed by the browser, we have to upload pages to the server before we can test them. Can you not just double-click on jstest.html from within Windows and get the browser to execute the local copy instead, and save yourself the hassle of uploading the file until it's ready for public consumption?

In theory, yes you can. You don't need to trouble your web server at all when you're developing JS pages. However, in practice, many web browsers don't like letting you do this. They regard it as a security risk, on the basis that any JS code in a local page must be malicious. Internet Explorer versions 7 and 8, for example, will pop up various warnings asking if you really want to allow this code to run. The only way to stop this is to reduce the security settings on your browser, which is not wise as it will also reduce your safety on the web in general.

Therefore, it's best to always run JavaScript code off a server. But it's important to realise that this is not because it's a requirement of the language.

Security and Cookies

Having a fully-functional programming language built into a web browser would be a serious security risk. Being able to find out the time on someone's PC is perfectly safe, but what if a malicious web site tried to read the user's private files instead, or write a virus to the computer? Thankfully, this is impossible, because JS is designed with security in mind. Most fundamentally, there are no facilities in the language that allows the programmer, and thus a malicious web site, to access any of the files on the visitor's computer.

Actually, there's one exception. Let me tell you about cookies.

A cookie is a small text file which a web site can write to the hard disk of any computer that visits the site, and then retrieve at a later date. It's part of the HTML specification, rather than anything to do with JavaScript particularly. But it needs a programming language such as JavaScript or PHP in order to operate.

Cookies are rather like pieces of electronic graffiti. They can contain any data which the web site operator wishes to store, but are limited to around 1000 characters each. A site can store more than one cookie on a visitor's computer. Crucially, for security purposes, the only site which can read a cookie is the site that put it there in the first place.

So what are cookies used for? Imagine a public web site where you're browsing a library catalogue in search of that excellent volume, "*Defeating The Hacker*" by Robert Schifreen. A couple of days later, you visit the site again and, as if by magic, you find yourself on the page about Defeating The Hacker. How did the web site know? Because last time you visited, the site stored a small cookie on your computer called "last-book-visited", the value of which was set to "defeating the hacker". Or, more likely, the value was set to the database record ID or the ISBN of the book, but that's not important right now. All you need to know is that each cookie has a name (in this case, last-book-visited) and a value (in this case, defeating the hacker).

Creating and reading a cookie isn't particularly difficult. There are commands built into JavaScript to do it, and it only takes a few lines of code. A Google search for "javascript set cookie" or "javascript read cookie" will show you some examples.

Unfortunately, reading cookies in JS is slightly more difficult than setting them. That's because there's no specific command in JavaScript that says "give me the value of the last-book-browsed cookie". Instead, there's only a command which says "give me a list of all my cookie names and values". Once you have this list, you then have to search through it in search of the cookie you want. But, as with everything web-related, a Google search will

bring you some ready-made JS code for searching for a specific cookie among the haystack of data.

One other thing worth knowing about cookies is that each one has an expiry date/time which is set when the cookie is created. When a cookie expires, it gets deleted from the computer. Any attempt to subsequently read its value will result in the return of nothing at all.

Expiry dates can be: a) a specific date and time, eg May 4th 2011; b) a duration, eg 31 days from now; or c) as soon as the user closes his or her browser. If you don't specify an expiry date when creating a cookie, option C is the default.

Cookies are a very useful feature, which most web developers use on their sites. It allows you to store information about particular visitors, without forcing them to create an account on your site and log in. But there's one golden rule about cookies, which you need to know if you're going to consider using them. Although the only web site that can read a cookie is the one which created it, any user of a PC who has administrator privileges can read and alter any cookie on his computer. That's because a cookie is nothing more than a small data file, albeit one with a strange name and often stored in a hard-to-find location on the hard disk. But it's a file nonetheless, and it can be accessed just like any other. This state of affairs allows for something called "cookie poisoning", which is where the user of a computer deliberately changes the value of a cookie.

Let's re-examine the earlier example, where a cookie called last-book-browsed is set to "defeating the hacker, by robert schifreen". With the necessary software on his computer, a visitor to the library site could change the value of the cookie to "defeating the afanc, by Rohan Stevenson". Having done so, next time he visits the library site he'll be taken to the page for that MP3 track, rather than the hacker book.

Does this matter? No, not in the slightest. But it neatly illustrates the golden rule about client-side web programming with JavaScript and HTML. Let's consider another example, to see why.

The library web site asks users to create an account and log in, if they wish to purchase books. The mechanism that they deploy in order to handle the authentication, ie the logging in, is as follows:

1. Ask the user for their username and password
2. If these details are correct, set a cookie on the user's computer called "is_logged_in" with a value of "YES".
3. Each time the user wishes to access a protected page, check the value of the is_logged_in cookie and only proceed if its value is YES.

A lot of web sites actually use a technique like this. But hopefully you can see the flaw. If a visitor poisons the is_logged_in cookie on his PC and changes the value from NO to YES, he can bypass all of the site security.

So, by all means use cookies for convenience. But don't use them for anything security-related, or for things where you need to be confident that the value of the cookie hasn't been changed without your knowledge. For such things as that, you need to use server-side programming instead, which is coming up later.

Morning All!

Here's another example Javascript program. To use it, copy and paste it into your jstest.html file in place of what's there already, using PSPad. But before you do, read through the code and see if you can work out what it does.

```
<HTML>
<body>
Good
<script type="text/javascript">
var d = new Date()
var time = d.getHours()
if (time < 12)
{
document.write(" morning ")
}
if (time >= 12)
{
document.write(" day ")
}
</script>
everyone!
</body>
</HTML>
```

The program displays a message, which is made up three words. The first word, part of the standard HTML component of the file, is "Good". The final word, in the same fashion, is "everyone!". In between, the second word is generated by some JavaScript and is either "day" or "morning", depending on whether the time on the visitor's PC is before or after mid-day. This is a simple technique, and one that you might want to use on your own pages.

Getting the Screen Size

Before we finish our brief foray into Javascript, here's one more example. It shows yet another reason why client-side programming, and being able to interrogate various features of the visitor's PC and browser, is so useful.

```
<HTML>
<body>
<script language="javascript">
var w = screen.width;
var h = screen.height;
if ((w=1024) and (h=768))
{
document.write("<img src=normal-pic.jpg>");
}
if ((w > 1280) and (h > 1024))
{
document.write("<img src=bigger-pic.jpg>");
}
</script>
</body>
</HTML>
```

This is the code for a simple web page which contains nothing more than a picture. The clever part, though, is the JavaScript code which looks up the screen width and height of the visitor's computer and then uses document.write to create HTML code on-the-fly. Specifically, it generates one of two different <img> tags. If the screen resolution is 1024 by 768, the normal-pic.jpg image gets loaded. But if the resolution is higher, the bigger-pic.jpg file is displayed instead.

Javascript Toolkits and Frameworks

If the idea of writing client-side code appeals, but you don't want to learn a whole new language, consider using a ready-made toolkit or framework. There are a number of such products around, that contain a large collection of ready-made Javascript effects that you can call upon within your web pages. One of the most popular is jQuery, which you can find at **www.jquery.com**. And the best news is that it's available completely free of charge.

The jQuery framework is simple to use. Just copy the single jquery.js file to your web server, then use a couple of lines of code in the <head> section of your HTML page to load it. From then on, each effect that you want to achieve on your web page is typically just a handful of

lines of special jQuery code. For example, here's the jQuery commands to make one paragraph of text reveal itself by sliding gracefully over another:

```
$("div.contentToChange p.firstparagraph:hidden").slideDown("slow");
$("div.contentToChange p.firstparagraph:visible").slideUp("slow");
```

There are lots more examples and tutorials on the jQuery web site, and it's definitely worth taking a look if you want to include visual effects on your web pages.

Incidentally, the above code makes reference to something called the DOM, or the Document Object Model, which is a way of referring to specific items on an HTML page. It's part of the HTML specification, rather than anything to do with jQuery. The DOM, and manipulating it with a technology called Ajax, isn't currently included in this book but may be added in the future. Contact me via **www.the-web-book.com** if you want it to be.

Finding Out More

As you are hopefully discovering, JavaScript is a very useful tool for anyone developing web pages. Sadly, though, not all of the language is as easy to use as the previous chapter might have led you to believe. A search in your local library or on web sites such as Amazon will lead you to some excellent books on JavaScript, some of which run to 1000 pages or so. It really is a very complex, full-featured language. But if you don't want to go to the trouble and expense of buying a book, a Google search for "javascript tutorial" will lead you to lots of useful online resources.

Because JS is all client-side, you can also look at other people's code too. Just use the View Source option in your browser, when looking at any page which you think might be using JavaScript, then search for a **<script>** tag. In fact, search for **<script** rather than **<script>**, because the "language=javascript part is actually optional (albeit bad practice) and this will find both cases.

There is also a wealth of ready-made JavaScript programs available for download from various web sites, many of which are free to use. One of my favourite sites is **www.hotscripts.com**, which has thousands of such programs available. The **www.dynamicdrive.com** site is another useful resource, which also offers lots of Javascript code for free download.

One of the most common uses for JavaScript is for navigation and menus. You've probably seen web sites with all sorts of interesting menu effects, such as drop-down menus that look like Windows applications, or menus that slide around the screen so that, no matter how you scroll the page, the menu is always in the middle. Almost all of these are done with

JavaScript, and sites like hotscripts will let you download the code to do it. Just follow the instructions that accompany the download, which are normally no more complicated than "copy this block of JavaScript into your HTML file and change the relevant text to customise the menu entries as you see fit".

Beware, though, of copying large chunks of code out of existing web pages unless you are certain that it is free to use. A company which pays thousands of pounds to have some custom menu code written for its site won't be too pleased if you start using it on yours too. Just because copying the code is technically possible doesn't make it legal.

Web-hosted Databases with MySQL

In the previous chapter we looked at client-side programming using Javascript, which runs in the user's web browser. In a later chapter we'll look at server-side programming with PHP. However, one of the most useful features of PHP is its ability to talk to server-based databases, so we first need to understand such things.

There are various server-based database products which store their information on a server rather than on the user's local PC. That server doesn't necessarily have to be a web server – the main point is that the database is held centrally and then accessed by multiple users, from multiple locations.

In the world of databases, the most popular language for interrogating a database is something called Structured Query Language, or SQL. Many database products use SQL, or something like it. Most SQL or SQL-like database server products cost serious money, such as Oracle or Microsoft SQL Server. One, though, is totally free of charge, and unsurprisingly this is the one that most web hosting companies install for your use. The product is called MySQL (pronounced my ess queue ell, or sometimes my sequel). Don't be put off by the price tag – MySQL is corporate-strength software, just as happy storing tens of millions of items as a mere handful.

In this chapter I'll show you how to manually connect to your web server via a browser and create a database. In the next chapter, you'll see how to access your database using PHP, which will finally allow you to create web-based applications and database-driven web sites.

You won't be able to try any of the examples that follow unless your web host offers MySQL. Almost all of them do, though some make a small additional charge to enable it. In the case of the hosting company that we're featuring throughout this book, hostmonster.com, MySQL is included.

Remember that MySQL is a server-based product, installed on your web server rather than your local PC. Equally, all of the data in its databases is stored on the server too. To access the system in order to create and browse databases, you use a web browser. In the case of many hosting companies, and hostmonster is no different, accessing MySQL requires you to access two different web sites. One location is for creating the initial databases. The other location is for managing those databases, which means: creating tables and fields; adding and deleting data; searching the database tables. The reason for this demarcation is because of security considerations.

Databases, Tables, Fields, Rows and Columns

At this point, you're probably getting confused by the mention of databases, tables, fields and so on. So before we continue, let's clear up the differences. It's vital that you understand some basic terminology in order to avoid confusion later on. Most importantly, what you probably think of as a database is actually referred to as a table.

A table is a collection of rows and columns. Think of it just like a spreadsheet. You might have a table called "contacts". The columns are headed "name", "address" and "phone_number". A row might consist of 3 fields (just like spreadsheet cells) containing the values "John Smith", "65 High Street, London", and "01234 567890".

A database is a collection of tables. Typically, those tables are all related but there's no rule that says they have to be. For example, let's imagine that you're creating a database application for a system that manages the running of a hotel. The database itself might be called "hms" (for hotel management system). Within the database might be 3 separate tables. One, called "customers", contains the contact details of all past, present and future guests. Another table, called "staff", might contain the contact and salary details for staff. A final table, called "rooms", might contain details of the bedrooms, with column headings such as room_number, occupied_by, bed_type, phone_number, needs_cleaning, date_last_occupied and so on.

Many hosting companies limit the number of databases that you can create. Typically, to between 1 and 5. However, this is not a problem as you are rarely limited in the number of tables that you can create within each database. Even if you only have one database, you can still create hundreds of tables in it. Obviously, being able to group your tables into related functions by using separate databases is handy, but it's not essential. If you only have the one database, and you want to create the hotel management system, the standard way of doing such things is to add a prefix to each of the related tables in order to indicate that they are linked. So we might choose an hms_ prefix for our hotel management system, and therefore name the tables hms_rooms, hms_guests, hms_staff and so on. Other tables that are connected to other systems would have their own prefix.

Normalization

While we're discussing MySQL terminology, there's one more thing that you need to know before we can start creating databases and tables for real. It's nothing to do with MySQL or web sites specifically, but is more to do with basic computer science and the creation of any computer database.

The golden rule of database design is: "don't store anything more than once. Instead, store it just once, give it a unique reference number, and refer to it by that number in future". Why? Because it's a much safer way to ensure that the database content remains accurate. If you need to change something, you only need to change it once, and all references to it are automatically up to date. Let me give you an example.

Consider our hotel management system. There's a table called "rooms", with columns called:

room_number
occupied_by
bed_type
phone_number
needs_cleaning
date_last_occupied

There's also a "customers" table, with columns called:

name
address
phone_number

Within the customers table is John Smith, and he's currently resident in room 225. How do we represent this in the database? The obvious answer is to add him to the customers table and then create a new row in the "rooms" table, as follows:

Room_number	Occupied_by	Bed_type	Phone_number	Needs_cleaning	Date_last_occupied
225	John Smith	Single	7225	NO	N/A

However, this breaks our golden rule of only storing something once. John Smith appears in both the "customers" table and the "rooms" table. It's a waste of space storing this information twice. Also, if John Smith calls the hotel and says that we've spelled his name wrong on his bill (it should be Jon), we'd have to search every table looking for instances to change.

There's one further problem, too, which is even more serious. What happens if a new guest, also called Jon Smith, checks into the hotel? There are now 2 Jon Smiths in the customers table, but there's no clue in the "rooms" table as to which of them is currently in room 225. Which will play havoc with the billing system.

So here's how we do it, according to the golden rule of giving everything a reference number and then referring to things by their number. We create the customers table like this:

id
name
address
phone_number

The difference is that we've added a reference number column. We've called it "id" rather than "reference number", simply because that's the convention that most MySQL database designers use, but it's not essential.

So now, Jon Smith's entry in the customers table might look like this:

Id	Name	Address	Phone_number
64	Jon Smith	65 High Street, London	01234 567890

And the reference to him in the "rooms" table looks like this:

Room number	Occupied by	Bed type	Phone number	Needs cleaning	Date last occupied
225	64	Single	7225	NO	N/A

No longer is room 225 occupied by Jon Smith. It's now occupied by guest number 64. And by checking the customers table and searching for the person whose id is 64, we know exactly which Jon Smith is in the room. Furthermore, if Jon changes his name back to John, all we need to change is the single reference in the customers table. All other references to him in the rooms table, the billing table, the restaurant bookings table, and so on, will only refer to him as customer 64 and not by name.

So now you know why large companies always want to know your customer reference number when you telephone them with an enquiry!

Designing databases in this way, ie with information referenced by a unique number, is known as normalization. It's often harder than it looks, but is well worth the effort invested. It ensures that your database, ie your collection of related tables, is working at peak efficiency.

As it happens, we've already missed a trick in the design of our hotel system. You'll see that there's a date_last_occupied field in the rooms table. But why do we need this? Presuably, hotel bookings will be held in a "bookings" table which has columns called id, customer, arrival date, departure date and room number. So in order to find out which guest last occupied a room, we don't actually have to store any data at all. We just search the bookings

table instead. It's this sort of planning that takes just seconds to think about when you're designing a system, but days or weeks to implement if you discover, towards the end of the project, that you've been working with an inherent design flaw. So, always give lengthy consideration to your database design before you start writing any programs.

MySQL is known as a relational database. Hopefully you can now see why. Within the database is a collection of tables, and the information in those tables is often related. In the case of our rooms table, for example, there's a relationship with the guests table. We'll return to this subject when we come to create further database tables for our hotel management system in future chapters.

Referential Integrity

A normalised database, where each type of data item is held in a separate table with a unique id, makes a great deal of sense. It's by far the most efficient way to store and manage information, whether you're designing a booking system for a 10-room hotel or an inventory manager for a web site the size of Amazon. However, it's important to ensure that you don't end up with tables that refer to non-existing information. In database terminology, this means making sure that you maintain the referential integrity.

Consider the example above, where room 225 is currently occupied by guest number 64. Five years from now, you notice that Jon Smith has stopped visiting your hotel so you decide to remove him from the customer database. Or perhaps he calls the hotel and asks to be taken off the mailing list, and you do this by simply removing his entry from the customers table.

As far as the customers table is concerned, there's no problem. Where once was record number 64, now there's nothing. But customer 64 is still referred to in the bookings table, and probably some other tables too. Which is going to cause problems. For example, when you run the "show me everyone who's stayed in room 225 over the past 2 years" report, and the system tries to find out the name and address of the person whose customer id is 64, there will be nothing on file. The integrity of the relational database has failed.

It's important that you handle such instances, and that you handle them correctly. In the case of the deleted customer, you can do one of three things:

1. Delete the customer from the table. In addition, also delete every affected row in every other table. Which in this case would mean deleting all of customer 64's room bookings, restaurant reservations, customer comments, and whatever other tables have a customer_id reference. The problem here is that you lose valuable data.

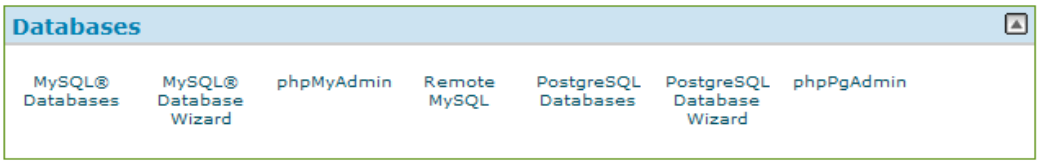
2. Replace the contents of customer's record with text such as "deleted". Leave the id number in place, but change the firstname, surname, phone number, address etc. Now, all your reports will still work correctly but, as per Mr Smith's wishes, his information has been deleted from your system.

3. Add a field to the customers table called, say, is_live, which specifies whether this is a live customer record or not. Keep it set to Y for all current customer records. To delete a customer, you don't remove anything from the customers table. Instead, you just set the customer's is_live flag from Y to N. Now, in the code that produces your reports, you check whether a customer is live or not and, if not, you display "non-live customer" rather than the real data. The old data is still there, but you simply choose not to display it. Bear in mind, though, that this can sometimes fall foul of data protection legislation in some countries, which states that you should delete personal customer information if you no longer need it or if the customer asks you to. Merely marking it as non-live doesn't count as having deleted it.

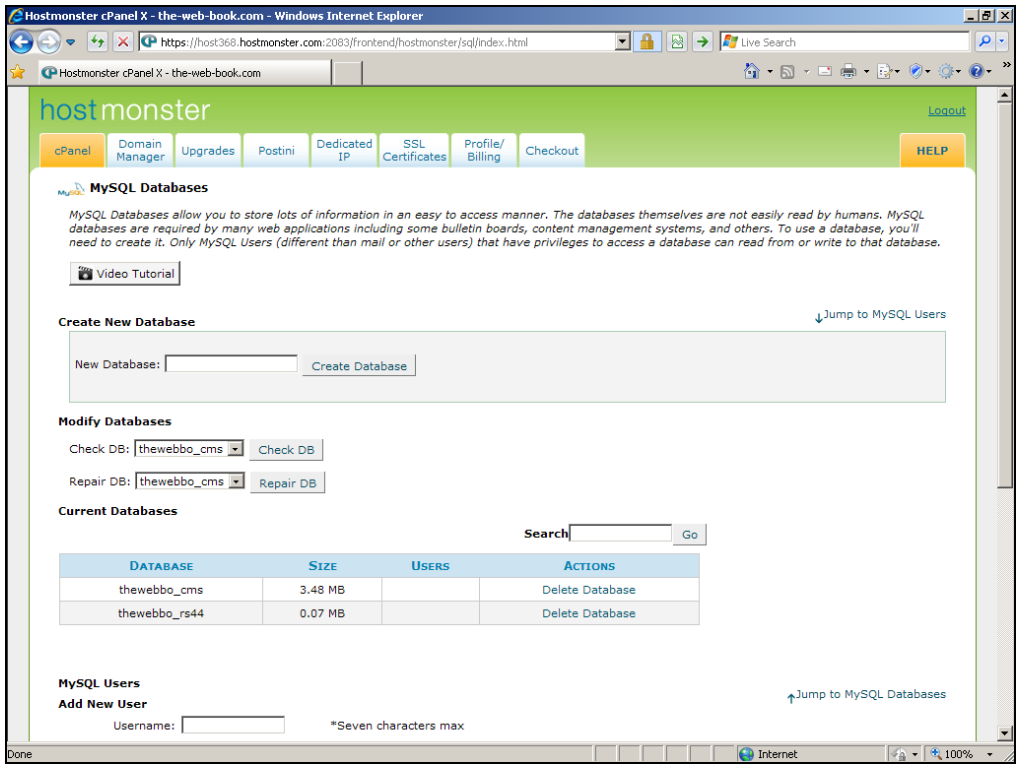
Creating A Database

Now that you understand the basics of relational database design, we can log onto the web server and actually create a database. Throughout the rest of this book, in the chapters that discuss MySQL and PHP, we'll continue building and enhancing HMS, our Hotel Management System, as our example database application. If you have a different project in mind, feel free to adapt the examples as you go. However, you will probably find it easier to start by following the examples as they appear. Once you're happy that the HMS system is working properly on your server, you can then adapt it into whatever you want.

To create a MySQL database, you'll need to find out how this is done on the particular hosting platform that you're using. In the case of Hostmonster, it's done through the Databases section of the control panel. So we start by logging into the control panel at hostmonster.com, with our username of thewebbo and the relevant password. This brings up the control panel, the relevant part of which is:

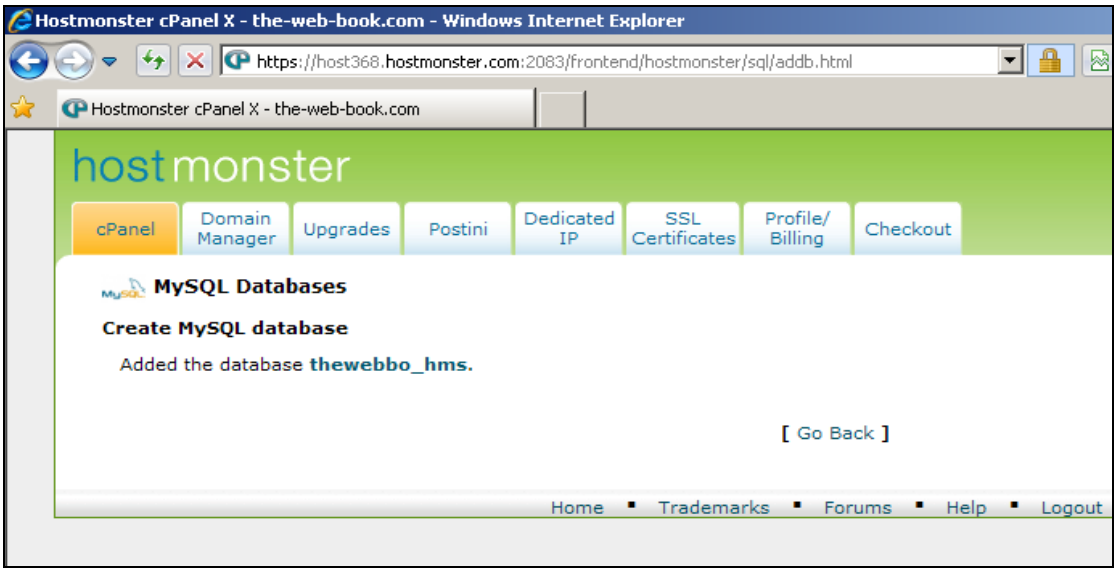


To create a database, click on the MySQL Databases link. You'll see the following screen:



As you can see, I currently have two databases created (thewebbo_cms and thewebbo_rs44). Hostmonster adds a prefix to its databases, based on the username of the hosting account they belong to, hence thewebbo_ at the start of each database name. The thewebbo_cms database is, if you have been reading this book from the start, the one we used in the earlier chapters when we installed Joomla, WordPress, phpBB and Plogger. Using table prefixes, we installed all of the products into the same database.

We need to create a new database for our hotel management system. We'll call it hms for simplicity. So, in the Create New Database section, type hms and click Create Database. You'll see a confirmation message:



And that's all there is to it. But note that the database name is thewebbo_hms rather than plain hms, for reasons stated earlier. You'll need to know the name of your database later, when we start to write PHP programs that access it.

With the database created, we have no further need for this part of the control panel. So click the Home link at the bottom of the screen to return to the main control panel screen and the full menu of database-related things.

The third option says "phpMyAdmin". This is a web-based program for managing MySQL databases on a remote server. It's a free, open source application that just about every web hosting company offers for use by customers (ie you, rather than the people who visit your web site). It lets you create tables within databases, and it lets you create and manage the columns and rows within those tables. The only thing that it can't do (at least, not in a way that satisfies the security requirements of most hosting companies) is to create the databases themselves, hence the previous step.

From now on we'll use phpMyAdmin to manage our new thewebbo_hms database and to create the tables in it. Later on, we'll write some PHP code which will use those tables to create a simple web-based hotel management application.

You'll be using phpMyAdmin extensively while you develop web sites, so it's a good idea to grab its URL from the control panel and save it as a shortcut or favourite. You'll still need to enter your control panel username and password each time you access it, but at least you'll be able to go straight to the program without first logging in to the control panel. To do this, just surf to phpMyAdmin as normal and then add the address to your web browser's favourites. Or copy the URL, and paste it into whatever web bookmark manager you prefer to use. Or you could even save it as a shortcut on your desktop, for easy instant access.

Port Problems

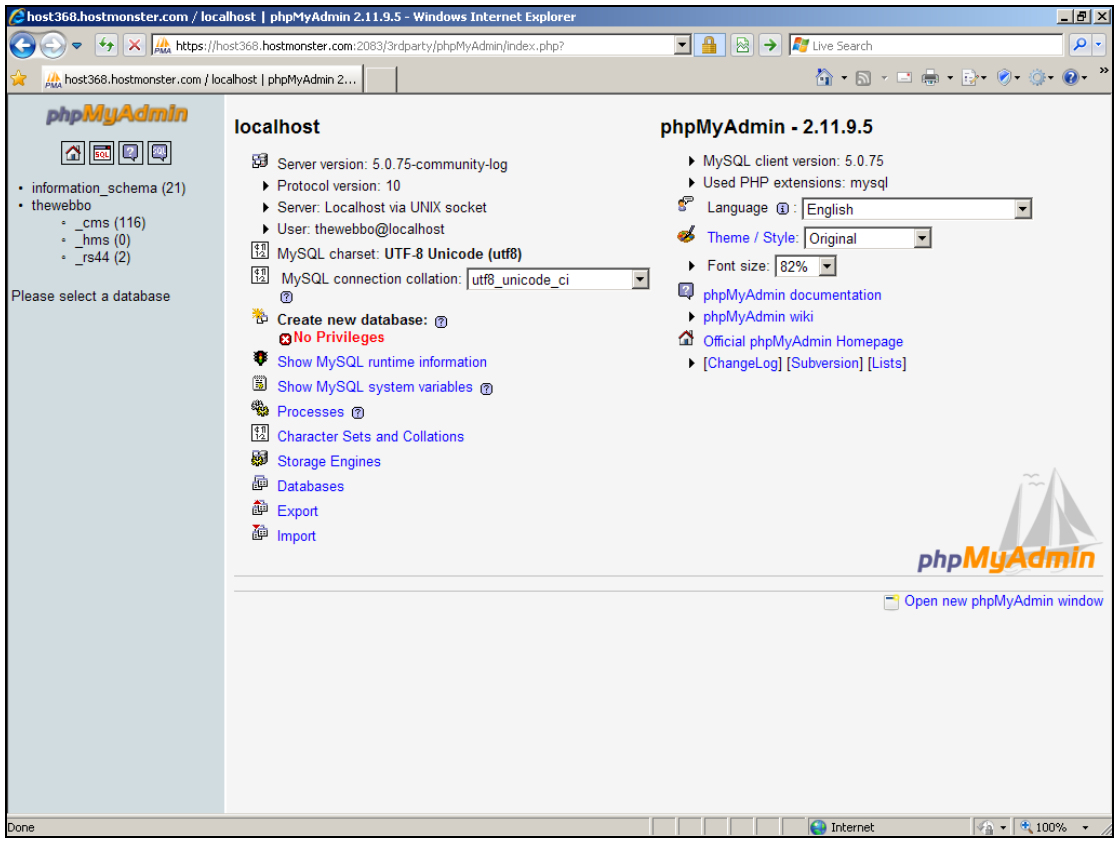
In theory, because phpMyAdmin is web-based, you should be able to reach it from any computer that has internet access. However, in common with many other hosting companies, hostmonster runs its installations of phpMyAdmin on a non-standard port.

Data travels across the internet using a system called TCP/IP, and information travels on different channels. These are known as ports, and are rather like radio frequencies or TV channels. Web traffic (ie, information that travels between web servers and the people who view the pages that are stored on them) generally uses port 80. FTP uses ports 20 and 21. Email generally uses port 25, and so on.

Most firewalls always allow traffic on these "common ports" to flow freely. But they also often block traffic on uncommon ports. So if your host runs phpMyAdmin on port 2083, for example, as Hostmonster does, then it's possible that it will be blocked. This doesn't mean that any of your visitors will ever have problems accessing your web sites because they won't be using phpMyAdmin at all (at least, you'd better hope not!). But it does mean that, for example, if you want to manage your databases from the office, or an internet café, or your school or college, you might not be able to. In which case you'll need to wait till you get home, unless you can persuade the person who manages your firewall to open the required port for you.

Using phpMyAdmin

Click on your new shortcut, or go via the host control panel if you wish, and you'll see the phpMyAdmin home page which looks like this:



Depending on which version of phpMyAdmin your hosting company runs, and the security permissions that it applies to your account, the screen may look slightly different, but the basic structure will be the same.

On the left hand side of the screen, in the blue panel, are your databases. In the example above, there are 4. One is called information_schema, and the number in brackets tells you that it contains 21 tables. There are also databases called thewebbo_cms, thewebbo_hms, and thewebbo_rs44. In this particular case, the _rs44 one is something that I have been testing, so you can ignore it. The _cms one is a remnant of our earlier experiments with Joomla, Plogger, phpBB and the like. The _hms one is what we're interested in right now, as this is what we'll be using throughout the rest of this book as we build our rudimentary hotel management system.

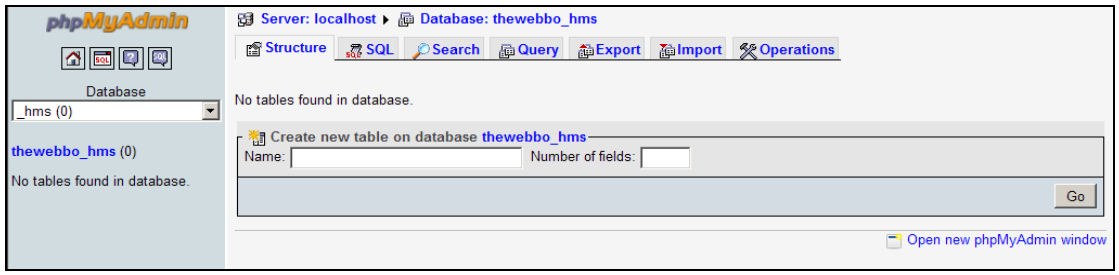
The information_schema database is part of the inner workings of MySQL on your hosting account. Some hosts allow you to see it, whereas others don't. If your host does, feel free to browse around it if you're curious, but don't change or delete any of the information in it. If you do, you face the serious risk of breaking your hosting account or at least your ability to use MySQL.

Creating The Customers Table

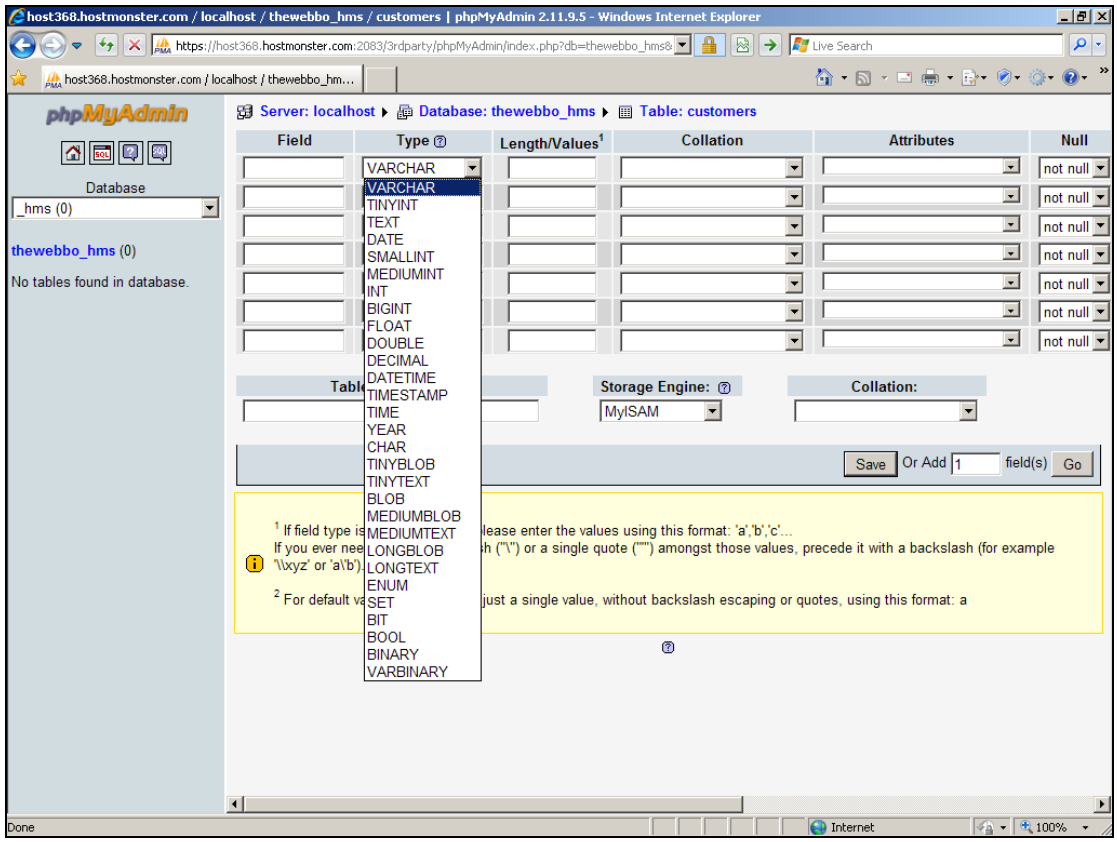
The first stage in building our hotel management system is to create the "customers" table. Once we've done this, we'll then use PHP to create a web site that allows hotel staff to create new customers and add them to the database. We can then use those customers to create bookings, and slowly build up our system.

Although it's possible to write PHP code which creates database tables (assuming the hosting company doesn't specifically block you from doing so), most web developers don't bother. Creating the tables for your database is something that you'll only ever do once, so you might as well let phpMyAdmin do it for you. If you make a mistake, or if you subsequently want to add new columns to the table (eg, you want to store a customer's car registration number), phpMyAdmin allows that too.

Fire up phpMyAdmin and get to the front screen, as shown above. Click on the _hms entry in the left hand panel, to view the thewebbo_hms database. There are currently no tables in the hms database so the screen will look like this:



We're interested in the "create new table on database thewebbo_hms" bit. Type the name of the table (customers) into the box. We'll start with 7 fields, so enter that number into the box and click the Go button. You'll see the following:



You now have 7 lines (one for each field) into which you can enter the field name, its type, its length, and a few other things too.

As you can see from the list of possible field types that I've clicked on in the top field, there are lots to choose from. It's important that you choose the right field type for each field (column) in your table. This generally means making two decisions. First, whether or not the content of that field will be purely numeric. Because if it will be, choosing one of the numeric field types will result in a database that can be searched and indexed more efficiently. The second decision is for when you decide that the field won't be purely numeric, but will also hold alphabetical text. In which case you need to specify the maximum length of any one field. I'll explain more about this as we create the customers table.

The field names are going to be:

id
firstname
surname
title
address
phone

email

So type those into the first column of each row on the screen. Note that we use all lower-case letters. That way, we won't have to worry about remembering whether a field is called FirstName, firstname, Firstname or FIRSTNAME when we're trying to access it later on.

Before we choose the "type" for each field, it's important to consider for a moment our choice of fields.

The "id" field exists because of our desire for normalization, as mentioned on page 183. Each time we refer to a customer within any table other than the "customers" one, we'll do so by id number rather than name.

You'll see that we've split the customer's name into 3 fields, namely firstname, surname and title. So Mr John Smith will actually be stored as a firstname of John, a surname of Smith, and a title of Mr. Why do we do it like this? Because, thinking ahead, we might want to use our hotel management system to create mailshots, and we can easily construct the first line of the mailshot by printing the word "Dear" followed by the customer's title and then their surname.

The address, phone and email fields are fairly obvious. Although, if you were creating this project "for real", you might want to consider splitting the address into 4 or 5 separate fields, and perhaps a separate one for the postcode or zip code. Why? Because you might want to create reports or mailshots based on specific postcodes, and it's much easier to search a database table by one specific field (eg, show me all the customers whose postcode starts with TN) rather than searching within a larger non-specific field (eg, show me all the customers where the letters TN appear somewhere within the address field).

Don't underestimate the value of thinking ahead in this way. When you're planning any database, the more thought and consideration you give to the way that your data is stored BEFORE you start writing code and firing up phpMyAdmin, the easier the task of creating the system will be. The decision to store a customer name in 3 separate fields is trivial to implement at this stage. If you were to go ahead with a single field called "name", then write half the system, and then decide that actually it would be better to store the name in 3 parts, it might take many days or even weeks to rewrite all the relevant parts of the system.

Anyway, back to our field creation screen. Hopefully you've typed in the 7 field names. Now to choose their types.

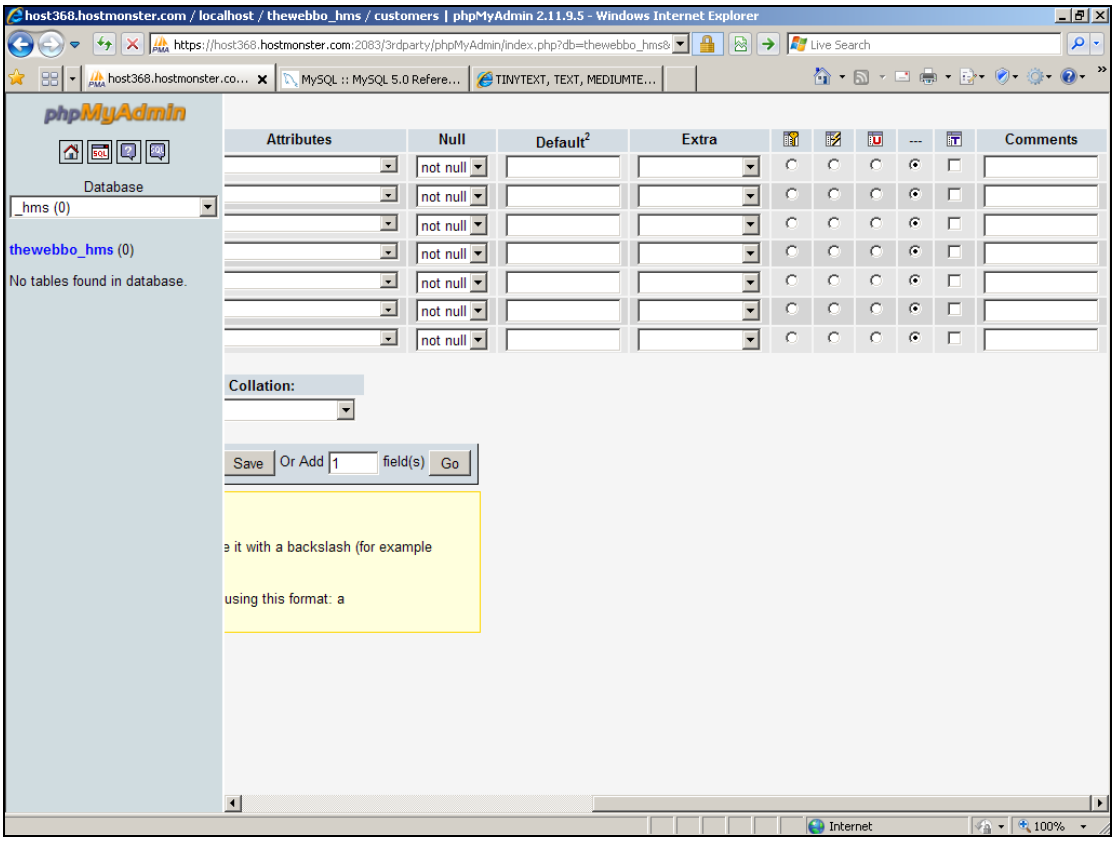
For the id field, choose MEDIUMINT. An INT field stores an integer, ie a whole number. You can't store text in there, and you can't store decimal numbers such as 3.5. But we'll use whole numbers for customer id's so that's fine. Why medium rather than a TINYINT or

BIGINT or plain INT? Because each can store a different range of numbers. TINYINT can only store numbers between 1 and 255, which clearly isn't enough to give every member of a hotel's customer database a unique number. MEDIUMINT stores up to 16 million (or -8m to +8m if you're going to want negative numbers in there too, which we don't). And that's plenty for us. If we need to change it later, we can do so without losing data.

Firstname, surname and title can all be TINYTEXT, which allows up to 255 characters of text. The address might conceivably be too long for this, so we'll use a TEXT type, which allows up to 65,000 characters or so. Not that we'll ever use that many, and nor will MySQL require 65,000 characters of disk space for every field in the database. But the space is there for the database to be sufficiently expanded should we use it.

TINYTEXT will also suffice for the phone and email fields.

There's one more thing we need to do. Scroll the screen sideways and you'll see some more options for each field:



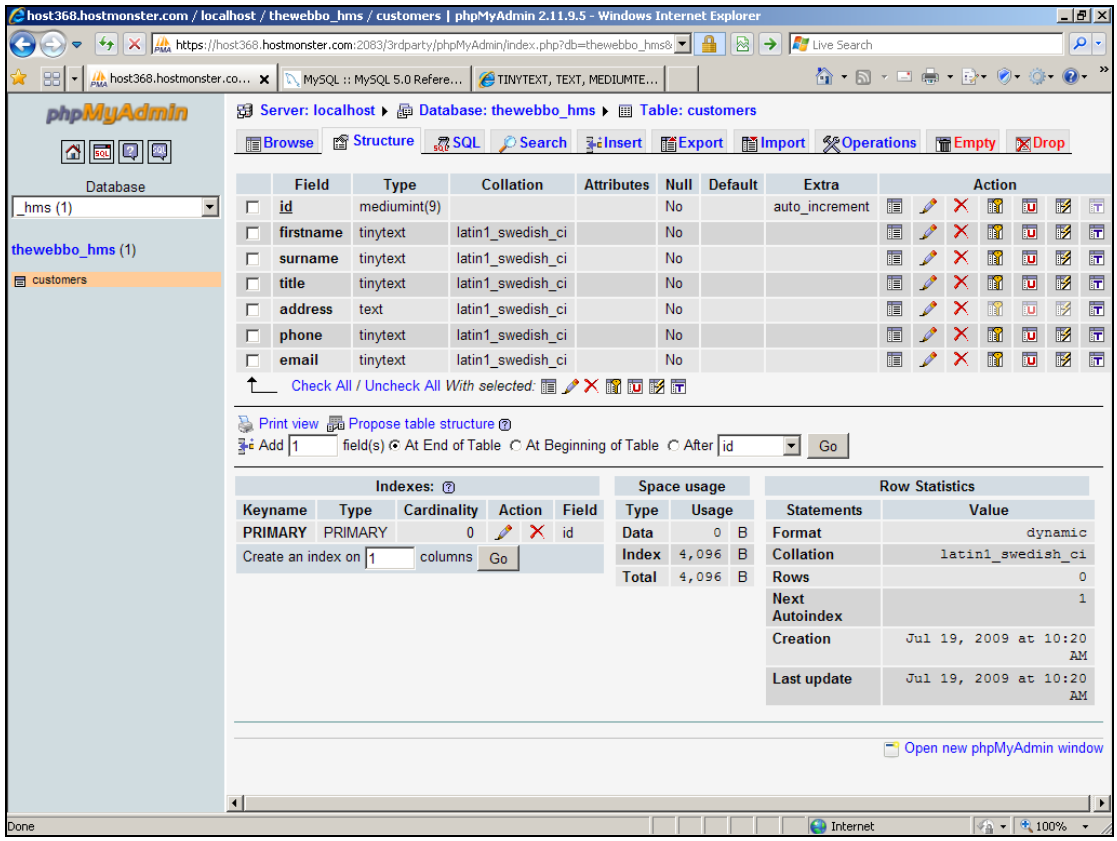
In the "Extra" entry for the id field, and *only for this field*, choose the auto_increment option. Each time you create a new row in the table, ie a new customer record, MySQL will

automatically generate a new unique id number. It saves you having to manually write code to look for an unused id number.

As for all the other options, such as Attributes, Null, Default and so on, you can leave these alone. They are not required here. Typically, all you need is a field name, a field type, and possibly an auto_increment on your id field.

When you're done, click on the Save button and you should see a summary of the table that has been created. That's it. The job is done.

In the left hand panel, click on the database name (thewebbo_hms in this example). You will now see a list of all the tables in that database, of which there is currently just one. So click on "customers" and you'll see the structure of the table like this:



Get used to this screen, as you'll be using it a lot when you start to become a regular user of phpMyAdmin. The "structure" tab shows you the field names and types. The other tabs that we'll use later are "browse" and "insert", which let you look at the data and create new rows.

Before we go any further, there are two fundamental things that you need to make sure you understand. First, the database and all its data resides on the web server. There's nothing

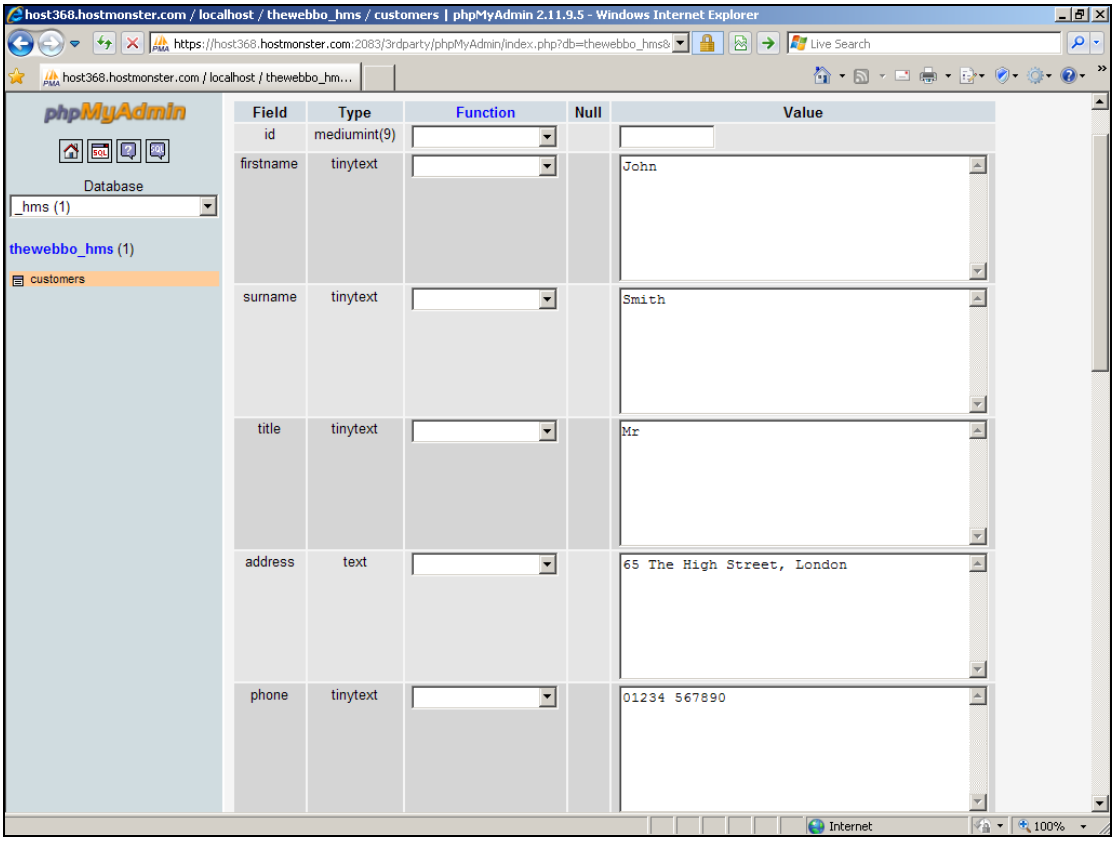
on your computer. So the data is accessible, via the web, from any computer that has internet access.

Second, everything we're doing so far is concerned with setting up the basic database and structure. We're using phpMyAdmin to do so because it makes the job easy. The visitors to our site will never see phpMyAdmin, or any of the screens that we've used so far. They will interact with the database using some much friendlier screens, which will comprise the hotel management system proper. The only reason we're not using those screens is because we haven't written them yet! That's where PHP will come in, later on.

Inserting Some Data

The customers table is now created. Let's use phpMyAdmin to insert some customer records into it. Later, when we've written the necessary PHP code to implement this part of our hotel management system, we'll be able to do it that way. But for now, this private "back door" route via phpMyAdmin will suffice.

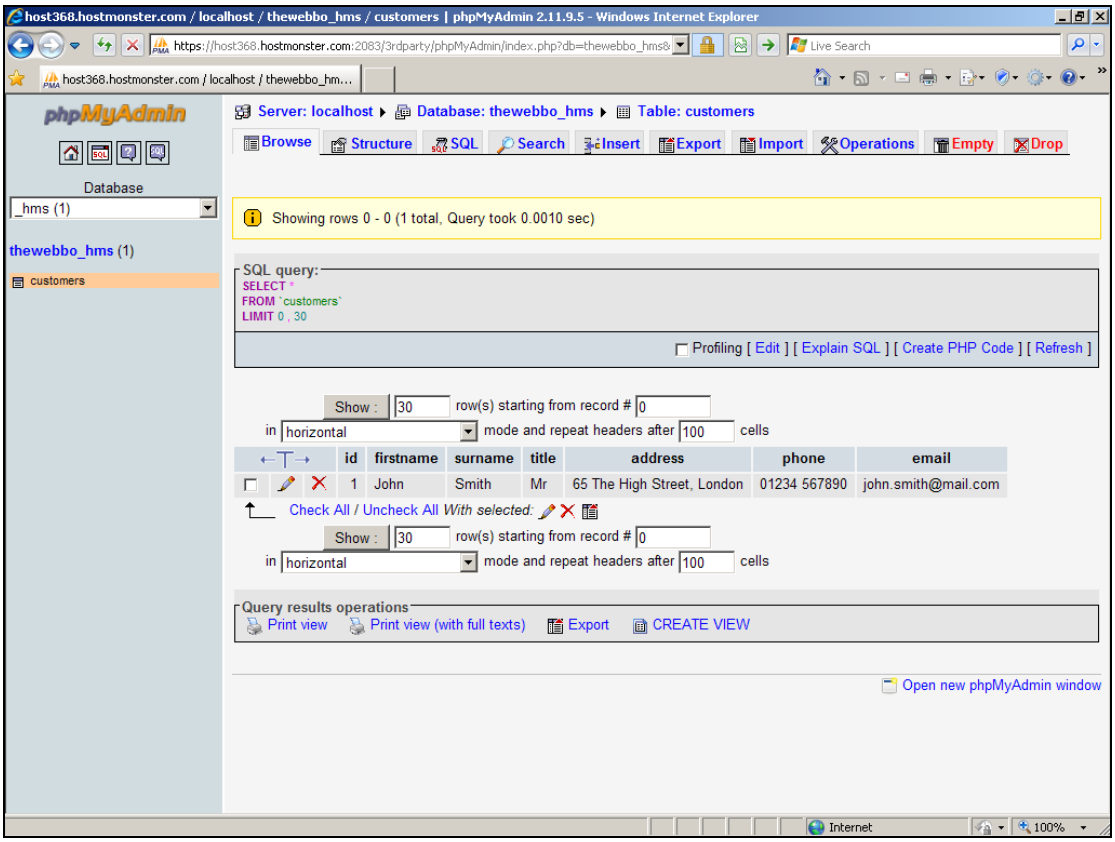
In the left hand panel, click your database and table. Then, in the top row of tabs, click on Insert and the screen will look like this:



Fill in the right-hand boxes for each field, as shown above. Ignore the "function" boxes as these aren't required.

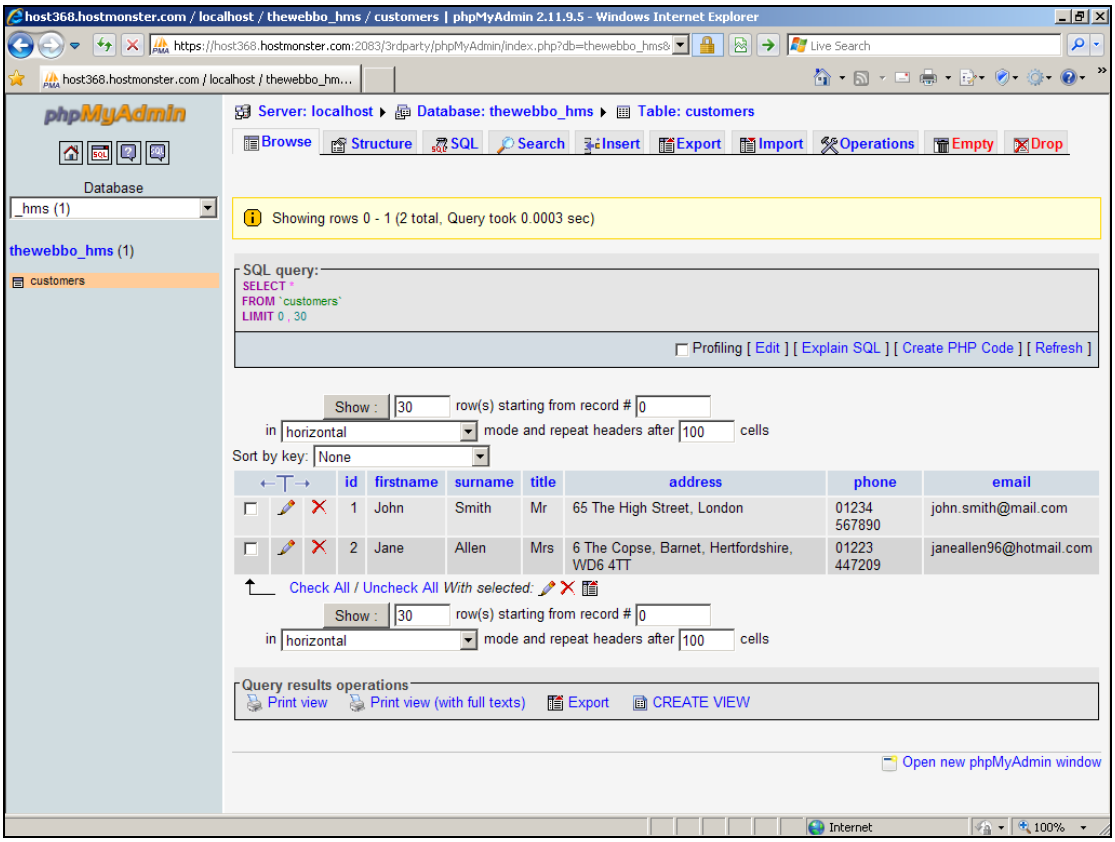
Note how I haven't filled in the id box. This is an auto-increment field, so MySQL should take care of this. We'll find out, in a moment, whether it does.

When you've completed filling in the fields, click the Go button underneath the final field. Then click the "browse" tab at the top of the screen, and you should see this:



It worked. In the centre of the screen you can see the row of data that you entered into the database. John Smith's information, complete with an automatically-generated id of 1, is there.

Before we declare this task finished, let's add another row. Click the "Insert" tab again, and create another customer. Then click Browse to ensure that it's been inserted correctly, as is shown here:



Our two rows have been given id's of 1 and 2. This is as you might expect. However, the MySQL database system makes no guarantees about the pattern of automatically-generated numbers, except that they will be integers and they will be unique. For example, if we have 3 rows in the table, with id's of 1,2 and 3, and then we delete the second entry, the two remaining entries will have id's of 1 and 3. When we insert another row, will it be given an id of 2? Or will it be allocated a 4, on the basis that this is the lowest number that hasn't yet been used?

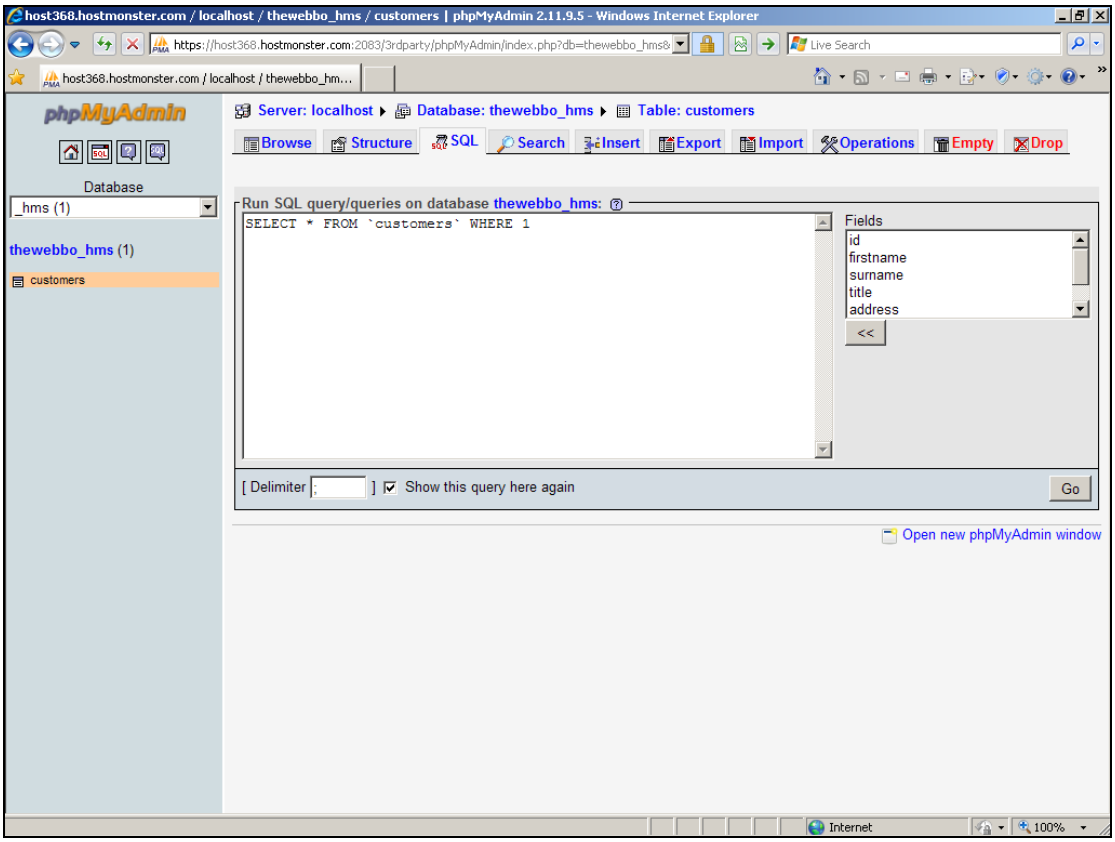
The answer is: no one knows. Either is a possibility, because "1,2,3" and "1,2,4" both fit the rule of being integers and unique. Sure, there's a gap in the second set, but MySQL never guarantees that there won't be gaps. This is important because, for example, you might want to produce a report which tells the user how many customers are on file in the database. Clearly you can't do it by searching for the highest-numbered id. Instead, you have to ask MySQL to tell you explicitly how many rows there are in the table. Not difficult, so long as you know what question to ask.

Querying the Customers Table

Before we leave phpMyAdmin and the creation of MySQL databases and tables, there's one more topic that we need to cover. This is actually crucially important, although it might not seem so right now.

What we're about to do is to query the customers table in order that MySQL will return the data we ask for. The reason that this concept is so important, is because the techniques we use here are the same ones we'll use later within our PHP code.

Click on the SQL tab and you'll see the following:

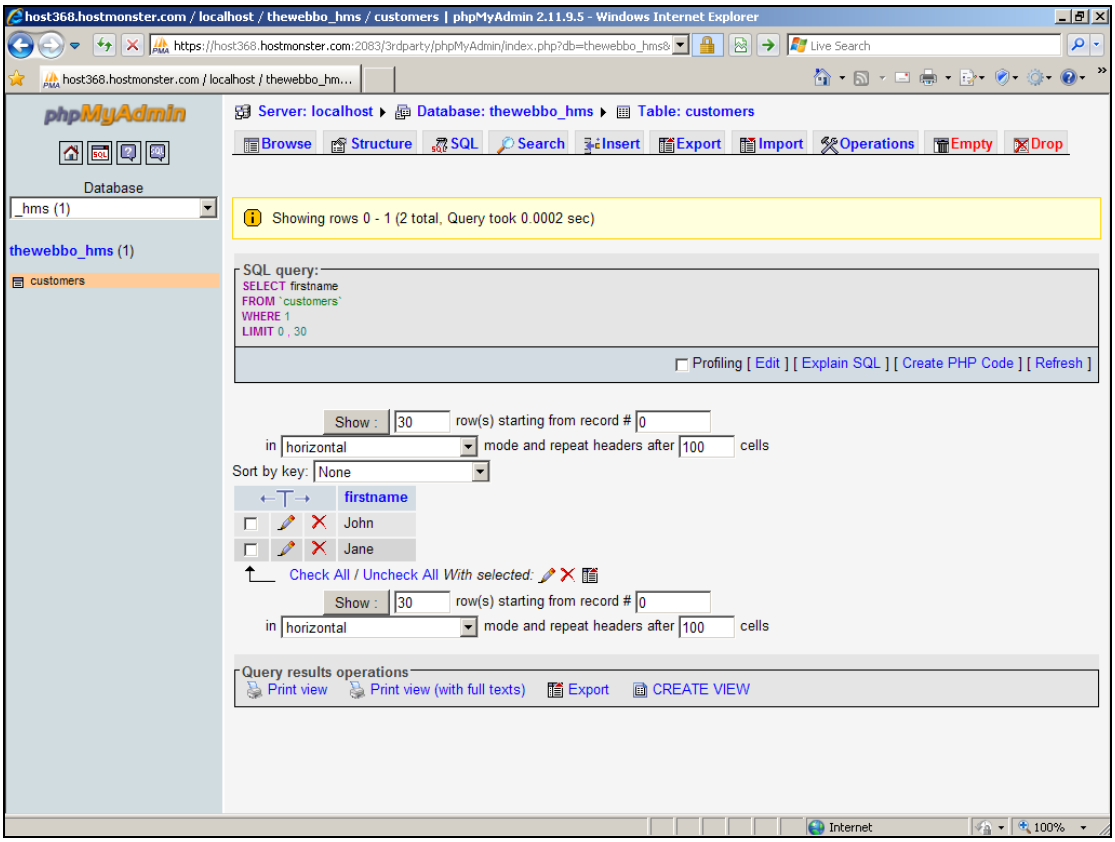


SQL stands for Structured Query Language. It's an entire computer language that was specifically invented for querying databases using an English-like syntax.

In the query box which currently says SELECT * FROM `customers` WHERE 1, change this to:

```
select firstname from customers
```

Then press the Go button and you should see:



Congratulations, you've just executed your first SQL query.

`select firstname from customers` should be fairly self-explanatory. You've told MySQL to select a specific field (firstname) from a specific table (customers). As you can see from the resulting screen, this is exactly what it has done. You now have a single column of results, namely all of the first names (John and Jane).

There's no guarantee as to which order the data will come out. Sometimes, as here, the names will appear in the order that they were originally entered, but this should not be relied upon. If you want to specify an order, you need to use the ORDER BY command as part of the query. Like this:

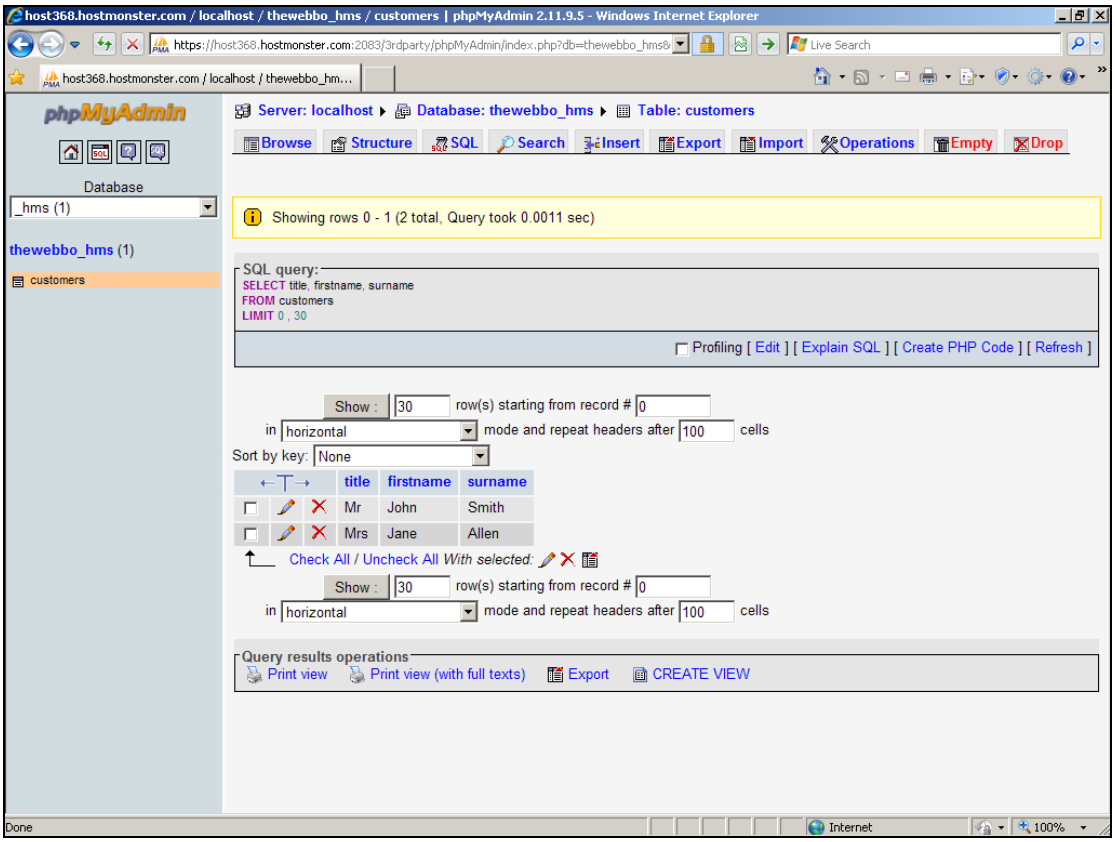
```
select firstname from customers order by firstname
```

So now our SQL command tells the database what field we want (SELECT), from what table (FROM), and how to sort the results (ORDER BY). In this case, we want to order by the firstname field. We could just as easily have used "order by email" or "order by surname" if we wanted to sort them in that way.

Let's retrieve the full names (title, firstname and surname) of all our customers, using:

```
SELECT title,firstname,surname FROM customers
```

And here's what comes back:

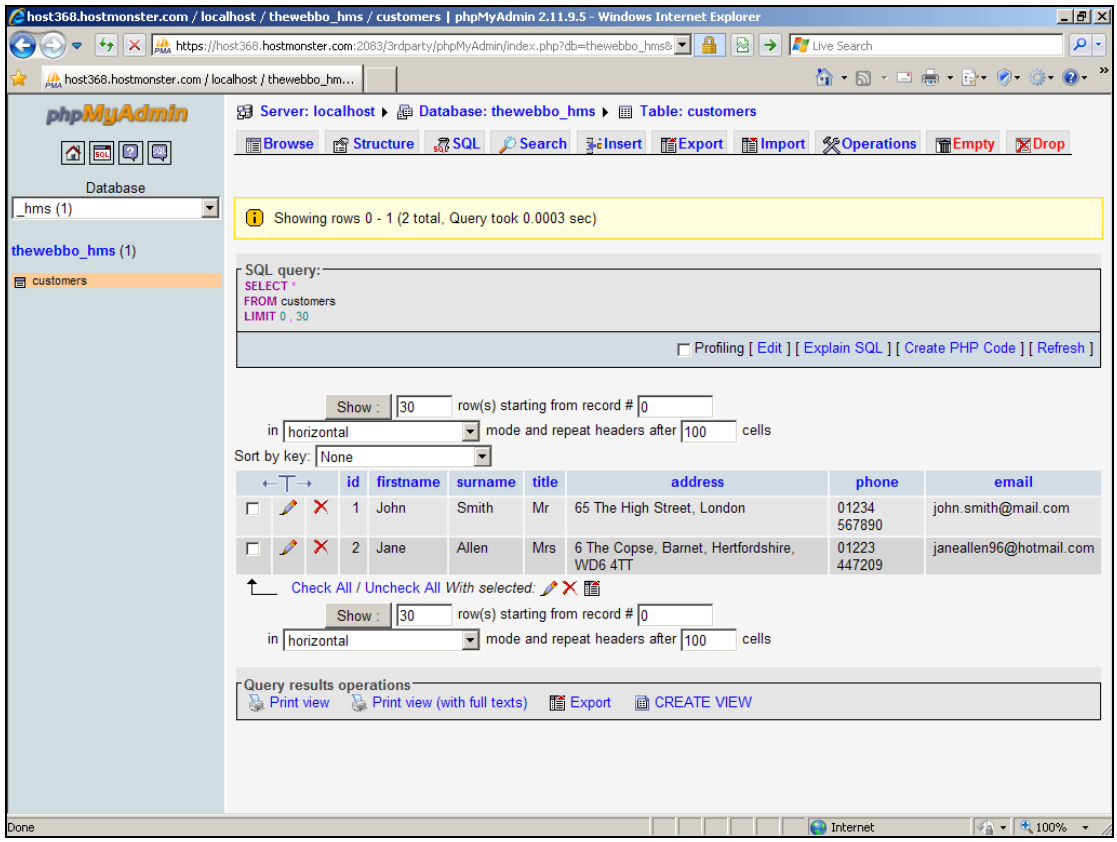


Note how we specify the fields we want to return, in the order we want them. So now we can see that we have 2 customers, namely Mr John Smith and Mrs Jane Allen. Not quite a proper hotel management system, but at least we have a database that we can soon start to access via PHP.

If you want to select all of the fields in a row, just use an asterisk like this:

```
SELECT * FROM customers
```

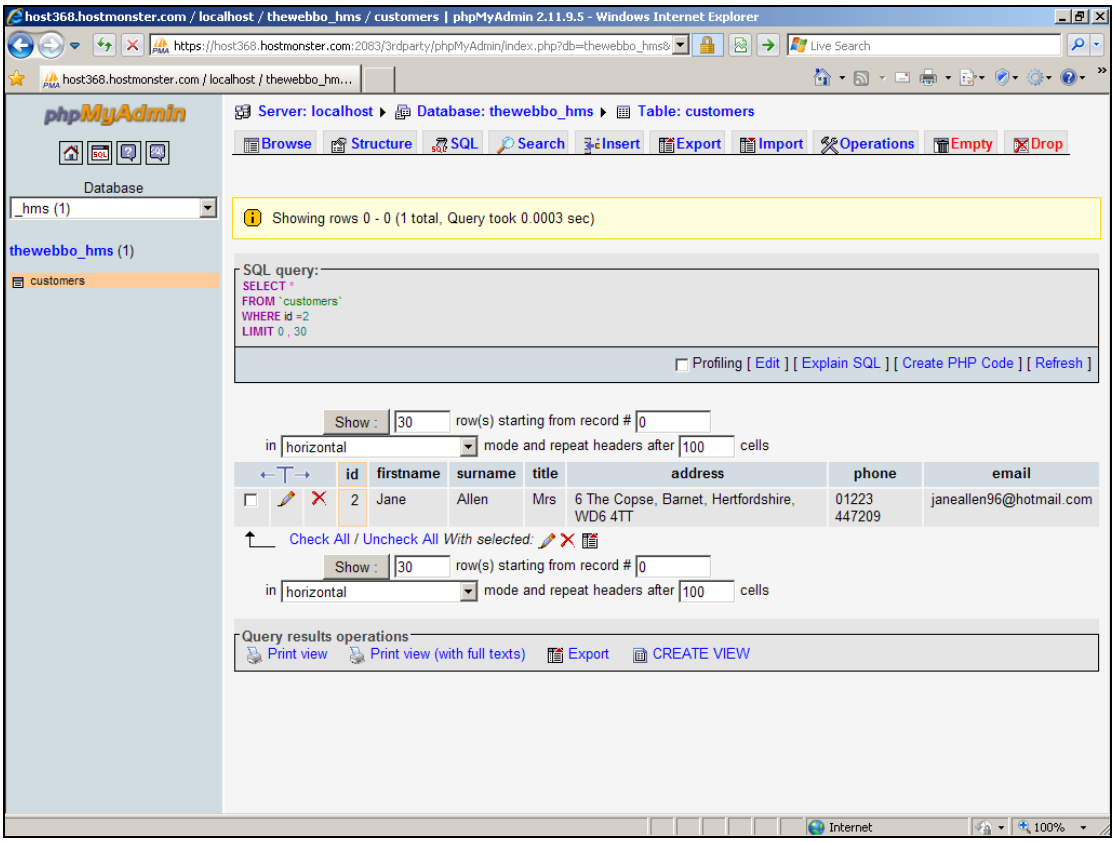
Which will produce:



The final component of the SELECT command that you need to know is WHERE. Have a look at the following MySQL command:

```
SELECT * FROM `customers` WHERE id = 2
```

As you might expect (try it!), it returns the following:

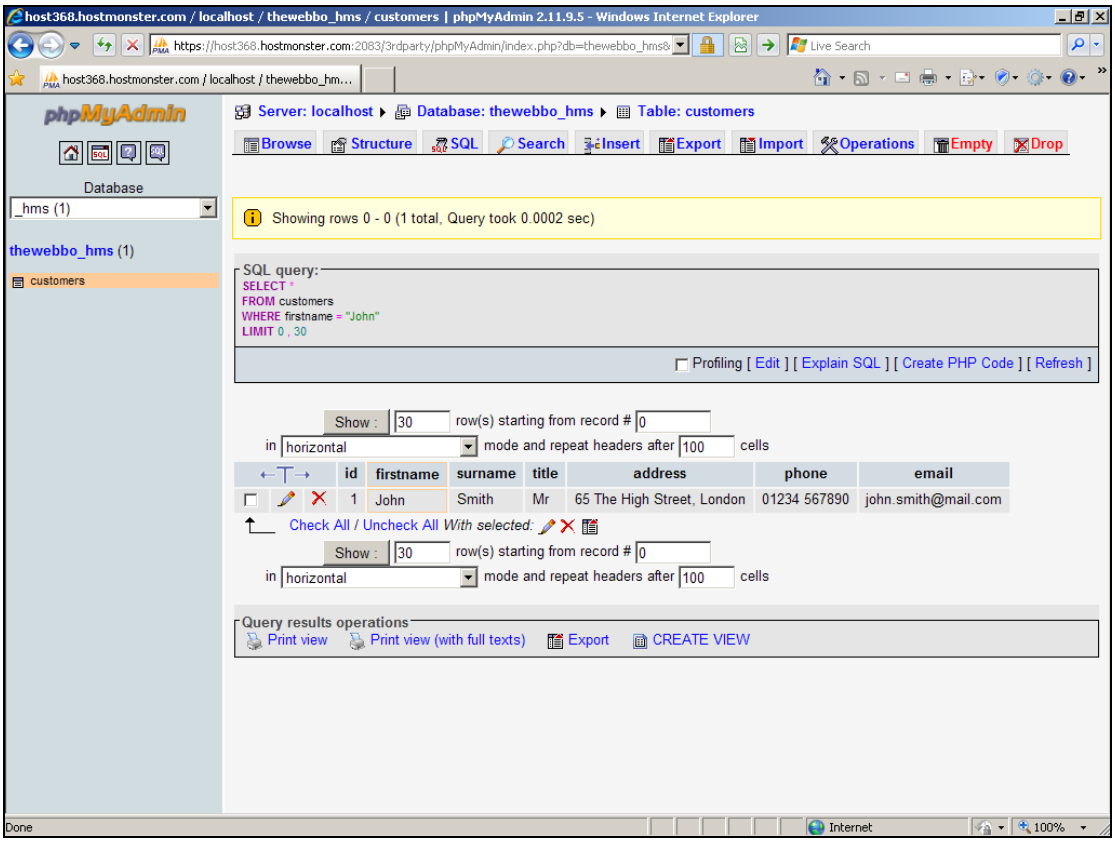


It brings back the row whose id is 2.

Now try:

```
SELECT * FROM customers WHERE firstname="John"
```

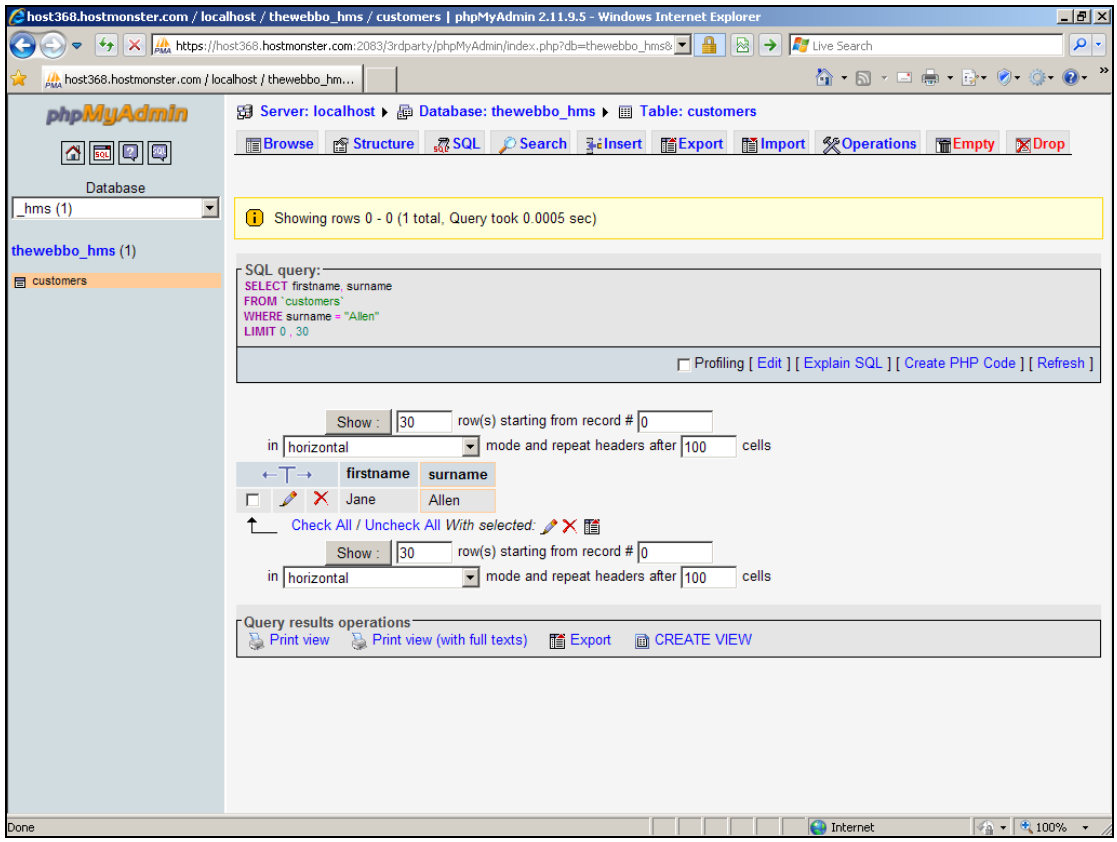
Here's what gets returned. As you might expect, it's John's record:



One more example:

```
SELECT firstname,surname FROM customers WHERE surname="Allen"
```

And here, as you have probably worked out by now, is what comes back:



Note how "Allen" and "John" need quotes around them when used as part of a WHERE clause, whereas 1 (as an id) doesn't. That's because firstname and surname are text fields. The id field is a numeric field, which doesn't. If you search for the record which has an id of "1" rather than an id of 1, nothing will be found.

For this reason, some MySQL programmers like to include a clue as to the type of a field, within the name of the field itself. For example, instead of creating a field called firstname, they'll create one called tt_firstname, where tt stands for TinyText. And instead of id they'll use int_id. That way, it's pretty obvious that:

```
select * from customers where int_id = "72"
```

isn't going to return any results, because int_id is a numeric field and so the 72 shouldn't be in quote marks.

Whether you adopt this style is entirely up to you. Personally, I don't. Instead, I try to ensure that I use text fields for everything except id's. Then, I know that every field name except id needs to be in quotes.

So now you know the basics of the SELECT command, to return one or more rows from a table based on a particular criterion. As you might expect, the SELECT command is much more powerful than this, and the official documentation for it runs to dozens of pages. But this will do for now.

With this basic knowledge of MySQL and the SELECT command, it's time to move on to the most advanced section of this book, namely the PHP programming language. Which, as you're about to see, makes it possible to do just about anything on a web page. If you've ever wondered how sites such as eBay and Amazon are created, or programs such as Joomla, you're about to find out.

Don't delete your hms database from the server, by the way. You'll need it later on.

Introducing PHP

On page 166 I talked about Javascript. This is a client-side language that allow you to write programs which get run in, and by, the web browser of someone who visits your site. It's now time to leave client-side languages behind, and talk about server-side programming instead.

Like Javascript, server-side languages allow you to write programs that add features and functionality to web sites. But, as the name should hopefully suggest, the big difference is that server-side programs are run by, and on, the web server itself. The visitor's web browser doesn't get involved, except to display the output of the program.

There are many server-side programming languages around, all of which do pretty much the same thing. The one you decide to use will mostly depend on what you feel most comfortable with, and what's offered by the hosting company that runs your server. Unless you run your own web server you'll be limited in the server-side languages that are available to you, as hosting companies won't allow you to install your own.

Among the most popular server-side languages are PHP, Perl, Ruby, ASP.NET, Python, Java and ColdFusion. Some are proprietary, such as ASP.NET and ColdFusion, while Perl and PHP have the advantage of being open source, free, and available for servers that run either Linux or Windows. Consequently, they are offered by almost all web hosting companies. Of the two, PHP is easier to learn than Perl, and more widely used. At the time of writing, hotscripts.com, one of the best places to find ready-made web programs, offers 4,000 Perl programs and 15,000 PHP ones for download.

PHP, which stands for PHP Hypertext Processor, is the most widely used server-side language in the web world, so that's what we'll be using. And yes, the first "P" does indeed stand for PHP. Computer people are like that, sometimes.

Don't Panic!

The remainder of this chapter is all about programming. If you've ever done computer programming before, in languages such as Basic or C or Visual Basic, then you should feel right at home. If you haven't, don't worry. This section is written for non-programmers so hopefully everything will make sense. You will, though, find that it helps to put aside this book occasionally and practice on your own. As with any language, foreign or computer, practice makes perfect. Or at least better.

As you start to discover the power of PHP, you'll probably want to start working on a "real" project alongside our example hotel management system. Maybe it's a project that you've

always wanted to write but have never known how. Or maybe it's something that only entered your mind as you were reading this book. But it's a project nonetheless, and you reckon you could be the next Mr Facebook or Mrs Ebay.

Although PHP is easy to learn, it takes time to learn how to do it right. And in web programming, "right" generally means "secure" and "fast". If you don't do it "right", you'll end up with a site that is easy for people to hack into, or which takes 15 minutes to produce a database report that should take less than half a second.

So here's my advice, which I strongly suggest that you follow. Start off with the hotel management examples that follow. By all means use them as the basis for further experimentation, to learn more about PHP and web programming. But give it a couple of months before you start work on The Big Project. Otherwise, at best, you'll end up disappointed, with a system that doesn't do what you'd hoped it would. At worst, you'll end up with something that you're convinced is going to be the Next Big Web Thing, only to have your hopes dashed when a student on another continent hacks your customer database.

Oh, and if all this makes you want to start looking for the chapter on PHP security, there isn't one. Building secure web applications isn't about reading the chapter on security and applying it to what you've already learned. It's about doing everything in a secure way from the start. So rather than teach you about security now, or at the end, I'll actually cover security at every point where it is relevant. Which, as you'll soon realise, is just about everywhere.

Your First PHP Program

A PHP program is just a text file, like any other Web page. Most HTML editors don't know about PHP, though, so it's best to use one that does. That way, the editor will automatically alert you if it notices you've made an error in the code that you're typing.

Thankfully, PSPad, which we used for creating our Javascript code, also knows about PHP. Hundreds of thousands of people across the world use PSPad as their PHP editor, so that's what we'll use too.

While Javascript files tend to have a .html or .htm extension, just like standard web pages, PHP files tend to end with .php instead. This isn't a hard and fast rule. Some web servers will allow php code to exist within .html files, but many insist on you ending all your files with a .php extension. So, to keep things simple and consistent, we'll stick to a rule of always using a .php extension on any file that contains PHP code.

Right, the time has come to write a PHP program. If you haven't already installed PSPad, you'll need to follow the instructions on page 162 and do so. With PSPad installed, launch the program. Click the FTP tab on the left hand side of the screen, then click the Connect FTP icon in the row below it, and choose your site name from the list that appears. Then double-click on the public_html folder, or whatever your host uses to store your web-accessible files.

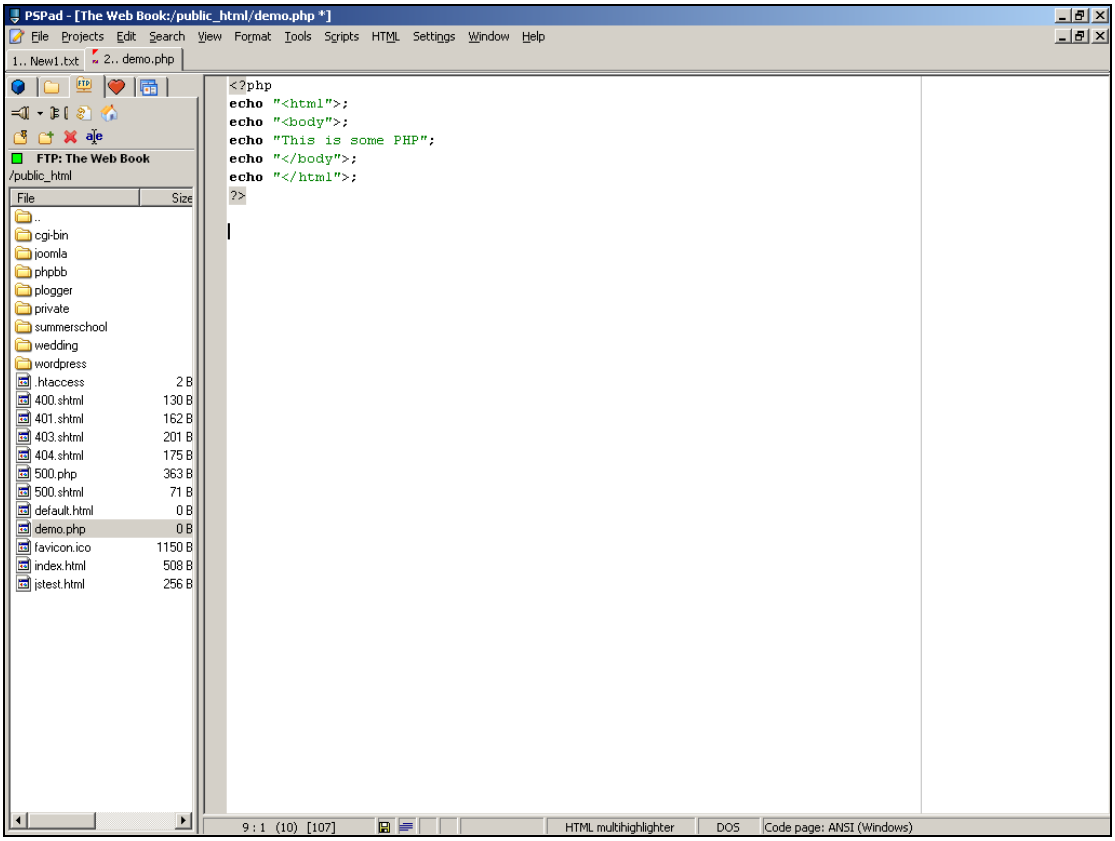
In the white space below the list of files and folders on the left hand side, right-click and choose New File. In the box where you're asked to choose a file name, type demo.php and click the OK button. You should see your new file appear in the list in the left-hand panel.

Now double-click your demo.php file to open it for editing. Look at the blue title bar at the top of the screen, the text of which should end with demo.php rather than new1.txt to indicate that you're editing the correct file. If not, double-click it again.

You can now start typing your first PHP program. Enter the following:

```
<?php
echo "<HTML>";
echo "<body>";
echo "This is some PHP";
echo "</body>";
echo "</HTML>";
?>
```

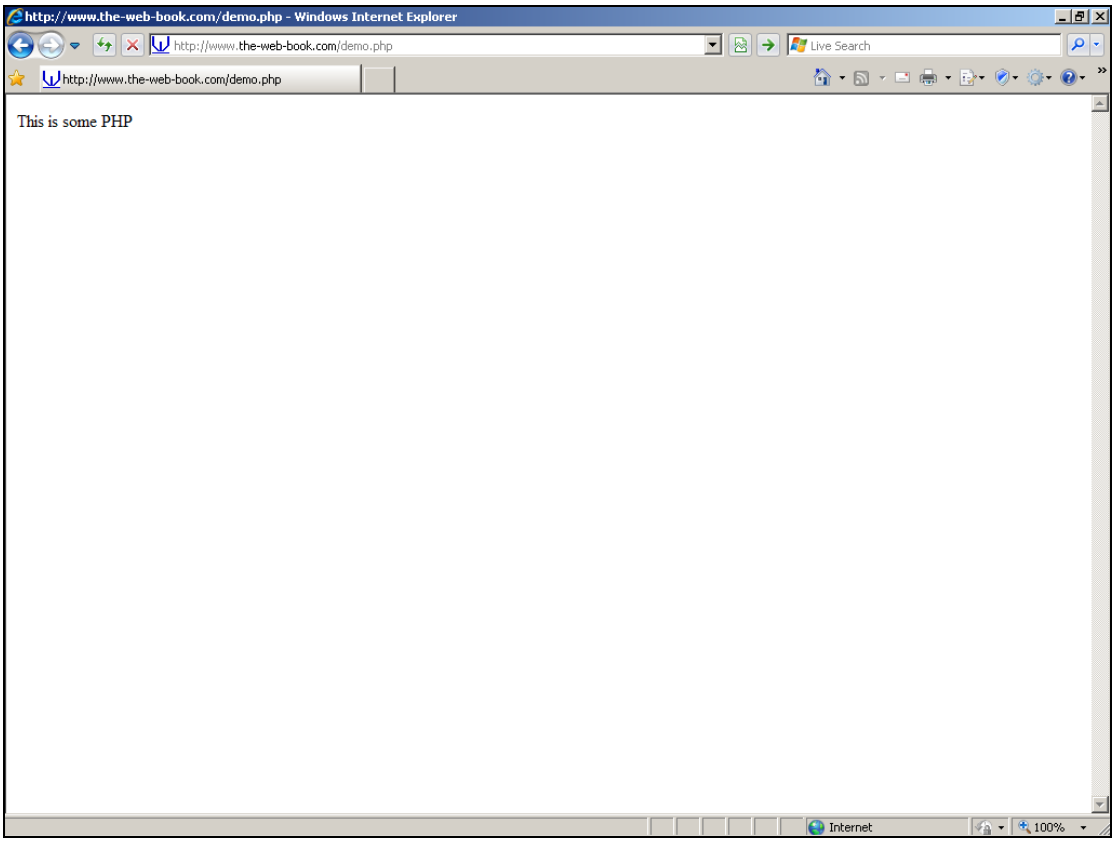
The screen should now look like this:



Save the file to the server (remember PSPad has built-in FTP so there's no need to upload anything manually) by choosing Save from the File menu. After a couple of seconds, when the file is saved, you can close PSPad.

Remember that, unlike Javascript, PHP code gets run by the web server, so you can't write and test PHP programs unless you're connected to a Web server. Hence the need to upload our program.

Let's see if everything worked. Open your web browser and surf to the page you just created. In our case that's **`www.the-web-book.com/demo.php`** and the end result is:



Excellent. It worked. The web server recognised and ran our program.

If you get an error message about syntax errors, go back and edit the file, and check your typing carefully. If you just see the PHP code itself, check that your web server really does support PHP and that it is enabled on your hosting account. It's possible that you have to specifically request it, though this is unusual.

Although this is a very simple PHP program, it does demonstrate a number of key points about the language.

First, you'll notice that the code starts with **`<?php`** and ends with **`?>`**. These special markers tell the web server where PHP code starts and ends, rather like the **`<script>`** tag that marks the start of some Javascript. Earlier versions of PHP allowed simply **`<?`** at the start of a script, but later versions insist on the full **`<?php`** instead. So even if you see web-based tutorials or books that tell you it's OK to use the shorter **`<?`** at the start, it's not. Get into the habit of using the full version and you'll be better off.

Next, notice the "echo" command. This is rather like document.write in Javascript, in that it creates content which is sent to the visitor's web browser to display. In this case, we've created an entire HTML page consisting of <HTML> tags, <body> tags, and some text. We have, in web developer terminology, created a dynamic web page. There is no .html file on

the server that contains this page. Instead, it has been dynamically created "on the fly", and sent to the visitor's browser.

Not particularly impressive, sure, but I'll show you some more powerful examples shortly.

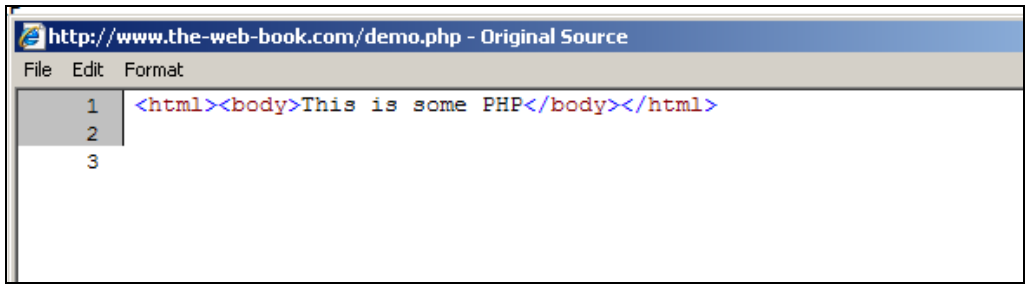
Note that you don't have to stick to pure PHP code within a php file. You can mix HTML and PHP code within the same file. For example, we could just as easily have written our program like this:

```
<HTML>
<body>
<?php
echo "This is some PHP";
?>
</body>
</HTML>
```

Here, we stay in HTML mode to create the first two tags. Then we switch to PHP mode to generate the text, and then back into HTML mode to generate the two closing tags.

As you start to write longer PHP programs, deciding on which way to work is important. There's no right or wrong way – just do as you prefer. Some people don't like the idea of flipping in and out of PHP mode 20 or 30 times within a complex file, whereas others prefer to stick in HTML mode and only use PHP mode where it matters. You'll see more about this later, but for the moment you need to bear in mind that there are two ways of doing things.

There's one more vital thing you need to understand about PHP, and indeed every server-side programming language. Assuming your web browser is still open, hit the View Source menu option and look at the HTML source code of the page. It looks like this:



Note how you can see the **output** of the PHP program, and **not** the PHP program code itself. The program was on the Web server, and the "echo" commands forced the HTML tags and other text to be sent to your browser. At no time does the PHP code itself leave the server, and at no time is it visible to the site visitor.

This is crucial to understand. It's the difference between server-side and client-side programming. And it means that you can safely use PHP to implement security-related features, because no one can tamper with the code.

If we'd written this program in Javascript, as a series of document.write statements, using the View Source option on our browser would have displayed the Javascript program code. But in this case, the browser never gets to see the code because it has already been run by the server.

One final point. Why are there no blank lines between each of the HTML tags, and before/after the text? Shouldn't the dynamically-generated HTML code look more like this, rather than being all in one line?:

```
<HTML>
<body>
This is some PHP
</body>
</HTML>
```

The reason that there are no "carriage return" characters in our dynamically-generated HTML, is because our PHP program never created any. Sure, we put each line of PHP code on its own line, but that won't cause carriage return characters to be sent to the browser.

It doesn't actually matter. Browsers don't insist on HTML code being neatly arranged in lines – it's only done for the benefit of humans who might want to read the code. But if you did want to generate neat, line-spaced HTML, you'd have to adjust your PHP code accordingly.

Needless to say, there's much more to PHP than the "echo" command. Like JavaScript, PHP is a hugely complex and capable language, which provides all of the tools you need to write sites such as eBay or Amazon. Perhaps the greatest benefit of knowing how to do server-side programming is being able to interact with a MySQL database, and we'll come to this soon. For now, though, the following chapter looks at some of the other things that PHP can do.

If you want to dive straight into PHP and experiment some more, the best resource is the official **www.php.net** site on the Web. Although the site can be confusing, it does contain everything you'll ever need to know. Perhaps its best feature is that anyone can add their own comments and example to the online manual.

To find out about any command in PHP, just go to **www.php.net/xxxx**, where **xxxx** is the command you want to know about. Alternatively, if you don't know what command to use, a Google search is a good place to start. For example, I mentioned earlier that our code didn't

insert carriage returns in the dynamically-generated HTML. Google for "php insert carriage returns" and you'll find out how to do so.

Don't worry about relying on the online documentation and Google. Even professional PHP programmers do it all the time. There's nothing wrong in not knowing about each one of the thousands of PHP commands, and the Web makes an ideal way to find out.

Some More PHP

As you'll see if you browse the online documentation on **www.php.net**, there are literally thousands of commands and functions in the PHP language, to allow you to achieve just about anything you want to do with a web page. It's impossible to cover everything here, or even to scratch the surface of what can be achieved. But before we progress to examine online databases, here's a small selection of other PHP stuff that might give you some ideas. Even if you don't intend to use these right now, you're recommended to read through this section as it will help a lot when we come to the database section later.

Incidentally, I've mentioned commands and functions. Is there a difference? Yes. A command does something, like open a file or display a message. A function returns information, such as the value of a database field or the sum of two numbers.

Random Numbers

Let's imagine that, on a particular web page, we want to include a message on our home page which thanks visitors for looking at our site. However, to avoid bombarding people with the message, we only want to show it approximately one time in ten. So, every ten times that the web server sends out the page to a visitor, it will include the message.

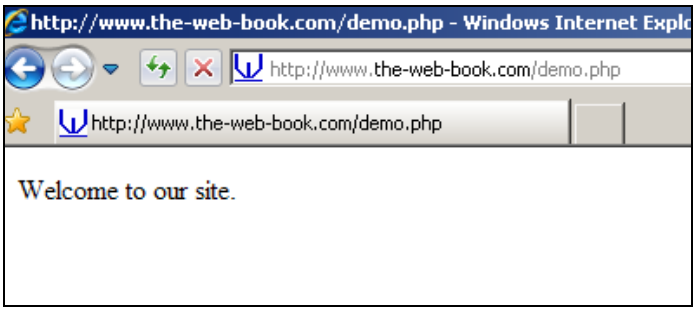
One simple way to do this is with PHP's random number generator. There's a PHP function which returns a random integer between 1 and any number we specify. So we'll ask it to generate a number between 1 and 10, and only display the message if the number happens to be, say, 7. This will result in the message only being shown approximately 10% of the time. That is to say, if 100 people visit the home page during one specific time period, roughly 10 of them will see the message.

Open PSPad and navigate to your demo.php file. Open it, delete everything that's in it, and replace it with the following:

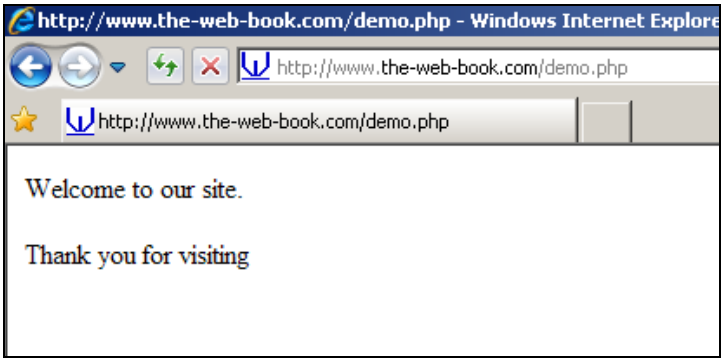
```
<HTML>
<body>
Welcome to our site.<br><br>

<?php
$r = rand(1,10);
if ($r == 7)
{
echo "Thank you for visiting<br>";
}
```

```
?>

</body>
</HTML>
```

Save the file (choose Save from the File menu), then open your web browser and surf to your demo.php file again. Keep pressing the Refresh button in your browser to reload the page. Most of the time you'll see:



But if you keep refreshing the page (pressing F5 is usually the same as clicking the Refresh button, if you prefer), sometimes you'll see this instead:



This is why PHP is so useful for generating dynamic HTML pages. It can generate different pages each time (or just some of the time).

Take a look at the source code of those 2 HTML pages (use the View Source option in your browser). Note how there's no PHP code in there, just the HTML that our code dynamically generated. The browser knows nothing about where the HTML code came from. It merely sent a "please send me the contents of demo.php" request to the server, and the server ran your program and sent some HTML back. The fact that the HTML was generated by a program, rather than being retrieved from a ready-made .html file on the server, is unknown to the browser.

Again, we've introduced a couple of new PHP concepts so let's take another look at the code we just typed and uploaded. Here it is again, with line numbers to aid explanation:

```
1   <HTML>
2   <body>
3   Welcome to our site.<br><br>

4   <?php
5   $r = rand(1,10);
6   if ($r == 7)
7   {
8   echo "Thank you for visiting<br>";
9   }
10  ?>

11  </body>
12  </HTML>
```

This time, I've chosen to mix HTML and PHP mode, and only to use PHP mode when necessary. So, to create our dynamically-generated HTML page, we start by outputting the necessary HTML and BODY tags. Remember that a PHP file always starts off in HTML mode, ie it's just a standard HTML file as far as the web server is concerned. Just because it has a .php extension doesn't change this. The only reason for the php extension is to tell the server that, if there is any PHP code in the file, it's safe to run it. Some servers will also run PHP code that's contained within a .HTM file, but many won't. And even if yours does, it's a good habit to be able to identify, from its extension, whether a file is plain HTML or PHP.

Line 3 generates the first line of text on the page, and I've added a couple of line break tags too, for clarity. (Now you can see why you need to know basic HTML before you can start writing PHP, and thus why the chapters in this book are ordered the way they are).

Once again, for clarity, I haven't created a css style sheet, or a doctype, or used <p> tags. The program will still work just fine, as browsers are generally pretty forgiving. When you're doing this for real, you'll need to do it properly.

Lines 11 and 12 close the HTML page neatly, by creating the closing body and HTML tags. Again, this is just standard HTML which we learned much earlier.

Line 4 flips the server into PHP mode. Line 5 creates a variable (we've called it $r) which uses the **rand** function. In this particular instance, the $r variable will be set to a value between 1 and 10. Note that every variable name in PHP has to be preceded with a dollar sign. Variables don't have to be given single-letter names. I could just as easily have called it

$my_random_value, but shorter is always best as there's less chance of making a typing mistake when you refer to the same variable later on.

Line 6 checks to see whether $r is equal to 7. If it is, everything between the 2 curly brackets gets executed. If it's not, everything between the curly brackets gets ignored.

Note two important points about the code in line 6. First, the **if** command is almost unique in PHP, in that it doesn't (indeed, mustn't) end with a semicolon. Second, when you're comparing something, you use 2 "equals" signs rather than just one. So, **$x = 99** will set the variable $x to 99. But to check whether $x is equal to 99 we say **if ($x == 99)**. Note, too, how the value to check for has to go inside rounded brackets.

Incidentally, if you leave PHP mode and then re-enter it within the same file, no information is lost. So, for example, the following:

```
<html>

<?php
$name = "Fred";
?>

Your name is

<?php echo $name; ?>

</html>
```

Will display the message **Your name is Fred**.

Sending Email with PHP

With server-side programming you can do just about anything, and that includes sending email. Try this little PHP program, by changing the contents of demo.php to the following and then surfing to the demo.php page:

```
<?php
$to = "robert@the-web-book.com";
$subject = "Hello";
$message = "This is my Web server, sending me email on ";
$message .= date("D");
```

```
$message .= " at ";
$message .= date("H:i");
$m = mail($to,$subject,$message);
echo "Return code from mail was " . $m;
exit();
?>
```

You should see a message which says "Return code from mail was 1". To understand why, and what this program does, let's examine it in detail.

The first line of code creates a new variable called $to, which is set to the "to" address of the email we're going to be sending. In this case, I'm sending the message to myself. If you want to try this example, please use your own email address rather than mine!

The next line sets the subject of the message in a similar way, and then we create $message, which is the text of the actual message we'll be sending.

But what about **`$message .= date("D");`**? Are we setting $message to something else entirely? No. Take a closer look and you'll see we're using **`.=`** rather than **`=`** on its own. The .= symbol means that PHP should add the new text to the end of what's in that variable already. But what text to add? In this case we're using the **`date()`** function and specifying "D" as what's known as the argument. The **`date()`** function returns lots of time- and date-related information, such as the current day, date, hour, minute, second, year, and so on. There are dozens of possible things that it can return, each symbolized by their own letter (and these are case-sensitive, by the way, so d is not the same as D).

In this case, D returns the current day of the week as a 3-letter abbreviation. So, this line is adding a 3-letter version of today's day onto the end of the $message string which we created in the line above. Thus, if today happens to be Friday, $message will now be "This is my Web server, sending me email on Fri".

The next line adds " at " to the $message string. So now it contains "This is my Web server, sending me email on Fri at ".

Finally we add the current time to the end of the message that we'll be emailing. We use the **`date()`** function again, with H and i. H returns the current hour in 24-hour clock format, and i returns the number of minutes. Anything that isn't a letter, within the bracketed part of the **`date()`** function, gets returned as-is. So if the time happened to be 9pm, **`date("H:i")`** would return 21:00.

PHP is particularly good at processing and manipulating strings and dates. Whatever you need to do with a string (count the number of characters, convert it to upper or lower case,

remove the last 4 characters, etc etc), PHP has a function to do it. As for dates, there are even functions which will, if you supply a date and a latitude/longitude location, tell you the time of the sunrise or sunset.

Right, back to our program. Next, we use the mail() command to send the message. It's as simple as specifying, in the correct order, the "to" address, the "from" address, and the text of the message to be sent. But note the **`$m =`** bit at the start, though. The mail() function returns a value to indicate whether the message was successfully sent to the mail server for delivery. We capture this into the $m variable, and display it in the next line.

Finally, the exit() statement ends the PHP program. You don't need to use this if it's unambiguous, as it is here. The program is clearly going to end, as there's no more PHP code left to run. But that won't always be the case, as we'll see later.

By surfing to the demo.php page, the program runs and the mail gets sent. The value of 1 for $m means that the mail message was successfully sent to the web server's outgoing mail system. It does NOT mean that the message was successfully sent to the recipient's inbox. Unfortunately there's no way of knowing that.

A few minutes later, the following arrives in my inbox:

Here's the message. It has the correct subject, and the text of the message is there too. But there are a couple of things to notice.

First, the message appears to come from **`thewebbo@host368.hostmonster.com`**. That's not particularly friendly. There's an optional extra item that you can include in the **`mail()`** function to specify the address that the message appears to come from, though I haven't bothered with it here. In theory you can set the "from:" address to be whatever you like. But in practice, most web servers won't send mail that appears to have come from anywhere outside of the domain name that the server is based on. So in this case I could send mail that

appears to come from **robert@the-web-book.com** but not **robert@microsoft.com**. This rule is imposed to help prevent web hosts from having their servers used by spammers.

It's important that you do set a "from:" address when sending mail. Because that's where the reply will be sent when the recipient hits their "reply" button. I'd much rather the replies came to a real mailbox at **robert@the-web-book.com** rather than something which I wouldn't even know how to log into if I wanted.

There's one other discrepancy too. See how the text of the message says that it was sent on Fri at 14.15. But look in the upper window and, according to my Outlook email program, the message was received at 21.15. The message appeared in my inbox within seconds of me surfing to the demo.php page, so why does it appear to have taken 7 hours? You will recall that PHP is a server-side language. So when you use the **date()** function to find out the time, it looks up the time on the server. Compare this to the client-side Javascript, which finds out the time on the visitor's own PC. The hostmonster web server is in the US, where it's 2.15pm. I'm in the UK, 7 hours ahead, where it's 9.15pm, so that's why the discrepancy arises. And it's well worth knowing about, because it can lead to problems if you create a web application that needs to know the correct (whatever that means, in this global market) time.

Passing Information to PHP

There are various ways of passing information to a PHP program, and we'll cover many of them in the database-related chapters that follow, and when we come to talk about HTML forms later on. But here's a fun example for now, which illustrates a couple of important points about PHP.

Open PSPad, delete the contents of demo.php and replace it with the following:

```
<?php
$person = $_GET["name"];
echo "<HTML>";
echo "Hello ";
echo $person;
echo ", how are you today?";
echo "</HTML>";
?>
```

Surf to the demo.php page and you'll see a message that says "Hello , how are you today".

Now surf to the page again, but use the following URL. Substitute **the-web-book.com** for whatever your domain is called. And you can change Robert to your own name if you wish, too:

```
http://www.the-web-book.com/demo.php?name=Robert
```

Note the question mark at the end of the URL, followed by **name=Robert**. You've probably seen URLs like this before, and you might have wondered what they meant. Now you're going to find out.

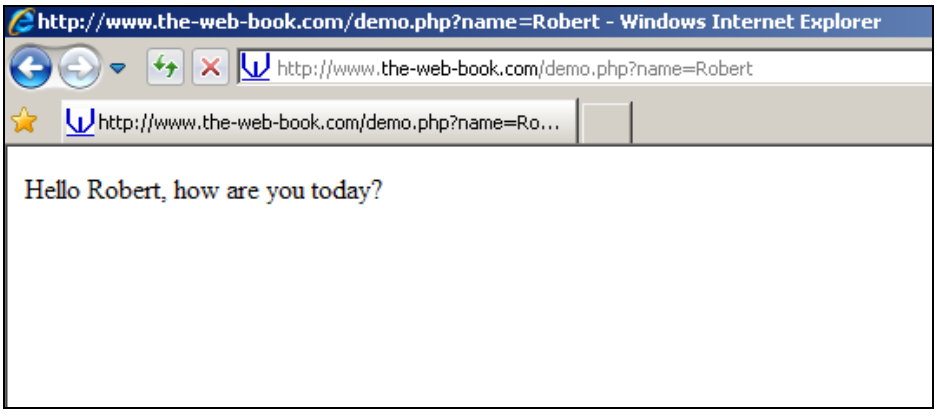
The question mark in a URL allows you to surf to a server-side program (in this case one called demo.php) and also to pass information to the program. In this case, we have created a variable called **name** with a value of **Robert**. The PHP program can, if it wishes, retrieve the contents of that variable.

Along with the question mark, the other symbol that you use for passing information to PHP programs via the URL is the ampersand (&) sign. This is used when you want to pass more than one variable. For example:

```
http://www.the-web-book.com/demo.php?firstname=Robert&surname=Smith
```

Here, we're passing 2 variables called firstname and surname. Note how only the first variable is preceded by a question mark. Any subsequent variables must be preceded by an ampersand.

Back to our demo.php program. Assuming you entered a name of Robert on the URL, you'll see the following in your web browser:



To understand how this works, take another look at the PHP program code. The only line that will be unfamiliar to you is **`$person = $_GET["name"];`**

The **`$_GET[" "]`** syntax is how you retrieve variables (known as parameters in this context) that were added to the end of the URL. In this case, because we referred to the parameter as **`name`** in the URL, **`$_GET["name"]`** retrieves it for us. I'm retrieving it into a variable called $person rather than $name, but that's entirely up to you.

Note that the URL variable in the square brackets doesn't have a dollar sign at the start, so **`$_GET["$name"]`** wouldn't work. And note that GET is in upper case – this is important too.

If you've got a few minutes spare, here's something that you can try. Enhance the program so that, if no name is entered on the URL, the program displays an error message. To do this, you'll need to know that the **`strlen()`** function returns the length, in characters, of a string. So having retrieved the name from the URL, code such as:

```
if (strlen($person) == 0)
```

will allow you to find out whether any name was supplied.

Alternatively, adapt the program to display not just the name that was entered, but also the number of characters in the name.

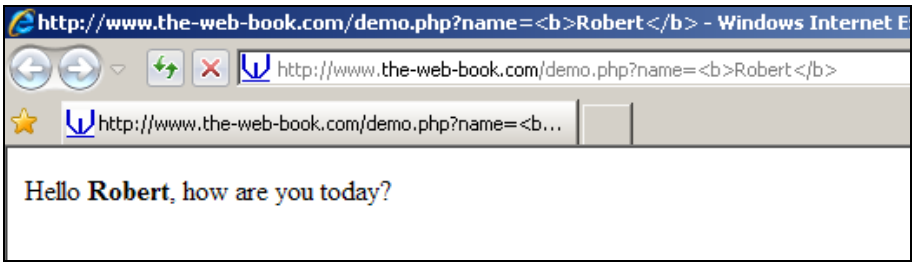
Never Forget to Sanitize

Being able to pass information to a PHP program so easily is a good one, and you'll find that you use it a lot. Mostly, it's used in forms. For example, your visitor fills in a web-based form with his name and address, which gets retrieved by a PHP program and displayed on screen or added to a database. But the ease with which PHP can accept information from visitors to your site hides a very serious security flaw. You can't trust that information, because the visitor is free to enter anything he or she likes.

Here's a very simple example. Surf to demo.php again by using the following URL:

```
http://www.the-web-book.com/demo.php?name=<b>Robert</b>
```

This is what you'll see in your Web browser:



See how my name is now displayed in bold? That's because I added the relevant HTML tags to the URL. The PHP program uses the echo statement to insert, into the dynamically-generated web page, whatever was specified on the URL. In this case it's not just some text, but some actual HTML code too. You can see it if you use the View Source option in your web browser.

But why is this bad? Because, at a trivial level, someone can mess up the look of your web page by forcing it to display lots of stray HTML tags. At the other end of the scale, consider what would happen if someone entered a "name" of <script> followed by a load of Javascript code. Would the web browser then execute that code? Yes, it would.

All of which leads to one of the most fundamental rules of security when it comes to PHP programming: filter or sanitize information whose source you can't be sure of.

Quite how you do this will depend on circumstances, and on the precise nature of the data. In our current example, where we're expecting the visitor to enter a name, it's obvious that the only characters we need to allow are the letters a to z (and A to Z). Any other character can be removed. This will include the pointy brackets which would allow someone to enter HTML code, and also many other unnecessary punctuation symbols. You'll see later, when we talk about SQL Injection attacks, that some of those other punctuation symbols are just as dangerous.

One way to filter or sanitize data is to use the PHP **str_replace** function. This stands for "string replace". It can quickly and easily replace any characters in a string (ie, in a variable that corresponds to some text) with another character. Or, if you prefer, it can replace them with nothing and thus delete them.

For example, having retrieved the name from the URL into the $person variable, we could then add a line which says:

```
$person = str_replace("<","",$person);
```

As you can see, **str_replace** requires 3 parameters. What to search for, what to replace it with, and the variable within which to do it. This line would have the effect of replacing all < symbols in $person with nothing, ie deleting them.

Although this would work, you'd need lots of str_replace lines to deal with every unwanted character that you need to filter out. But thankfully there's a neater way, using a function called **ereg_replace**. Take a look at this:

```
$person = ereg_replace("[^A-Za-z0-9 .,';:?]", "", $person);
```

This is similar to str_replace, in that you specify what you want to look for, what you want to replace it with, and the string to operate on. But take a closer look at what we're searching for. No longer are we specifying a single character, but:

```
[^A-Za-z0-9 .,';:?]
```

Although this might look like gobbledygook, it's actually just shorthand for the entire list of characters to search for. Let's go through it in detail. First, the whole thing goes in square brackets, because that's the rule for ereg_replace. A-Z means every character from A to Z. Equally, a-z means all the lower-case letters. Then, in addition to those 52 characters, we list a few others. Namely the space, full stop, comma, apostrophe, semicolon, colon, and question mark.

The most important character in this whole collection, though, is the ^ at the start. This is ereg_replace shorthand for "everything except". So what the whole command actually does, is to replace every character in $person with a blank (ie, to delete that character), EXCEPT where the character is a letter, a space, a comma, an apostrophe, and so on.

You don't really have to understand it. Just add this line to your PHP code, after the line which retrieves $person from the URL. Now surf to demo.php again and try entering "forbidden" characters. You'll notice that they don't appear in the generated web page.

Sanitizing strings is fiddly but, thankfully, it only normally takes just line or two of extra code. And it is vital that you do sanitize every string that comes from a visitor to your site, because you have no idea what the visitor has typed. Get into the habit now. Every time you use **$_GET** to retrieve information from a URL, sanitise it before using it. Failure to do this will mean that your site WILL get hacked.

Incidentally, the idea of deleting everything EXCEPT a specified range of characters isn't just convenience, it also makes a lot of sense from a security perspective too. It's much better to say "these are the characters I'm going to allow", rather than "I'm going to allow everything apart from the following". In the latter case, forgetting to add something to your list of banned characters poses a serious security risk. In the former case, there's no risk of this

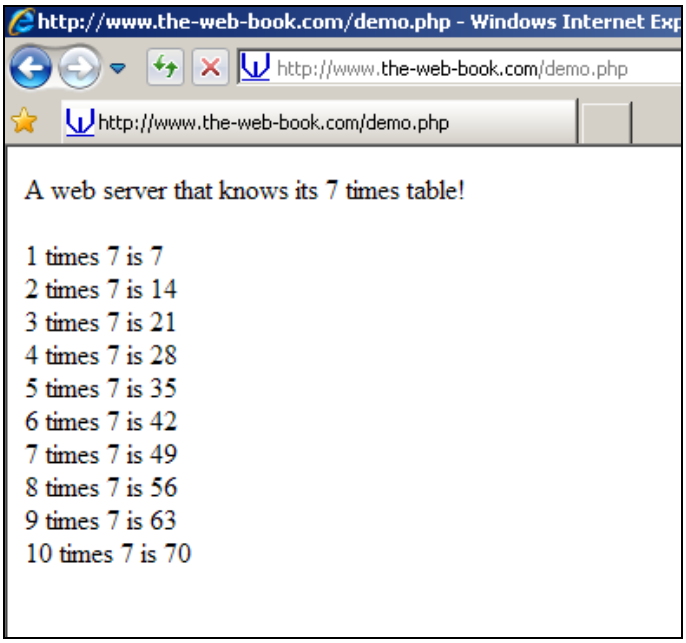
happening, and the worst that can happen is that some characters won't get through until you remember to add them to the list.

Loop the Loop

Every programming language has a number of methods for creating loops, and PHP is no exception. Try this program, for example, either by uploading it the server and seeing what happens, or just by reading the code and trying to work it out:

```
<?php
echo "<HTML>";
echo "<body>";

echo "A web server that knows its 7 times table!<br><br>";

for ($x = 1; $x <= 10; $x++)
{
echo $x . " times 7 is " . $x * 7 . "<br>";
}

echo "</body>";
echo "</HTML>";
?>
```

Run the program, and you'll see the following in your web browser:



One line of PHP that you'll not have seen before is:

```
for ($x = 1; $x <= 10; $x++)
```

This is one of PHP's methods of creating a loop. In this case, it's a "for" loop. If you ever learnt to program in Basic many years ago, it's just the same as a FOR...NEXT loop in that language.

Translating the line of PHP into English, the line of code means: set variable $x to 1, and execute everything within the curly brackets that follow. If $x is less than, or equal, to 10, exit the loop and jump to the first line of code after the section in curly brackets. Otherwise, add 1 to $x and go round again.

The **$x++** is a PHP shorthand way of saying "add 1 to $x". Another way, which works just as well and which is more familiar to Basic programmers, is **$x = $x + 1**.

By changing the starting value (1) and the finishing value (10), you can adapt this code to run as many times as you like. Try it.

As for generating our 7 times table, the line that does all the work is:

```
echo $x . " times 7 is " . $x * 7 . "<br>";
```

Here, we've used a single echo statement to produce the entire line of our table. Each part of the line is separated by a dot. Literal text such as **times 7 is** gets enclosed in quote marks, whereas we leave the quote marks out if we want PHP to echo the value of a variable such as $x, or make a calculation such as $x * 7.

Don't forget that, whenever you write a PHP program, you're creating a program that will generate a web page. This needs to be a valid web page, complete with <HTML> and <body> tags. Otherwise, the visitor's web browser may get confused.

Another type of loop that's used extensively in PHP, especially when accessing MySQL databases, is the "while" loop. Here's the 7 times table again, this time using a "while" loop:

```
<HTML>
<body>

My web server still knows its 7 times table
<br><br>
```

```
<?php
$i=1;
while($i <= 10)
  {
  echo $i;
  echo " times 7 is ";
  $product = $i * 7;
  echo $product . "<br>";
  $i = $i + 1;
  }
?>

</body>
</HTML>
```

Just to be different, I've mixed HTML and PHP modes this time.

Notice how a "while" loop works. First, we set $i to 1. Then we create the loop, which says "do the block of code within the curly brackets, but only while $i is less than, or equal to, 10".

Because there's a line within the block which adds 1 to $i, there will come a point at which $i is not less than or equal to 10, so the program will fall through to the next line after the closing curly bracket.

It's worth studying these two examples of loops, and especially the "while" loop, as they are used extensively in database-related PHP programming.

Arrays

Programming in PHP is all about manipulating variables, ie those items of data whose name starts with a dollar sign. The key to manipulating data is to ensure that the data is stored in the most efficient way, in order to make the manipulation as efficient as possible.

For example, let's return to the subject of our hotel management system and put together a list of all the drinks available at the bar. We want to store the drinks menu in a set of variables, so that we can display them, sort them, record orders for them, and so on. Here's one way of doing it:

```
$drink0 = " Whiskey";
$drink1 = "Brandy";
```

$drink2 = "Pint of lager";
$drink3 = "Shandy";
$drink4 = "White wine (large)";
$drink5 = "White wine (small)";
$drink6 = "Red wine (large)";
$drink7 = "Red wine (small)";

Having defined our variables, we now want to use a set of echo commands to display the drinks menu on screen. But how? Each drink name is stored in a completely separate variable. Sure, each one starts with the letters d,r,i,n and k, but then come different numbers. As far as PHP is concerned, the variables are totally unrelated to each other, even though their names are similar. Displaying our menu is going to be difficult.

There's a better way, using something called an array. An array lets you create a group of variables which are linked, and you can easily access any member of the chain. Here's how to do it:

$drink[0] = " Whiskey";
$drink[1] = "Brandy";
$drink[2] = "Pint of lager";
$drink[3] = "Shandy";
$drink[4] = "White wine (large)";
$drink[5] = "White wine (small)";
$drink[6] = "Red wine (large)";
$drink[7] = "Red wine (small)";

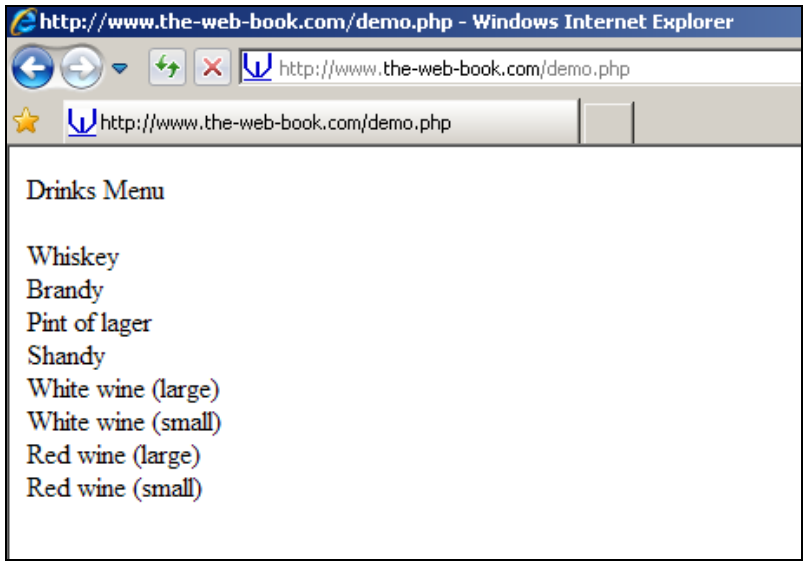
Note the minor addition of those square brackets. They're important. No longer do we have 8 separate variables. Instead, we have one single array which has 8 elements. Note how we started at zero, by the way. In PHP, array elements generally start at zero. This is a slight pain, as it means the final element of a 10-element array is number 9 rather than 10. But once you get used to it, and remember to write your program code accordingly, it's pretty simple.

Here's where arrays come in so useful. To display the value of drink number 3, we just use **`echo $drink[3]`** as you might expect. But even better, we can use a loop to display the full drinks menu like this:

```
<?php
$drink[0] = "Whiskey";
$drink[1] = "Brandy";
$drink[2] = "Pint of lager";
$drink[3] = "Shandy";
```

```
$drink[4] = "White wine (large)";
$drink[5] = "White wine (small)";
$drink[6] = "Red wine (large)";
$drink[7] = "Red wine (small)";

echo "<HTML>";
echo "<body>";

echo "Drinks Menu<br><br>";

for ($i = 0; $i <= 7; $i++)
{
echo $drink[$i];
echo "<br>";
}

echo "</body>";
echo "</HTML>";

?>
```

Which displays the following in the visitor's web browser:



Our "for" loop counts from 0 to 7, and displays the relevant element of the array each time.

The only slight annoyance is that, if we add another drink to the menu by creating element number 8, we'd also have to change the 7 to an 8 in the "for" loop. This is inconvenient, and

introduces the possibility of an error in the program if we forget to do this. As you might expect, there's a way round it. Just change the "for" loop to:

```
$total_drinks = count($drink);
for ($i = 0; $i <= $total_drinks; $i++)
```

The count() function returns the number of elements in the specified array. But remember that arrays start at zero, so treat the returned count accordingly. In this case, the function returns 9, but that doesn't mean the highest-numbered drink is element 9. The list starts at 0, so the highest-numbered element will be one less than the value which count() returns.

When you're adding data to an array, you don't even have to specify the numeric bit if you don't want to. The following will work just fine, and PHP will allocate the numbers automatically, starting from zero. This is handy if you're reading a list of items from a database and you want to add them to an array in order to manipulate them.

```
$drink[] = "Whiskey";
$drink[] = "Brandy";
$drink[] = "Pint of lager";
$drink[] = "Shandy";
$drink[] = "White wine (large)";
$drink[] = "White wine (small)";
$drink[] = "Red wine (large)";
$drink[] = "Red wine (small)";
```

PHP has lots of useful functions for manipulating data in arrays. For example, **`array_sort`** does what you might expect, allowing you to add just one line of code to the above program to present the list of drinks in alphabetical order. Although, if you were reading the list of drinks from a database table, it's far more efficient to use the **`ORDER BY`** clause as part of the MySQL query, in order to retrieve a ready-sorted list, rather than read the data in a random order from the database and then sort it within the array.

Before we finish our very brief introduction to arrays, there's something important that you need to know about them. In some languages, such as Basic, array subscripts (ie, the bit in square brackets) have to be sequential and numeric. In PHP, this is not the case. We could, if we wish, create our array of drinks thus:

```
$drink["a"] = "Whiskey";
$drink["fred"] = "Brandy";
$drink[2] = "Pint of lager";
$drink["hello"] = "Shandy";
$drink[412] = "White wine (large)";
```

```
$drink["-8"] = "White wine (small)";
$drink[6] = "Red wine (large)";
$drink[7923] = "Red wine (small)";
```

It's unlikely that you'd ever want to do this (at least, I hope not!), but it's worth bearing in mind the pitfalls of not having subscripts (or keys, as PHP calls them) which follow a pattern. Why? Because, if you use the **`count()`** function to return the total number of elements in an array, you can only use the for.. loop method as described above if you know that the elements are numeric, sequential, start at 0, and stop at one less than the value returned by the **`count()`** function. If they don't, you'll need to find another way of displaying them.

Should you need to do this, there's a PHP function called **`array_keys()`** that returns (as an array, naturally) the list of element names of a given array. In this case it would return an array consisting of a, fred, 2, hello, 412, -8, 6 and 7923. You could then use this array to help you navigate the $drinks array.

Be aware that, if you delete an element from an array using the **`unset`** command, the remaining array will have a "hole" in it. For example, we currently have our array of drinks from 0 (whiskey) to 7 (small red wine). We then delete element 3 (shandy) because it's no longer available on the menu. We can do this with:

```
unset $drinks[3];
```

There are now 7 elements in the array, numbered 0,1,2,4,5,6,7. A call to count($drinks) will return 7, but those elements aren't numbered 0 to 6. Instead, they're numbered 0 to 7 with a non-existent 3.

The easiest way around this is to renumber the array and close up the gaps every time you delete an element. Which is as simple as:

```
$drinks = array_values($drinks);
```

Now, $drinks has elements numbered 0 to 6. Just remember to call the count() function again, so the number of the highest element is updated.

If you've ever used arrays in Basic before, trust me. This ***will*** catch you out!

User-Defined Functions

PHP programmers make great use of something called user-defined functions. So far, we've used only PHP's built-in functions like **`count()`**, **`mail()`**, **`strlen()`** and so on. But the

great thing about PHP is that you can create your own. For example, here's a slightly different version of the "how are you today?" program as shown on page 222:

```
<?php
$person = $_GET["name"];
$person_san = sanitize_url_string($person);
echo "<HTML>";
echo "Hello ";
echo $person_san;
echo ", how are you today?";
echo "</HTML>";
exit();

function sanitize_url_string($string)
{
$s = ereg_replace("[^A-Za-z0-9 .,';:?]", "", $string);
return $s;
}
?>
```

There are a couple of differences between this version and the original. First, after using $_GET to retrieve the value of the name from the URL, we then call the **sanitize_url_string()** function to clean it up and remove any potentially dangerous characters. The function itself is defined further down, so we need an **exit()** statement before the function definition, in order that the code which defines the function doesn't get executed. It doesn't need to be – PHP will search for it automatically in the file if it needs it.

Note how I've created a new, sanitized version of $person, called $person_san, and this is the version that gets displayed by the echo statement. I could, instead, just have said:

```
$person = sanitize_url_string($person);
```

which would have replaced the unsanitized version with the clean one. But it's a good idea to get into the habit of giving all sanitized data an easily identifiable moniker, such as _san. That way, it's easy to spot places in your programs where you might have forgotten to clean up toxic data.

Finally, the definition of the function itself. This takes the form of a line which starts with the word "function", then the name you want to use, then, in brackets, the list of however many parameters your function requires. In this case it needs just one, namely a string to be sanitized. The name of the variable ($string) within the function definition needs to match the name used within the code of the function, but doesn't have to match the name used

outside. There's no need to have referred to $person in the function at all. It just so happens that we're using the function to sanitize a person's name. Later on in our program we might call the same function to sanitize $postcode or $occupation.

Note the **`return`** command. This is what returns the results, ie the sanitized string, to whatever called the function. Without this, $person_san would be blank.

One of the characteristics of "good" programming, whether in PHP or any other language, is that you don't include the same bit of code more than once. Whether that "bit of code" is a single line or a block of 100 lines, the rule still applies. By including the code just once, and turning it into a function which can then be referred to many times, your programming becomes more efficient. It saves space, and means that, if you make a change to that "bit of code", you only have to update one copy of it rather than locating and amending every instance.

HTML Forms

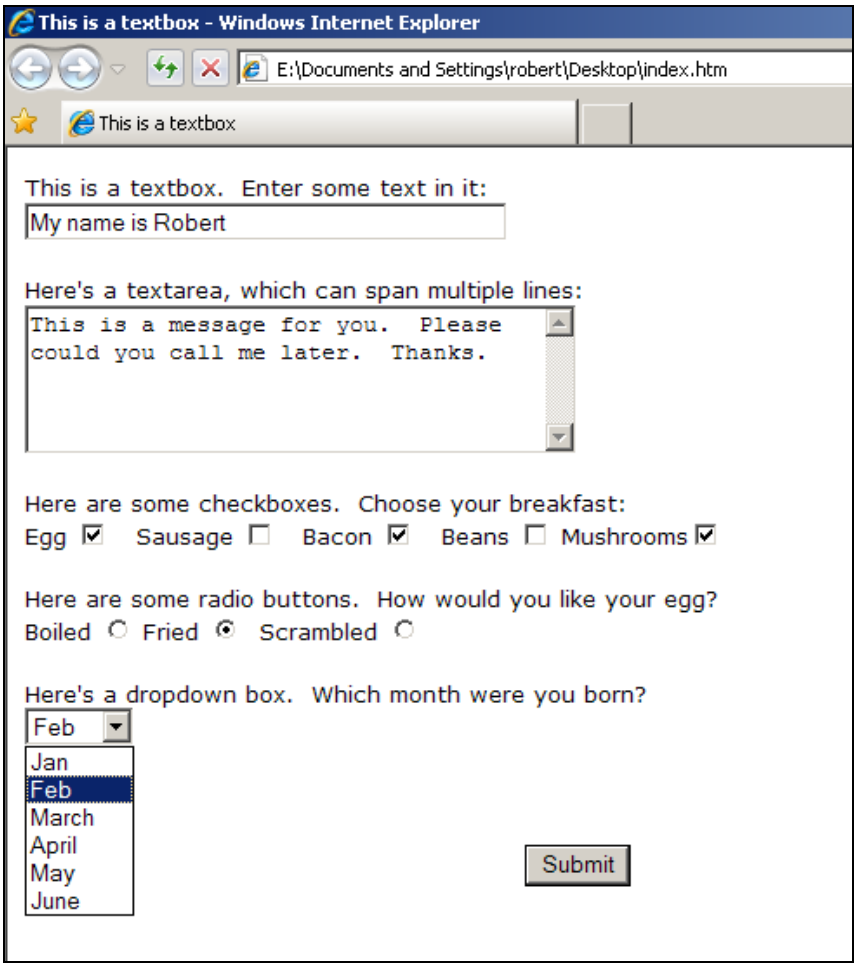
So far, The Web Book has been all about publishing information in one direction, ie from you and your web site to your visitors. Sometimes, though, you'll want to be able to request information in the other direction, ie from a visitor. This might be a choice of drink from the menu in our hotel management system, or someone's name and address in order to add them to the subscriber database, or their username and password in order to attempt to log them into a protected system.

There are two ways of getting information into a PHP program. One is to use parameters on the URL, and retrieve them with $_GET. The other method, which is much more powerful (and more secure, if done properly) is to use an HTML form.

When we finally start to design database applications with PHP, we'll use forms extensively to request information from visitors that ultimately gets stored in the database. Therefore, you need to familiarise yourself with the contents of this chapter before reading further.

A form consists of two main components. First, one or more input fields into which the visitor types or clicks the information you have requested. Second, a "submit" button which, when clicked, sends the contents of the form to a server-side program for processing in whatever way it wishes.

There's a variety of form field types that you can use, depending on the type of information that you're requesting from a visitor. Here are examples of the ones you'll use most often:



First is a textbox. This is used for requesting a single line of text. If you're using a textbox to ask a visitor to enter a password, you can configure the textbox as a password box. If you do this, characters that the visitor types into the box appear as dots or stars, to stop someone looking at the password over the visitor's shoulder. However, when the submit button is clicked, the real content of the text box (ie, the password as entered) gets sent to the receiving program.

Next comes a textarea, which can be as wide and high as you wish. You might want to use this if you have a feedback form on your site, into which visitors can type a message.

Next, the two sets of option boxes. First, checkboxes. A set of checkboxes allows the user to choose one or more items from a list. Here, we're asking him to choose which items he'd like for breakfast.

The second set of option boxes are radio buttons. These differ from checkboxes because you can only choose one option. In this case, we're asking how the visitor would like their eggs cooked. Another common use for radio buttons is to ask a visitor whether they are male or

female. If you used checkboxes for such a question, a high percentage of people will think it amusing to tick both boxes. But with radio buttons, they can't.

Finally, a dropdown box. From here, a visitor can select an option from a pre-determined list. By holding down the control key in their Web browser they can actually select more than one item, so long as you permit this when you create the form.

A dropdown box is a powerful tool, which should not be underestimated. Where possible, it's much better to use a dropdown box than a text box. For example, imagine an application where you ask someone to enter their favourite make of car. If you provide a text box, the replies might include "ford", "Ford", "FORD", "Ford I guess" and so on. If you limit the choices by using a dropdown box, you'll end up with much more consistent data. Which will be extremely helpful when you come to produce a report from your database on the percentage of people who prefer Ford cars.

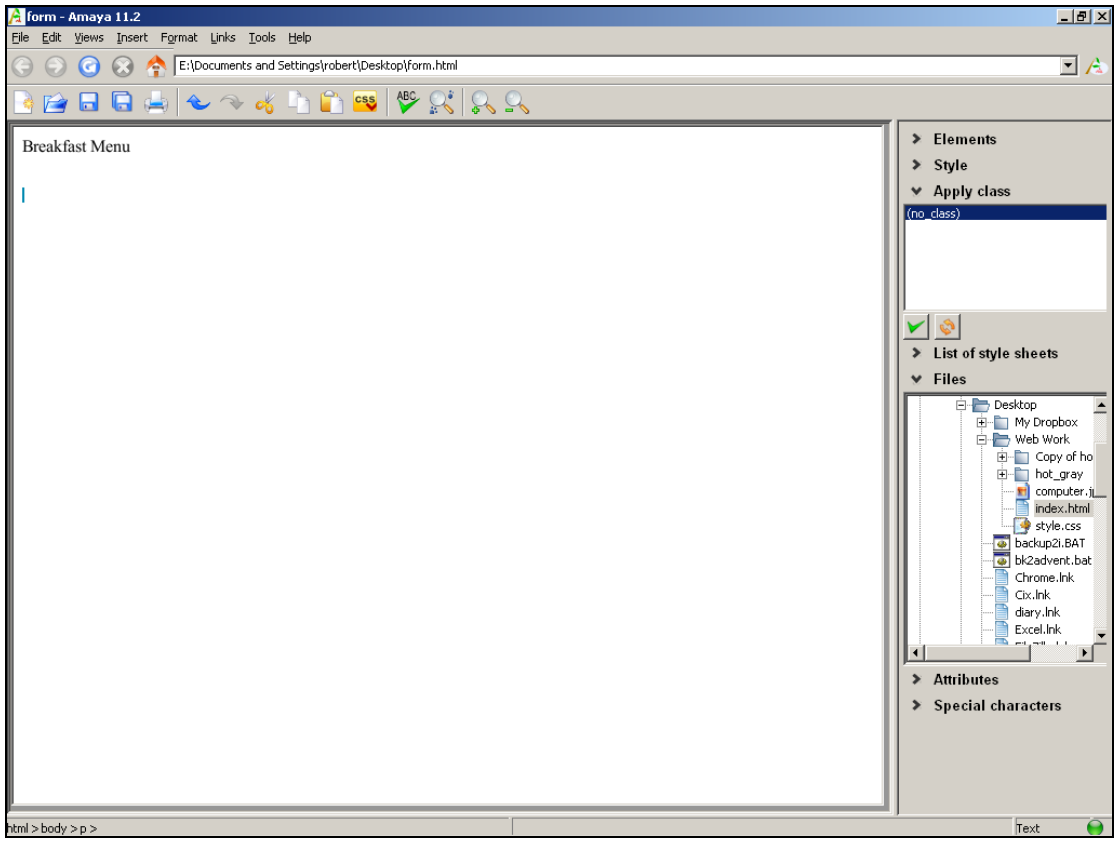
Creating a Form with Amaya

Although you can create a form by hand, typing the HTML code into a text editor such as PSPad, it's easier to use a dedicated HTML editor that has form capabilities. Throughout this book we've been using the free Amaya HTML editor, so that's what we'll use to create a form.

To begin creating a form, start Amaya. From the File menu choose New, and then choose New Document. Choose a name (form.html will be fine) and location for the file (your Web Work folder is probably easiest location, as we'll want to upload the form to our server later and FileZilla should already be configured to look there).

Once you've chosen the location and a name, click the Create button.

Type "Breakfast Menu" at the top of the page, then press Return a few times to move further down the page. Your screen should look like this:



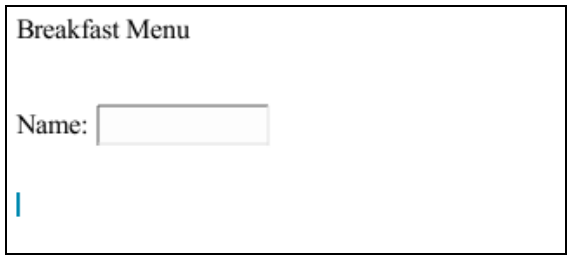
You're now ready to create the form.

From the Insert menu, choose the Form option, then choose Insert A Form. You'll be asked to enter details of a form action, but this is something we'll cover later so, for the time being, just press OK.

You won't notice anything on the screen, but Amaya has inserted the necessary HTML code to create a form. Click on Views and then Show Source, and you'll see it. There's not a lot there at the moment – just a **<form>** tag or two. But that's about to change, as we start to add our form fields. So if you're looking at the source view, return to standard view by unticking "Split View horizontally" from the Views menu.

Let's insert a text box, for our breakfast guest to enter their name. Type "Name:", which we'll use as a label for the textbox. Then, from the Insert menu, choose Form and then Text Input.

You should see something like this:



Let's make that text box slightly longer, as it's not quite big enough to hold a name as it stands. Some HTML editors let you drag the box to adjust its size, or right-click it to see a menu of things you can change. But Amaya doesn't have such a feature, so we'll just have to change the HTML code directly. It's not difficult, and it's a good way to learn anyway.

From the Views menu, click on Show Source. Locate the HTML tag which says **<input type="text" />**. The **<input>** tag is how form items such as textboxes and checkboxes are created. The **type="text"** part is what makes this a text box. What's missing is any specification for the width of the box. So change the tag to read:

```
<input type="text" size="40" />
```

Click back into the top window and see how the length of the box changes. If you want to make it even longer than 40 characters, feel free.

That's the first half of our simple breakfast menu form complete. Now let's add some checkboxes and meal items.

At the bottom of your page, type "Please choose your order", to act as a label for the next collection of form fields. On subsequent lines, type some food choices. In my example I'll use eggs, bacon, sausages and beans.

Then place the cursor next to each food item in turn and, from the Insert menu, choose Form and Checkbox.

In a similar way, type a "How do you like your eggs?" label, then some possible choices. Next to each choice, from the Insert menu, choose Form and then Radio.

Finally, we need to add a Submit button otherwise the user won't be able to make use of the form. So, at the bottom of the page, again from the Insert menu, choose Form and then Submit.

Your screen should now look something like this:

Incidentally, you might well be wondering why your form looks so, well, messy, and what you can do about it? Specifically, you probably want to be able to line up the checkboxes and radio buttons in a neat vertical column. The easiest way to do this is to use a table. For the food order checkboxes, for example, create a table with 2 columns and four rows. Put the food labels down the left hand column and the checkboxes in the right-hand cells. Then everything will line up neatly. For now, though, we're not too concerned about cosmetics so I'll keep things simple and table-less.

Naming the Form Objects

Having designed a form, there are 3 more things you need to do before it's ready for use. These are:

1. Ensure that each form object is named properly
2. Add an "action" to the <form> tag
3. Write some PHP code to handle the submitted forms

We'll go through each of these in order.

First, the naming of form objects (ie, all those checkboxes, textboxes and so on).

When the site visitor presses the Submit button, the contents of the form will be sent to a PHP program as a series of variables. The names of those variables will be the names that you assign to the objects on the form. Therefore, it makes sense to ensure that the names you choose are sensible. Obviously, no HTML editor will know in advance what you intend to use the contents of, say, a checkbox for. Some editors, like Dreamweaver, generate names such as "checkbox1" or "radio1", while some, such as Amaya, don't generate anything at all. Either way, you'll definitely want to change the names to something meaningful. It makes a lot more sense for a PHP program to be dealing with a variable called $name or $food rather than $checkbox1 or $radio.

So, in Amaya, go to the Views menu and select Show Source. Start by locating the definition of the "name" textbox, which looks like this:

```
<input type="text" size="40" />
```

There's nothing in that HTML tag which explicitly says that this is the "name" textbox, but you can work it out from various clues, the most obvious one being that it comes right after the word "Name" which we used as a label. Welcome to the world of the HTML detective!

Change the line of code to:

```
<input type="text" name="tb_name" size="40" />
```

As you can see, naming a textbox is no more complicated than adding **name="whatever"** as part of its tag. Note how I've called it **tb_name** to remind me that this data has come from a textbox. This is a useful habit to adopt. Sometimes it will be helpful to remember what kind of data you're dealing with.

The order of the various data items, or parameters, within an HTML tag after **input** doesn't matter. The line above happens to specify type, then name, then size, but there's no rule about which has to come first or last.

Next, we need to name the 4 checkboxes, the code for which currently looks something like:

```
Eggs <input type="checkbox" />
Bacon <input type="checkbox" />
Sausages <input type="checkbox" />
Beans <input type="checkbox" />
```

Change the contents of the 4 tags to:

```
Eggs <input type="checkbox" name="cb_eggs" value="Y" />
```

```
Bacon <input type="checkbox" name="cb_bacon" value="Y" />
Sausages <input type="checkbox" name="cb_saus" value="Y" />
Beans <input type="checkbox" name="cb_beans" value="Y" />
```

Checkboxes work in a different way to a textbox. For each tag, I've specified a name and a value. If the visitor ticks the box, the specified variable name will be set to the specified value. If the visitor leaves the box unticked, the specified variable will be blank. So, for example, if the visitor ticks the "Bacon" box, $cb_bacon will be equal to "Y". If he leaves it unticked, $cb_bacon will be blank.

Finally we need to name the radio buttons. Again, these work in yet another slightly different way.

The code for the radio buttons is currently:

```
Fried <input type="radio" name="radio" />
Poached <input type="radio" name="radio" />
Scrambled <input type="radio" name="radio" />
```

Amaya has given the radio buttons a name, but we're going to change it. Change the HTML tags to:

```
Fried <input checked type="radio" name="rb_eggs" value="F" />
Poached <input type="radio" name="rb_eggs" value="P" />
Scrambled <input type="radio" name="rb_eggs" value="S" />
```

A couple of important things to note about these radio buttons. See how the first radio button (fried) includes the word "checked". This means that, when the form is displayed, this is the option that will be selected by default. You don't have to check any radio button like this but, if you do, make sure you only check one of them.

Next, see how each radio button has the same name (rb_eggs) but a different value (F, P or S). If the visitor clicks the "Fried" button, $rb_eggs will be set to "F". If he chooses "Poached", $rb_eggs will be "P" and so on. Each "set" of radio buttons on a form, ie each collection from which the visitor can make one choice, needs to have its own name that is shared among all the buttons in the set but which is not used anywhere else on the form.

As to why I've used values of F, P and S rather than "Fried", "Poached" or "Scrambled", this is mostly personal preference. And because it saves me having to remember whether the value of a radio button is Fried or FRIED or fried when I come to check it in the PHP program.

With the form fields correctly named, there's just one more thing we need to do before our form is ready. When the visitor fills in the form and clicks the Submit button, the data will be sent to a PHP program of our choice. The name of this PHP program needs to be specified in the **<form>** tag. Currently it probably just says:

```
<form action="">
```

You need to change this to:

```
<form method="POST" action="breakfast.php">
```

Now, when the user clicks Submit, the breakfast.php program will be run and the form's data sent to it.

As for **method="POST"**, this tells the web server how to send the submitted form information. You don't need to worry about how it works. Just make sure that **method="POST"** appears in every <form> tag that you create. If you don't, your form won't work.

With everything in order, we're now ready to write the breakfast.php program.

Handling Form Data and Quote Marks

Take a look at the following PHP program. Note how I've sanitized $name, deleting any character that isn't A-Z, a-z or a space. There's no need to sanitize the other input, as I'm not echoing it directly, just comparing it to certain values.

```
<?php
$name = $_POST["tb_name"];          # Diner's name
$name_san = ereg_replace("[^A-Za-z ]", "", $name);

$cb_eggs = $_POST["cb_eggs"];       # Will be Y if wants eggs
$cb_bacon = $_POST["cb_bacon"];     # Will be Y if wants bacon
$cb_sausages = $_POST["cb_saus"];   # Will be Y for sausages
$cb_beans = $_POST["cb_beans"];     # Will be Y if wants beans
$eggs_style = $_POST["rb_eggs"];    # Will be F, P or S

echo "<p>";
echo "Thank you ," . $name_san . "<br><br>";

echo "Your breakfast order is:<br><br>";
```

```
If ($cb_beans == "Y")
{
echo "beans<br>";
}

If ($cb_bacon == "Y")
{
echo "bacon<br>";
}

If ($cb_sausages == "Y")
{
echo "sausages<br>";
}

If ($cb_eggs == "Y")
{
If ($eggs_style == "F")  { echo "Fried eggs<br><br>"; }
If ($eggs_style == "P")  { echo "Poached eggs<br><br>"; }
If ($eggs_style == "S")  { echo "Scrambled eggs<br><br>"; }
}

echo "</p>";

?>
```

One thing you'll not have seen before is the use of a hash (#) symbol to signify the start of a comment. Also, note how we retrieve each form variable with **`$_POST["something"]`**.

There are two ways that data gets sent to PHP programs. One is via the URL, such as

`www.yoursite.com/breakfast.php?meal=breakfast&eggs=fried`

In this case, the PHP program must retrieve the data with $_GET. But if the data comes from a form, as it does here, you need to use $_POST instead. And yes, that's why we added **`method="POST"`** to the form tag.

Don't try changing to **`method="get"`** in your form tag, by the way, and then using $_GET to retrieve the contents of the form. It will work, sometimes, but the contents of the form will appear in the URL. There are serious security implications in doing this.

Note how I've generated **<p>** tags around the output, so that the page will neatly adopt the layout as specified in the **<p>** tag style for whatever css style sheet you decide to use. To do so, you'll need to add another set of "echo" statements which generate <head> and <body> sections, and a link to the style sheet. The link should go in the head section and looks like:

```
<LINK href="style.css" rel="stylesheet" type="text/css">
```

However, if you're going to generate this line with some PHP, note that the following won't work:

```
echo "<LINK href="style.css" rel="stylesheet" type="text/css">";
```

The problem is that the quote marks (") are being used here for 2 different purposes. The very first one and the very last one are part of the PHP code, and specify the start and end of the text you're outputting to the visitor's web browser. The others, though, are part of the text being output.

When you want to echo a literal quote to the visitor's browser, the rule is that you need to precede it with a backslash. Like this:

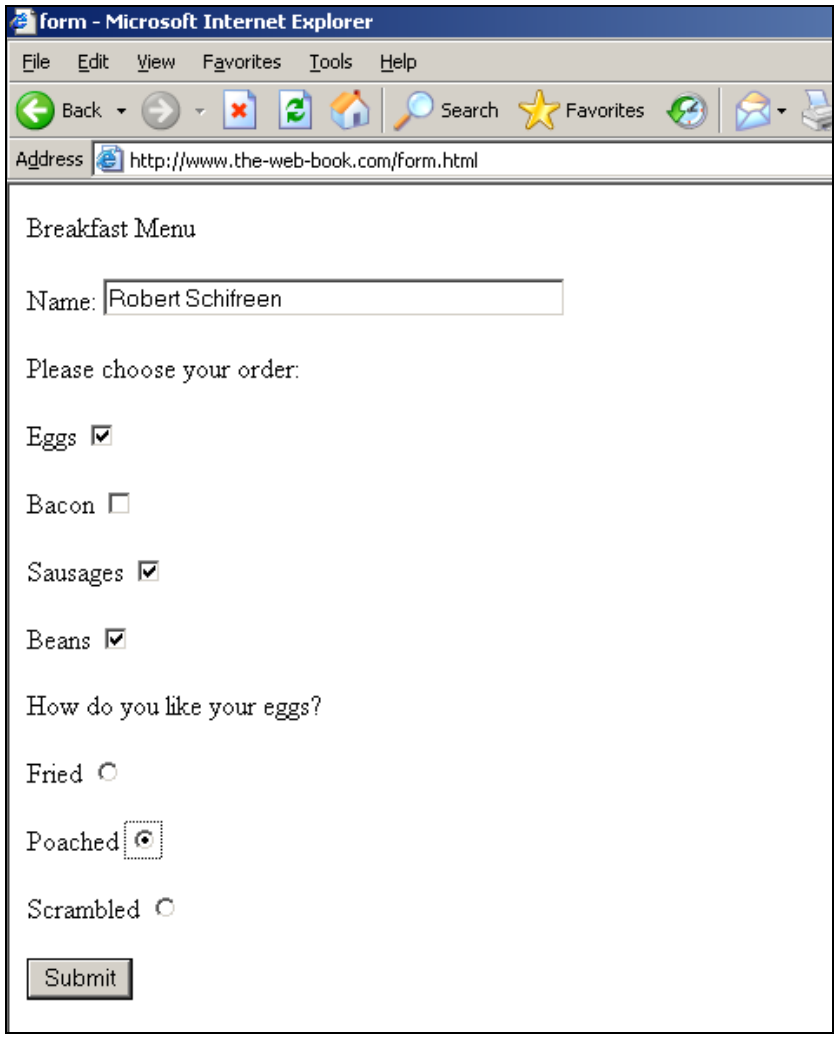
```
echo "<LINK href=\"style.css\" rel=\"stylesheet\" type=\"text/css\">";
```

Note, too, how I've used a different layout for some of the if... statements. If the visitor wants eggs, I've then tested for the type of eggs using a form that takes just one line for the comparison and the display of the message. This is useful if the amount of code you want to run as a result of the test is short, and keeps things neater.

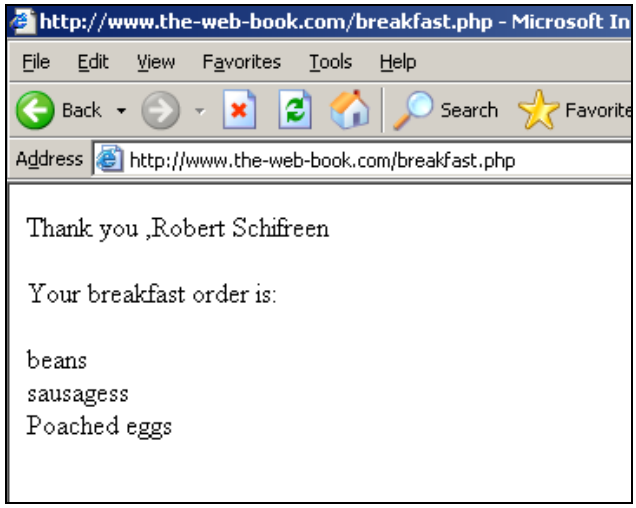
Testing The Form

Type or paste the PHP program above into a file called breakfast.php. Upload this to your web server, along with the form.html file (or whatever you called it). Make sure that both files get uploaded into the same folder on the server, and that the name of the PHP file is the same one specified in the "action" part of the form tag. On some servers this is case-sensitive, so breakfast.php is not the same as BREAKFAST.PHP.

With both files uploaded, surf to the page that contains your form and you should see this:



Check that you can type into the Name box, and that you can select an assortment of food items and a single egg type. Fill in the form as you wish, then click Submit. With luck, you should see something like this:



It works. The data from the form has been transferred to our breakfast.php program, and has been processed and displayed correctly. Plus, because form input is being sanitized, this program is also written in a secure way.

Being able to create forms, and write PHP programs to deal with the data from them, is a vital part of developing PHP-based web applications. Only by using forms can you request information from visitors to your site.

One of the most useful things about forms is that you can populate them from a database automatically, and this is something we'll cover in more detail later. But imagine, for example, that the list of breakfast items (eggs, bacon etc) is held in a database table. Having accessed the database and retrieved the list of items, you then wish to present them in a menu for the user to choose.

Here's the code we originally used, in form.html, to create the menu:

```
Eggs <input type="checkbox" name="cb_eggs" value="Y" />
Bacon <input type="checkbox" name="cb_bacon" value="Y" />
Sausages <input type="checkbox" name="cb_saus" value="Y" />
Beans <input type="checkbox" name="cb_beans" value="Y" />
```

And here's a PHP version of it, based on an array of items that we have read from the database.

```
<?php
for ($i = 0; $i <= count($items); $i++)
{
echo $items[$i];
echo "<input type=\"checkbox\" name=\"cb_" ;
echo $items[$i]  . "\" value=\"Y\" />";
}
?>
```

For an array called $items[] that contains values of "bacon", "sausage" and "beans", this code will produce:

```
<input type="checkbox" name="cb_bacon" value="Y" />
<input type="checkbox" name="cb_sausage" value="Y" />
<input type="checkbox" name="cb_beans" value="Y" />
```

Hey presto, a dynamically-created collection of checkboxes. Now, hopefully, you can start to see how PHP and MySQL work together to allow you to create powerful web-based applications. And you're probably itching to get to the database stuff. Don't worry, not long now.

Note how I've preceded quotation marks with a backslash when I want to echo them to the visitor's browser, ie when they're not part of the PHP code itself. This was discussed in more detail on page 246.

Retrieving Textarea and Dropdown Data

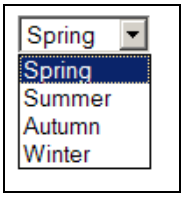
So far, I've shown you how to retrieve the contents of a textbox, checkbox and radio button using PHP. For completeness, it's worth explaining how to retrieve the value of a textarea and a dropdown too.

A textarea's content is retrieved in the same way as a textbox. Just use **$_POST["xyz"]**, where xyz is the name of the textarea as defined in the form. Remember, though, that it may contain carriage return characters if the user pressed Return while entering the contents of the textarea. Carriage returns are harmless from a security perspective, but if you don't want them you can remove them with the same type of ereg_replace command used for sanitizing. The ereg code for a carriage return character is **\r** although it's often also followed by a "line feed" character (**\n**) as well. So you need to search for **\n** and **\r** and delete them both.

As for a dropdown box, this too is easy to retrieve. Just use **$_POST[]** with the name of the dropbox as specified in the form. For example, here's the HTML code for a simple dropdown box. As you can see, it is created by a **<select>** tag, though few people ever refer to it as a select box:

```
<select name="dd_season">
<option>Spring</option>
<option>Summer</option>
<option>Autumn</option>
<option>Winter</option>
</select>
```

This produces, on a form, the following:



Using **$_POST["dd_season"]** in PHP will return the name of a season. However, there's another option that you may find useful. Consider this:

```
<select name="dd_season">
<option value="1">Spring</option>
<option value="2">Summer</option>
<option value="3">Autumn</option>
<option value="4">Winter</option>
</select>
```

Now, the values that **$_POST[]** will return are no longer Spring, Summer, Autumn or Winter. Instead, the values returned will be 1, 2, 3 or 4 respectively.

Imagine you're populating a form from a database, as described above. You want to produce a drop-down list of all the guests who are currently staying in the hotel. So you retrieve, from the database, their id number and their name. Where I've listed the names of the seasons above, you display the guest's name. But where I've shown 1,2,3,4 above, you echo the guest id number. Now, when the hotel receptionist chooses a particular name from the drop-down list, the PHP code knows the guest's database id number rather than their name. Consider this while re-reading the section on database normalization (page 183) to see just how incredibly useful this is.

Checkbox Arrays

When you're creating HTML forms, you'll often find yourself creating a large set of checkboxes. For example, keeping with the hotel management system analogy for the moment, perhaps you want to show a screen which lists each bedroom by number, along with a checkbox, so that the operator of the system can choose which rooms to mark as being in need of cleaning.

One way is to do something like this:

```
Room 1 <input type="checkbox" name="room1" value="Y" /> <br>
Room 2 <input type="checkbox" name="room2" value="Y" /> <br>
Room 3 <input type="checkbox" name="room3" value="Y" /> <br>
Room 4 <input type="checkbox" name="room4" value="Y" /> <br>
```

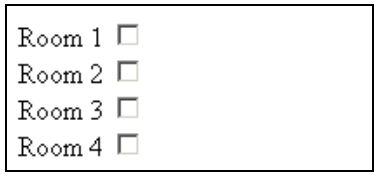
Which would work, of course, but will cause problems when the PHP code on the receiving end of the checkboxes needs to do its stuff. There might be dozens of checkbox variables to, er, check, each with a different yet slightly-similar name. There's no easy way to use a loop

and quickly scan through the values of $room1, $room2, $room3 and $room4 to find out which are blank and which are "Y".

As you might expect, there's a handy shortcut, and that's to create an array of checkboxes. You'll remember the concept of PHP arrays from page 229. Here's a version of the above HTML code that uses a checkbox array:

```
Room 1 <input type="checkbox" name="rooms[]" value="1" /> <br>
Room 2 <input type="checkbox" name="rooms[]" value="2" /> <br>
Room 3 <input type="checkbox" name="rooms[]" value="3" /> <br>
Room 4 <input type="checkbox" name="rooms[]" value="4" /> <br>
```

Note that each checkbox now has the same name, which ends with 2 empty square brackets. Note too, that each checkbox has a unique value which corresponds to the room number. The above HTML displays the following, when included in a form:

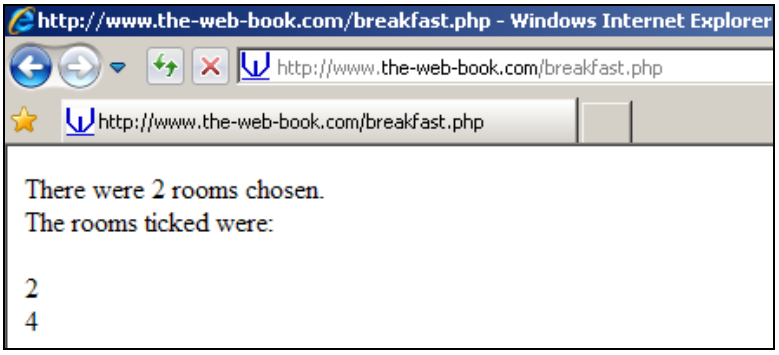


When the contents of the checkboxes is sent to the PHP code, the results will be in an array called $rooms. This array will contain ***only*** those checkboxes that were ticked (there's no way to retrieve a list of those that were not, which is sometimes annoying). The values in the array will be the same as the **value=** settings for those checkboxes. So, for example, if the visitor ticks boxes 2 and 4, the $rooms array will contain 2 elements (numbered 0 and 1, because arrays always start at 0). $rooms[0] will contain 2 (the first room selected) and $rooms[1] will contain 4.

Here's some PHP code to handle the above HTML. It steps through each of the entries in the $rooms array and displays the value:

```
$rooms = $_POST["rooms"];
$room_count = count($rooms);
echo "There were " . $room_count . " rooms chosen.<br>";
echo "The rooms you ticked were:<br>";
echo "<br>";
for ($i=0; $i <= count($rooms); $i++)
{
echo $rooms[$i] . "<br>";
}
```

And here's what it displays if you tick rooms 2 and 4:



Feedback Forms

Perhaps the classic use for an HTML form, outside of something that displays information from a database, is the feedback form. This is a great way to allow site visitors to send you a message.

Just create a form on a web page which contains a textbox for the visitor's name, and a textarea into which they can enter a brief message. When they click Submit, your PHP program retrieves the sender's name from the textbox and the message from the textarea. It can then use the mail() function in PHP to send the contents of the textarea to your email address, after adding the contents of the textbox to the start of the message.

I won't include an example of how to do this, as all of the required steps have already been covered in this book. Just use **`$_POST`** to retrieve the two items of text, and the mail function (see page 221) to send it.

If you want to allow visitors to your site to send you comments, suggestions etc, then a feedback form is the best way to do it. The only other option is to include your email address on a web page, perhaps as a hyperlink. If you do this, however, you have no control over what happens when the visitor clicks on that link. Most often, the visitor's computer will launch its standard email program (whatever that is set to). Assuming there is one, of course, otherwise nothing will happen at all. Very messy and unprofessional. By having a feedback form, you never have to divulge your email address in public. The visitor fills in the form, clicks Submit, and your PHP code sends you the message.

Hidden Fields

A form is a great way to pass information back to a PHP program from a web page. Sometimes, though, you might want to include other information in addition to the contents

of the form fields. As it happens, one excellent use of this feature is to help spam-proof a feedback form.

Consider the case of the feedback form as mentioned above. It provides an easy way for anyone to type in a message, click Submit, and have it mailed to you. The trouble is, spammers often use these forms as a way to send unwanted mail. If the form merely sends its contents to you alone, it's inconvenient. If the form posts its contents to your company's online forum, it's a much bigger problem. So here's a neat way to create a feedback form that's virtually spammer-proof.

Almost all form-based spam is sent by automated programs rather than humans. The programs surf to your feedback form page, paste some text into it, and click the submit button. All within a very short time. And that's their downfall. Spam is only worth sending if you can do it quickly, in order to be able to send sufficient quantities. Real people, on the other hand, type relatively slowly. It probably takes them at least 15 seconds, after the feedback form is displayed, before they click Submit.

So is there a way of finding out how long the user took to fill in the form? If so, we can simply discard the contents of any form that gets submitted too quickly, rather than emailing its contents to anywhere.

The answer is to use something called a hidden form field. You're probably accustomed to seeing something like this in the HTML code of a form:

```
<input type="text" size="40" />
```

Now consider this:

```
<input type="hidden" name="timecode" value="12345" />
```

This creates another form variable called **$timecode**, with a value of "12345", which will be sent back to the PHP program when the form is submitted. It doesn't correspond to any form item like a textbox or checkbox, but is what's known as a hidden field. Hidden because it doesn't appear anywhere on the form. If you ever want to send other information back to the PHP program, hidden fields are the way to do it. And one of those is the key to our spam-proofing.

In the HTML file that creates the feedback form, add a few lines of PHP as follows, before the end of the form (ie, before the </form> tag):

```
<?php
$t = time();
```

```
echo "<input type=\"hidden\" name=\"timecode\" value=\"" . $t . "\" />";
?>
```

PHP's time() function returns a long number (10 digits or so) which corresponds to the number of seconds since January 1st 1970. It's known as the timestamp, and is often used in programming to find out the current time or to do time-based calculations. For example, if you want to know what the time is going to be in 2 hours, make a call to time(), then add 7200 to the result (3600 seconds in an hour, times 2), then call one of the many PHP functions that convert a timestamp back to a date and time.

We now have a form which, in addition to the name and message, includes a hidden field called $timecode that contains the timestamp at which the form was displayed to the visitor.

When your PHP code receives the data from the form, make a call to time() and compare the value to $timecode by subtracting one from the other. The result will be the number of seconds that elapsed between the form being sent to the visitor's browser and the Submit button being clicked. If this is less than, say, 5, you can safely assume that the form was submitted by an automatic spam-sending computer rather than a human.

Accessing MySQL Databases with PHP

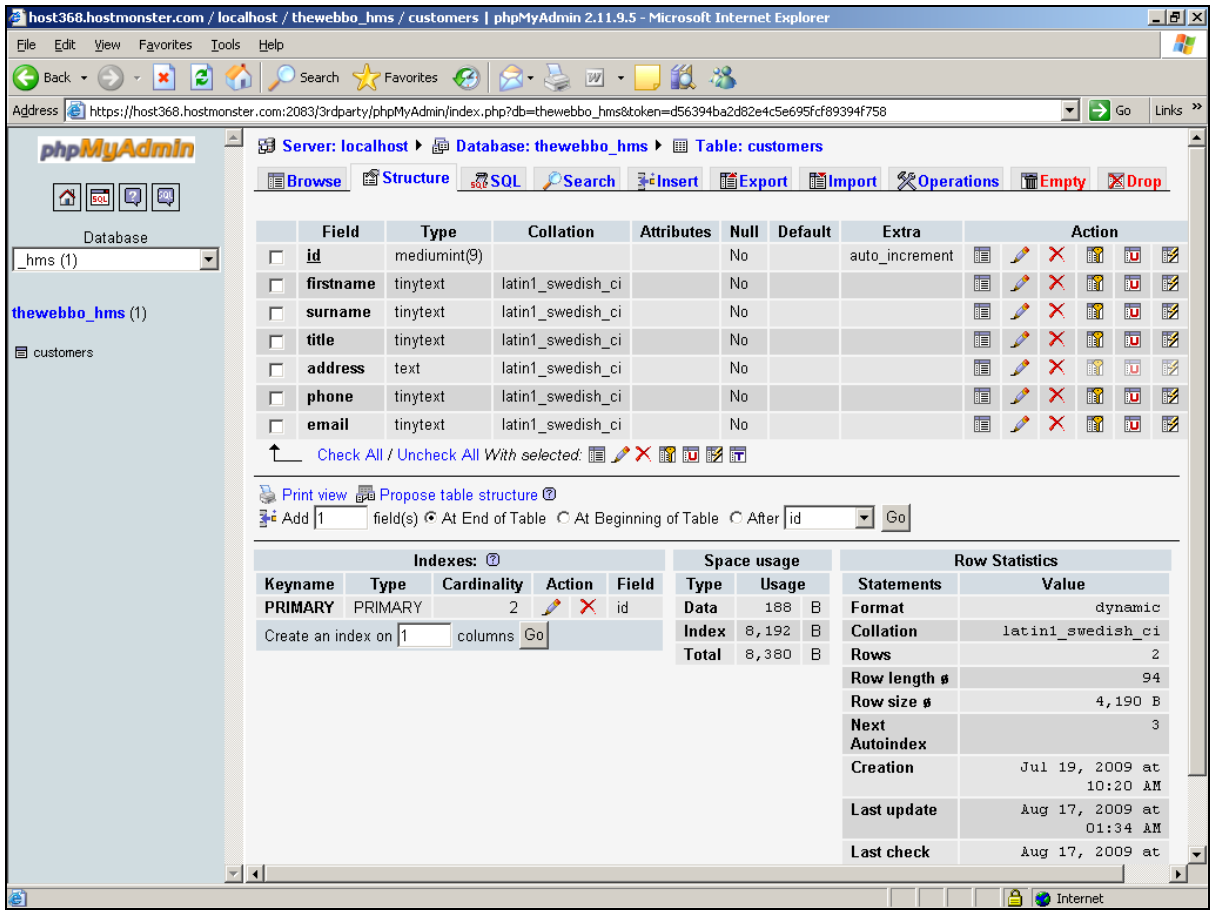
On page 190 we covered the basics of creating a web-based MySQL database and querying it via phpMyAdmin. On page 208 we covered writing programs using PHP. And on page 239 we covered HTML forms. Once you know how to do these three things (and if you have forgotten, you would be advised to recap before going further), we can now put it all together and create an online database application.

To continue, you will need to ensure that the hms database we discussed on page 187 exists on the web server, and that there's a table within it called customers. The customers table needs 7 fields:

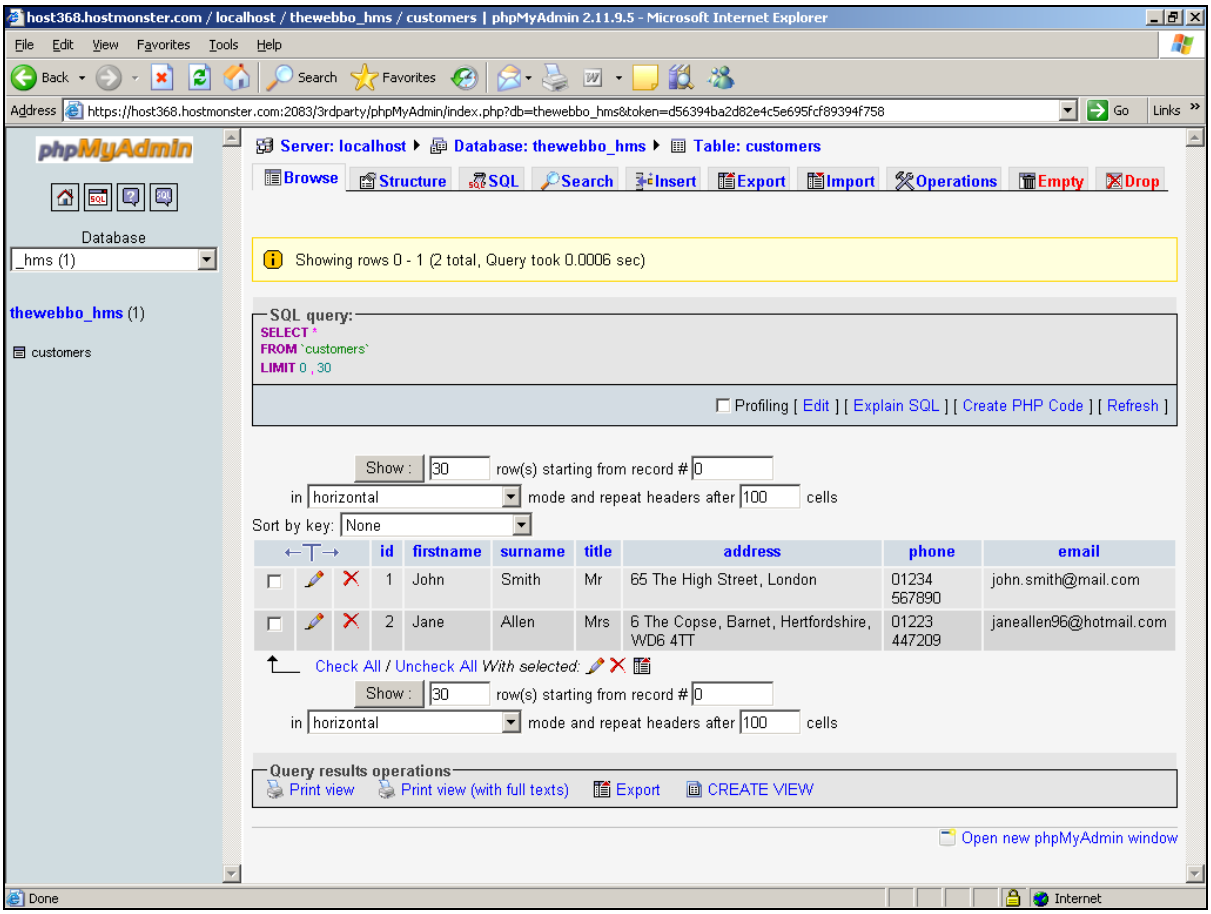
id
firstname
surname
title
address
phone
email

All of these should be text fields (tinytext should suffice) except id, which should be an integer. Further, id should be set as an auto-increment field. If you've forgotten how to do so, refer back to page 190 for coverage of phpMyAdmin.

With the database and table created, your phpMyAdmin screen should look like this when you view the structure of the Customers table:



You'll also need some data in the table. Hopefully, your data will still be there from when we covered phpMyAdmin. If not, browse to the Customers table, click on Insert, and create a couple of records. Now, when you click on Browse, you should see the data in the table like this:



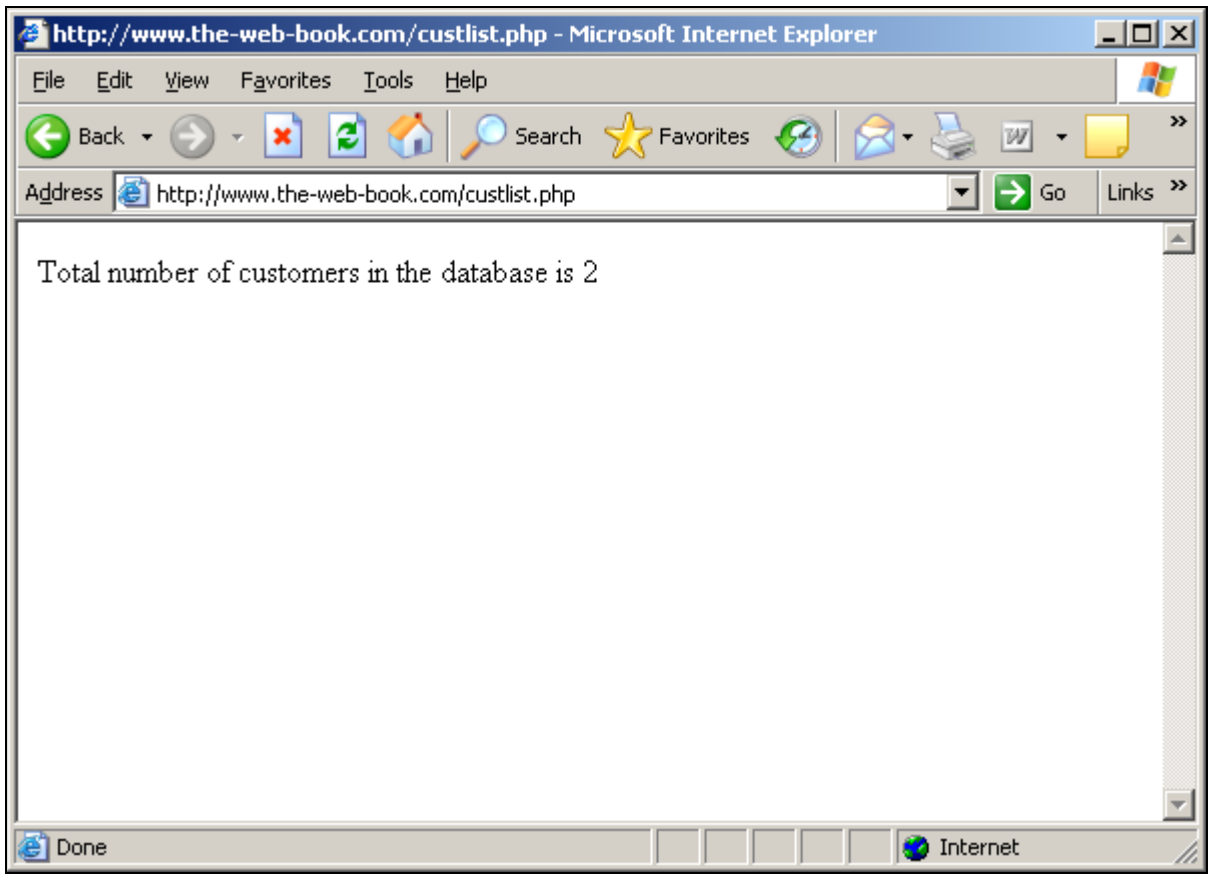
Counting Rows

Now we're ready to start writing some real code. Close your Web browser, open PSPad, and create a file in your public_html folder called custlist.php which contains the following:

```
<?php

$db_host = "localhost";
$db_database = "thewebbo_hms";
$db_username = "thewebbo";
$db_password = "abc123xyz";
$dbcnx = mysql_connect($db_host,$db_username,$db_password);
mysql_select_db($db_database);

$query = "select id from customers";
$q = mysql_query($query);
$total = mysql_num_rows($q);

echo "Total number of customers in the database is " . $total;
```

```
?>
```

Open your web browser and surf to **`www.yoursite.com/custlist.php`** and you should see the following:



For simplicity, I haven't generated <HTML>, <body> and <p> tags. Technically, therefore, the HTML page is not valid code. However, it keeps things simple for now, and just about every browser will allow it. When you come to develop real applications, make sure that you generate valid HTML pages which contain all the necessary tags.

Let's look at this code in detail.

The first 6 lines are used to connect to your database. You need to ensure that these lines are present at the start of every PHP program you write, if you intend to access a database. The first 4 lines are where you specify the hostname, database name, username and password (use the same password and username that you use for accessing phpMyAdmin or your web hosting control panel). The hostname will probably be "localhost" unless your web hosting provider stores its databases on a server other than the web server. If this is the case, you

will be advised accordingly. Make sure you change the database information accordingly. Don't simply copy and paste my example into your programs, as this won't work.

Having made the connection to our database, we can now access it. This takes just 4 lines of code, as follows.

```
$query = "select id from customers";
```

This creates our MySQL query (you should recall these from when we queried the database directly with phpMyAdmin). We store the text of the query in a variable called $query. In this example, we want to select all the id values from the customers table.

```
$q = mysql_query($query);
```

Now we execute the actual query. The **mysql_query** function performs the query, which we have previously stored in the $query variable. Note how we assign the result of the query to the $q variable.

```
$total = mysql_num_rows($q);
```

Having performed the query, we now use the **mysql_num_rows()** function, which returns the total number of database records (ie, rows) which were returned in the $q query. That is, the total number of id values. At this point we haven't actually retrieved any data. All we have is the total number of rows. Which is all we need right now, so let's display it:

```
echo "Total number of customers in the database is " . $total;
```

Reading Data

We now have a complete PHP program which can access a MySQL database. So let's adapt it to display some real data. Change the program to the following:

```
<?php

$db_host = "localhost";
$db_database = "thewebbo_hms";
$db_username = "thewebbo";
$db_password = "xyzabc123";
$dbcnx = mysql_connect($db_host,$db_username,$db_password);
mysql_select_db($db_database);

$query = "select firstname,surname from customers";
$q = mysql_query($query);
```

```
echo "These are the customers on file:<br><br>";

while ($row = mysql_fetch_array($q))
{
echo $row["firstname"];
echo " ";
echo $row["surname"];
echo "<br>";
}

?>
```

As before, we start with the standard 6 lines for connecting to our database. Then we create a database query (**`select firstname, surname from customers`**) and execute it. Note how we specify which fields we need. In theory you can just say **`select * from customers`**, which would return all 7 fields. But it's good practice to only request fields that you're actually going to use. It reduces the load on the database server.

When you perform a query that returns data, as we have just done, you then need to loop through the array of rows that the query returns. That's what the **`mysql_fetch_array`** code does:

```
while ($row = mysql_fetch_array($q))
{
echo $row["firstname"];
echo " ";
echo $row["surname"];
echo "<br>";
}
```

Here, we use a **`while`**... loop, which returns an array of fields called **`$row[]`** for each row of the database returned by the query. The names in square brackets match the field names in the table. So in this example, for each matching row, we echo the row's firstname field, then a space, then the row's surname field, then a line break. This produces a neat listing of every customer's full name.

You don't have to use $row as the array name, of course. You could call it $record, $customer, or whatever else you choose. But many PHP programmers tend to call it $row, so that's what I use.

Try adapting this program so that it also prints out the customer's email address. Remember to adjust the database query so that it returns 3 fields rather than just 2.

Also, try changing the query to:

```
select firstname,surname from customers where id=3
```

This time, notice how just one matching row gets returned. Remember that id is a numeric field, so there's no need for quotes round the number 3.

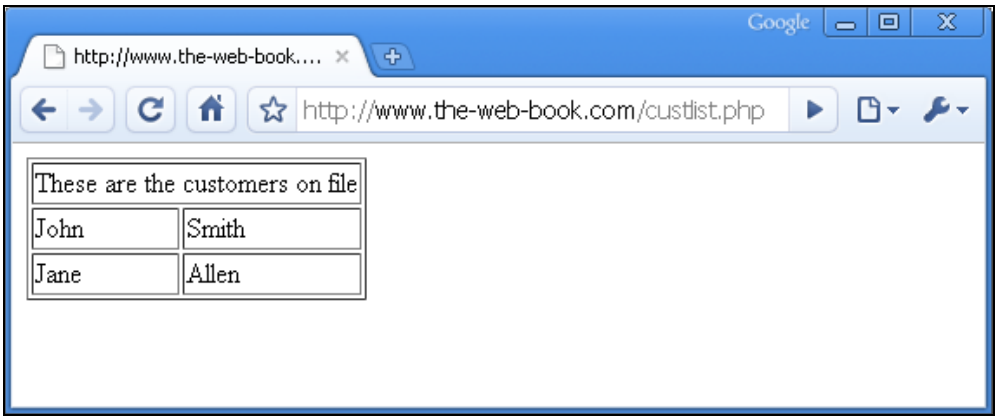
Let's neaten up the output slightly by using a table. Here's a new version of the program. Again, I've kept the HTML tags to a bare minimum in order to make the listing as short as possible for clarity. When you do this for real, you must include HTML and body tags. Plus, you'll probably want to create a style sheet that specifies the design of <p> or <td> tags.

```
<?php

$db_host = "localhost";
$db_database = "thewebbo_hms";
$db_username = "thewebbo";
$db_password = "xxxx";
$dbcnx = mysql_connect($db_host,$db_username,$db_password);
mysql_select_db($db_database);

$query = "select firstname,surname from customers";
$q = mysql_query($query);

echo "<table border=1>";

echo "<tr>";
echo "<th colspan=\"2\">These are the customers on file</th>";
echo "</tr>";

while ($row = mysql_fetch_array($q))
{
echo "<tr>";
echo "<td>" . $row["firstname"] . "</td>";
echo "<td>" . $row["surname"] . "</td>";
echo "</tr>";
}

echo "</table>";

?>
```

This time, we start by creating a table and setting a border width. There are lots of other useful options that you can add to the table tag, or specify with a css style, but this will do for now. Next, we create a table header row. Note the use of colspan, to indicate that this row spans both columns of the table. And see how I've used a **th** tag for the header row cell, rather than the standard td tag in every other row. This will allow me to specify a different design for the header row when I get around to creating the css file.

Finally, within the while... loop that brings in each row of data from the table, I create an HTML table row.

The end result is:



Not particularly pretty, but all of the building blocks are there. All that remains is to create a css file which contains entries for the **p**, **th** and **td** tags, and generate another line of HTML which attaches it. I shall leave this to you. One of the great things about CSS is that you can design the structure of your site independently from its look. Once the PHP code is working, and returning the information you need, then you can work on the css to make it look pretty. So long as the PHP code simply generates tags such as **th**, **td** and **p**, you can then control the appearance of your site via a css file without needing amend the PHP code at all.

Before we leave the topic of reading data, here's one more neat example.

```
<?php

$db_host = "localhost";
$db_database = "thewebbo_hms";
$db_username = "thewebbo";
$db_password = "xyzxyz";
$dbcnx = mysql_connect($db_host,$db_username,$db_password);
mysql_select_db($db_database);
```

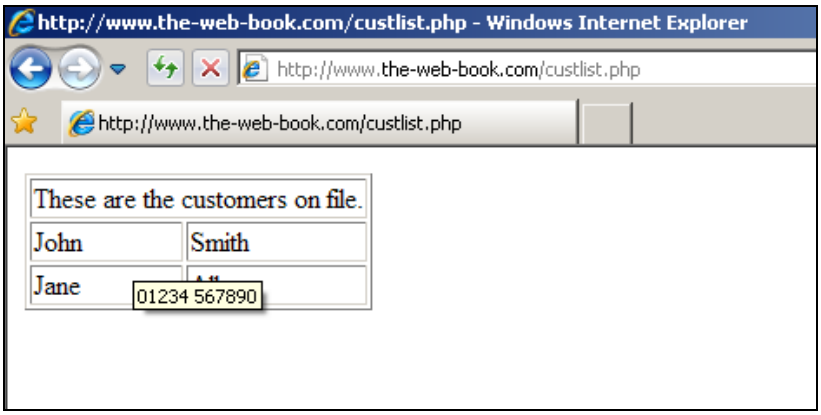
```
$query = "select firstname,surname,phone from customers";
$q = mysql_query($query);

echo "<table border=1>";

echo "<tr>";
echo "<td colspan=\"2\">These are the customers on file.</td>";
echo "</tr>";

while ($row = mysql_fetch_array($q))
{
echo "<tr>";
echo "<td title=\"" . $row["phone"] . "\">";
echo  $row["firstname"] . "</td>";
echo "<td>" . $row["surname"] . "</td>";
echo "</tr>";
}

echo "</table>";

?>
```

It's only a couple of minor changes to the code. First, I've amended the query to return the phone number field as well as firstname and surname. Then, when I create the table row that contains the first name of the customer, I also add a "title" to the td tag, the contents of which is the phone number field.

For example, the generated HTML code for John Smith with a phone number of 1234 would be:

```
<td title="1234">John</td><td>Smith</td>
```

The "title" parameter in a **td** tag creates pop-up text which appears when you hover your mouse over the cell. So now, just hover the mouse over someone's first name and their phone number magically appears. Like this:

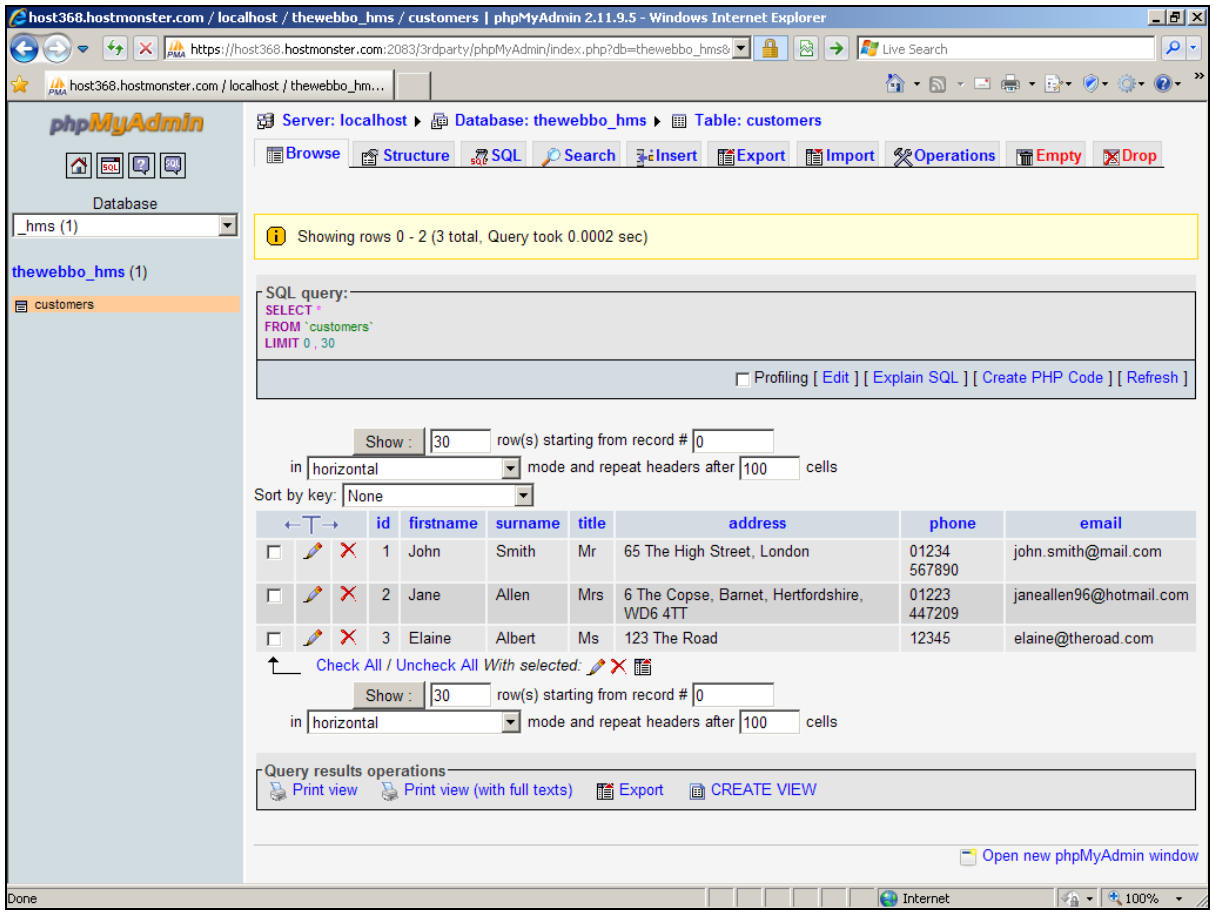


Searching A Table

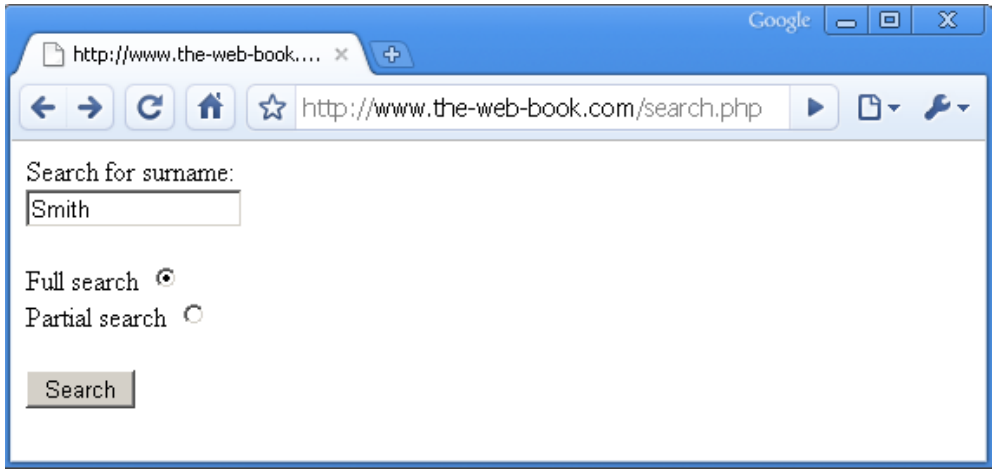
Now we'll write a simple PHP application that allows the user of the site to search the customer database for a specific surname. This is the sort of thing that a hotel receptionist might do if they're using the Hotel Management System that we're developing.

Before you go any further, you need to use phpMyAdmin to ensure that at least 2 records in the customer database share at least 2 consecutive letters. This is because we want to add a partial search facility, and it won't work if there's no matching data to return.

As you can see from the screen shot below, we have 2 customers with surnames of Allen and Albert. We should then be able to search on a partial match of "Al" and return both names.



With the necessary test data in place, we can put the application together. Before we look at the code, here's the app in action. First, the search form:



We fill in a surname, choose Full or Partial search, and click the button. In this instance, John Smith's details are displayed:

Now let's try a partial search. We'll search for "Al" and select the relevant radio button. As expected, both of the matching records are returned:

To show how this all works, here's the complete PHP code for the application:

```
<?php

# Start the page properly
echo "<HTML>";
echo "<body>";

# Check whether the searchtype radio button has been set

# If not set, display the search form.
if (!isset($_POST["searchtype"]))
{
echo "<form method=\"POST\" action=\"search.php\">";
echo "Search for surname:<br>";
echo "<input type=\"text\" name=\"searchtext\" size=\"15\">";
echo "<br><br>";
```

```
echo "Full search ";
echo"<input type=\"radio\" value=\"FULL\" checked name=\"searchtype\"><br>";
echo "Partial search ";
echo "<input type=\"radio\" name=\"searchtype\" value=\"PARTIAL\">";
echo "<br><br>";
echo "<input type=\"submit\" value=\"Search\" name=\"submit\">";
echo "</form>";
} # if

else   # Searchtype was set, so retrieve form data and do the search

{
$searchtext = $_POST["searchtext"]; # Retrieve from the form
$searchtype = $_POST["searchtype"]; # Retrieve from the form
$searchtext_san = sanitize_form_text($searchtext); # Prevents SQL injections!

# Now connect to the database
$db_host = "localhost";
$db_database = "thewebbo_hms";
$db_username = "thewebbo";
$db_password = "xxyyzz";
$dbcnx = mysql_connect($db_host,$db_username,$db_password);
mysql_select_db($db_database);

# Construct the appropriate query
if ($searchtype == "FULL")
{
$query = "select firstname,surname from customers where surname='$searchtext_san'";
} # if

if ($searchtype == "PARTIAL")
{
$query = "select firstname,surname from customers ";
$query .= "where surname LIKE '%$searchtext_san%'";
} # if

# Now do the query
$q = mysql_query($query);

$total = mysql_num_rows($q);

if ($total == 0)
```

```
{ echo "Sorry, no matches found."; }

if ($total > 0)
{

while ($row = mysql_fetch_array($q))
{
echo $row["firstname"] . " " . $row["surname"] . "<br>";

} # while
} # if matches found
} # else

# End the page properly
echo "</body>";
echo "</HTML>";

exit();

function sanitize_form_text($t)
{
$t = strip_tags($t);
$t = ereg_replace("[^A-Za-z0-9@._-]", "", $t);
return $t;
}

?>
```

If you've been reading through this book in sequence, then hopefully you'll be able to understand most of the above. Here are some points to help you.

- The program displays one of two web pages. The first time around, it displays the search form which, when submitted, launches the same php program again. The form contains a textbox called searchtext, and a pair of radio buttons called searchtype with possible values of FULL or PARTIAL. If the searchtype has been set (to either PARTIAL or to FULL) at the start of the program, this indicates that the form has been submitted, so, rather than displaying the form again, we do the actual search instead.
- I've created a function called sanitize_form_text, which is listed at the bottom of the program. At the point where the value of the search text is retrieved from the form, it then gets sanitized.

- Depending on the value of the radio button, we then execute one of 2 MySQL database queries. For the full search, ie where we only return complete matches, the query boils down to:

  ```
  select firstname,surname from customers where surname='xyz'
  ```

  where xyz is the search text entered in the form. Because surname is a text field, you need quotes round the item you're searching for. Using single quotes is easiest, because they don't conflict with the double quotes that make up part of the PHP code.

  The MySQL query for the partial search ends with:

  ```
  …from customers where surname LIKE '%xyz%'
  ```

  LIKE %xyz% will match surnames which contain xyz anywhere within them.

Remember that a MySQL query will return the data in a random order unless you specify otherwise. So if you want the results sorted by surname, say, add `order by surname` to the end of the query.

Preventing SQL Injection Attacks

This section is vitally important. Please read it carefully.

Although writing web-based database applications such as this one isn't particularly difficult to do, it's vital that you pay careful attention to security. Hackers love nothing more than trying to find ways to steal, delete or corrupt the contents of MySQL database tables. And sadly, it's a lot easier than you might imagine.

Take the example above, where we allow the site visitor to enter some search text, and we then pass the result of that search text to a database query. If the visitor enters some search text of Smith, then our query ends up as:

```
select firstname,surname from customers where surname='Smith'
```

No problem there. But what if the visitor enters some search text as follows:

```
Smith' or surname != 'Smith
```

This might look weird at first. But paste it into the query above, in place of the "permitted" surname, and you end up with the following:

```
select firstname,surname from customers where surname='Smith' or
surname != 'Smith'
```

The **!=** symbol means "not equal to". So now we have a query which returns every record from the table where surname **is** Smith or where surname is **not** Smith. In other words, it will return the entire contents of the table! Imagine a guest being able to display every customer's record in addition to their own. Or a secure login facility where the visitor is only granted access "*if the password is correct or the password is not correct*".

Here's another example. Consider what happens if the following is entered as a surname:

```
Smith' or surname != 'Smith; delete from customers
```

The semicolon is the standard character in MySQL for separating multiple commands on a single line. So now, after your program searches for the entered surname, it will then delete the entire contents of your customer database!

By preventing unwanted characters from being passed to the MySQL query, we prevent SQL injection attacks. So, this needs to be your golden rule when writing database applications: if you are going to use untrusted data as part of a query, sanitize that data first. At the very least, do not allow symbols such as < > ! '; and so on. Ideally, allow only letters and numbers, plus anything else that is specifically required. For example, if you're asking someone for their email address you'll need to allow a dot, dash, underscore and the "@" symbol.

Do not underestimate the importance of protecting against injection attacks on your system. Hackers have sophisticated tools that automatically look for such errors on web sites and try to exploit them. Therefore, always sanitize your text first, and always use the _san version within a query.

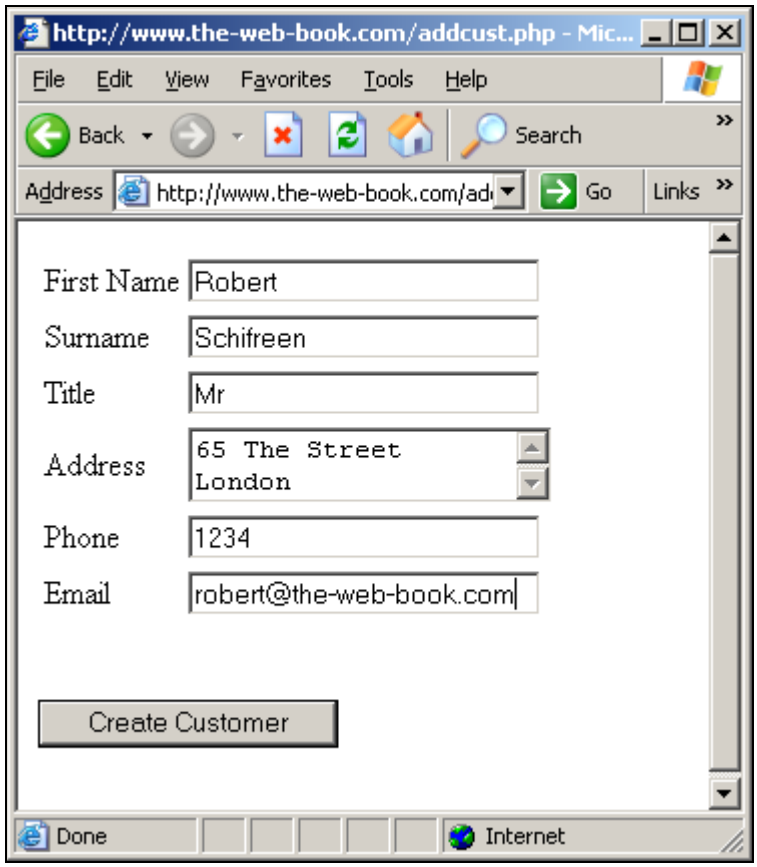
Remember, too, that you should only allow the characters that you actually need. Otherwise, you risk inadvertently allowing characters which can lead to problems. For example, %3B is the hex code for a semicolon so, if you block the user from entering semicolons, they can still delete your customer database by using %3B instead. The cure? Block the % symbol too.

Adding Data to a Table

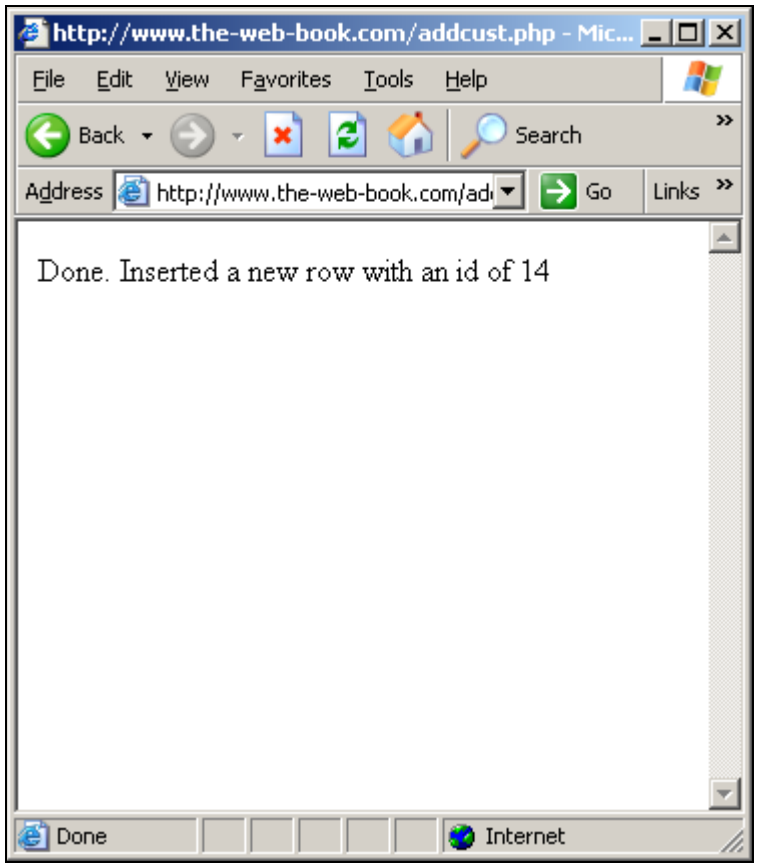
Now that we can read and search a MySQL table, it's about time that we added the ability to store new data records too. This will allow us to stop using phpMyAdmin to insert data, as we'll be able to use our own program instead.

Let's write a little application that allows us to create a new customer in the existing customers table. We'll start by displaying a form with space for the 6 fields (firstname, surname, title, address, phone and email). Then, when the user presses the Submit button, we'll add the contents of those fields into a new database record (or, to use the correct terminology, into a new table row). We'll also need to create a new id, of course. However, MySQL will take care of that automatically, as the id field is set to auto_increment.

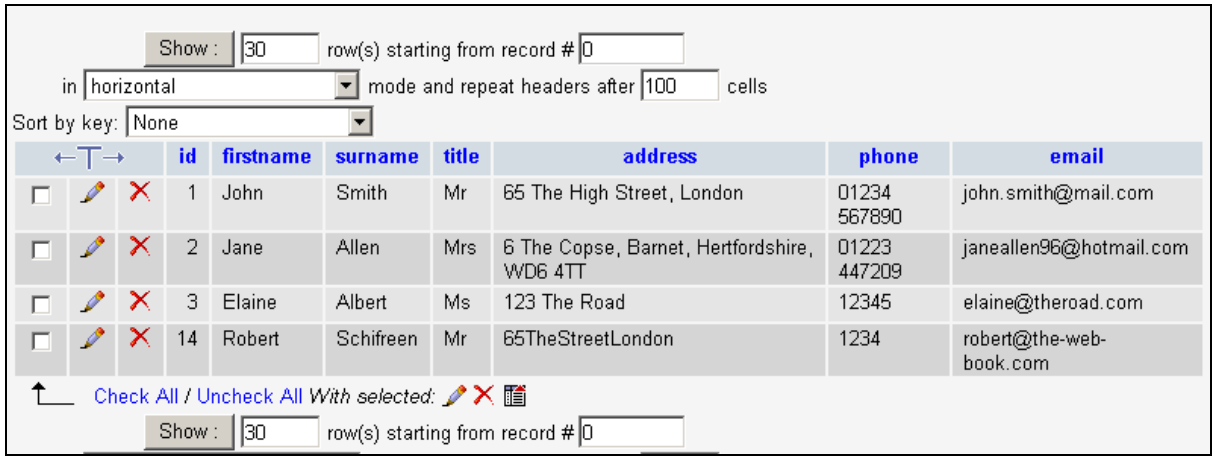
Here's the application in action. First, the data entry form:



Having filled in the form, I click the button and see the following:



Neat. Not only has the data from the form been inserted, but we also know the id that was generated for the auto_increment id field. Just to make sure it really did work, let's pop into phpMyAdmin and check that a new customer record has indeed been created:



Excellent, although there's one minor bug immediately apparent. The spaces and carriage return characters have been stripped out of the address, which has been stored as 65TheStreetLondon. Can you work out why, and what needs to be done in order to fix this?

One other point worthy of mention here. Why do the id's go 1, 2, 3, 14 instead of 1,2,3,4?

While I was writing and debugging the addcust.php program that you're about to see, records with ids from 4 to 13 were created. I subsequently used phpMyAdmin to delete those records. And yet, next time MySQL auto-generated an id, it used 14 rather than going back to 4.

There's no rule as to how auto-increment fields are generated. All that you're guaranteed is that they will be unique, ie that there won't be two customers with the same id. You certainly should never assume that they will be consecutive. So, for example, attempting to count the number of customers in the table by looking for the one with the highest-numbered id simply will not be reliable.

Here's the full PHP code for the addcust.php application:

```
<?php

# ADDCUST.PHP - Add a new customer to the customers table

# Check whether there's a surname specified.
# If not, then just display the form.

if (!isset($_POST["tb_surname"])) # if no surname specified
{
echo "<HTML>";
echo "<body>";
echo "<form method=\"POST\">";
echo "<table>";
echo "<tr>";
echo "<td>First Name</td>";
echo "<td><input type=\"text\" name=\"tb_firstname\" size=\"25\"></td>";
echo "</tr>";
echo "<tr>";
echo "<td>Surname</td>";
echo "<td><input type=\"text\" name=\"tb_surname\" size=\"25\"></td>";
echo "</tr>";
echo "<tr>";
echo "<td>Title</td>";
echo "<td><input type=\"text\" name=\"tb_title\" size=\"25\"></td>";
echo "</tr>";
echo "<tr>";
echo "<td>Address</td>";
echo "<td><textarea rows=\"2\" name=\"ta_address\" cols=\"20\"></textarea></td>";
```

```
echo "</tr>";
echo "<tr>";
echo "<td>Phone</td>";
echo "<td><input type=\"text\" name=\"tb_phone\" size=\"25\"></td>";
echo "</tr>";
echo "<tr>";
echo "<td>Email</td>";
echo "<td><input type=\"text\" name=\"tb_email\" size=\"25\"></td>";
echo "</tr>";
echo "</table>";
echo "<br><br>";
echo "<input type=\"submit\" value=\"Create Customer\" name=\"button\">";
echo "</p>";
echo "</form>";
echo "</body>";
echo "</HTML>";
} # if no surname specified

else
{ # a surname was specified so create new record

# Retrieve the data from the form
$title = $_POST["tb_title"];
$firstname = $_POST["tb_firstname"];
$surname = $_POST["tb_surname"];
$address = $_POST["ta_address"];
$phone = $_POST["tb_phone"];
$email = $_POST["tb_email"];

# Now sanitize everything
$title = sanitize_form_text($title);
$firstname = sanitize_form_text($firstname);
$surname = sanitize_form_text($surname);
$address = sanitize_form_text($address);
$phone = sanitize_form_text($phone);
$email = sanitize_form_text($email);

# Now connect to the database
$db_host = "localhost";
$db_database = "thewebbo_hms";
$db_username = "thewebbo";
$db_password = "xyz";
```

```
$dbcnx = mysql_connect($db_host,$db_username,$db_password);
mysql_select_db($db_database);

# Now construct the MySQL query to create the new record
$query = "insert into customers ";
$query .= "(firstname,surname,title,address,phone,email) ";
$query .= "values (";
$query .= "'$firstname','$surname','$title','$address','$phone','$email')";
$q = mysql_query($query);
$new_row = mysql_insert_id();

echo "<HTML>";
echo "<body>";
echo "Done.  Inserted a new row with an id of " . $new_row;
echo "</body>";
echo "</HTML>";

} # else

exit(); # We're done.  Nothing below except functions.

function sanitize_form_text($t)
{
$t = strip_tags($t);
$t = ereg_replace("[^A-Za-z0-9@._-]", "", $t);
return $t;
}

?>
```

Almost everything should be familiar here, as the structure is very similar to the previous search example. The most important new feature is the way we insert a new row into a table. Once you're connected to the database, the syntax of the MySQL query is:

```
insert into table (field1,field2) values ('value1','value2')
```

We then embed it into a PHP **mysql_query** function call. In the above example, I've built up the query string over 4 lines. This is purely for clarity, including the way that the listing looks when printed. You don't have to do it like that if you don't want to.

The two sets of items in brackets correspond to the field names you want to insert, and the values for them. You don't need to specify every field – if you don't, ones you omit will be

left blank unless you specified a default value when you created the table initially. Note that we didn't bother specifying a value for the id field – this was generated automatically because it's set to auto_increment. Also note the single quote marks around the values, which are required if you're inserting data into a text field as opposed to a numeric one.

There's a handy PHP function called **mysql_insert_id()**, which returns the value of the last auto_increment that MySQL generated. After inserting the record, we call this function in order to find the value, and then we display it. You might want to adapt the program so that, once the row has been inserted, you then retrieve it from the database and display all of the new customer's data for verification. Knowing the id makes this easy – you can just do something like:

```
$new_id = mysql_insert_id();
$query =  "select * from customers where id=$new_id";
$q = mysql_query($query);
```

If you are executing a MySQL query which you know will only be returning precisely one record (not zero, and not more than one), you don't need a while... loop in order to skip through the results. Instead, you can simply do:

```
$row = mysql_fetch_array($q);
echo $row["firstaname"];
echo $row["surname"];
```

and so on. There is only ever going to be one array to fetch, so a loop is unnecessary.

It's worth pointing out that I didn't generate all of the HTML code for the data entry form by hand. I used a separate HTML editor. Once I'd designed the form and it looked the way I wanted it, I then copied the code into the PHP file and added the **echo** statements around it, as well as changing all the HTML double quote marks to **\"** in order to distinguish them from the ones that are part of the PHP code. Using an HTML editor to mock up page which then become PHP programs is common practice among developers. Don't assume that you have to write all your HTML by hand.

Editing a Data Record

So far, we've covered reading data from a table, searching a table, and adding new records. Next, you'll need to know how to edit an existing record. Just as with using **SELECT FROM** to read data and **INSERT INTO** to create new records, existing a record is just as easy. It's simply a matter of knowing how to use an **UPDATE**...**SET** query.

A typical edit might be:

```
update customers set phone='1234' where id=25
```

This sets the phone number field to '1234' for the customer whose id is 25.

Another example might be the case of Jane Allen who calls to say that she's recently married and her surname is now Chambers. To update her entry in the database, you would probably use:

```
update customers set surname='Chambers' where surname='Allen'
```

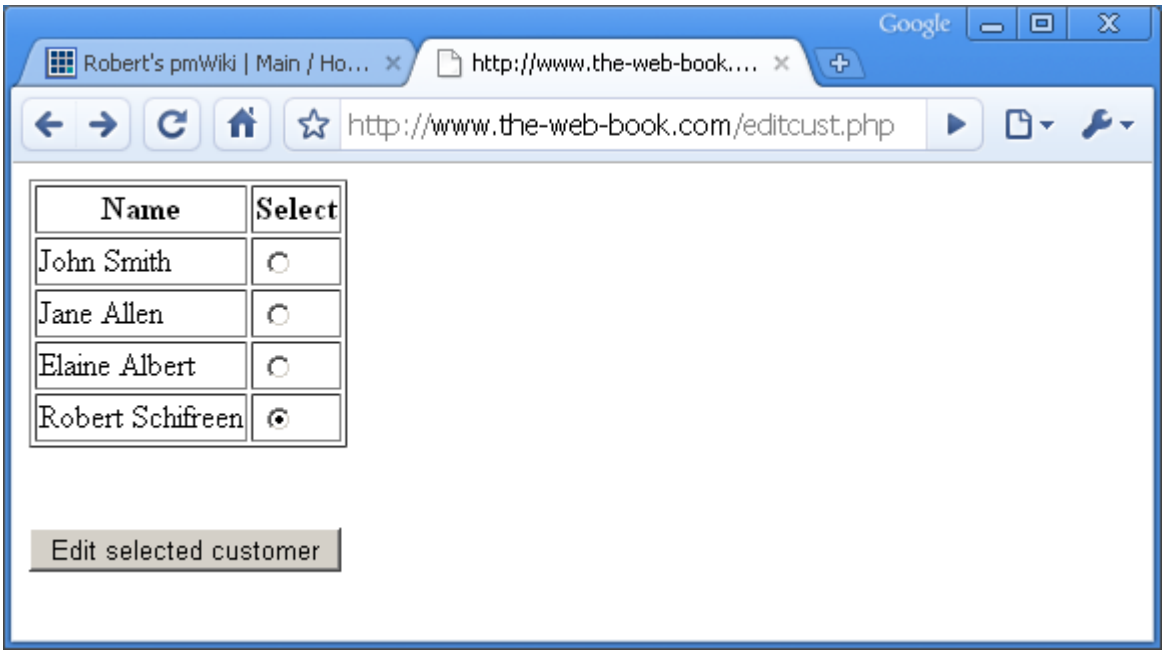
Which, at first glance, might appear to be sensible. It would certainly work, in that it would change Jane Allen to Jane Chambers. Trouble is, it'll change the surname of every 'Allen' in the customers table, and not just Jane. One solution is to do the following instead:

```
update customers set surname='Chambers' where surname='Allen' and firstname='Jane'
```

This will probably work. But again, there might be more than one Jane Allen in the database, so we need a more robust, reliable solution.

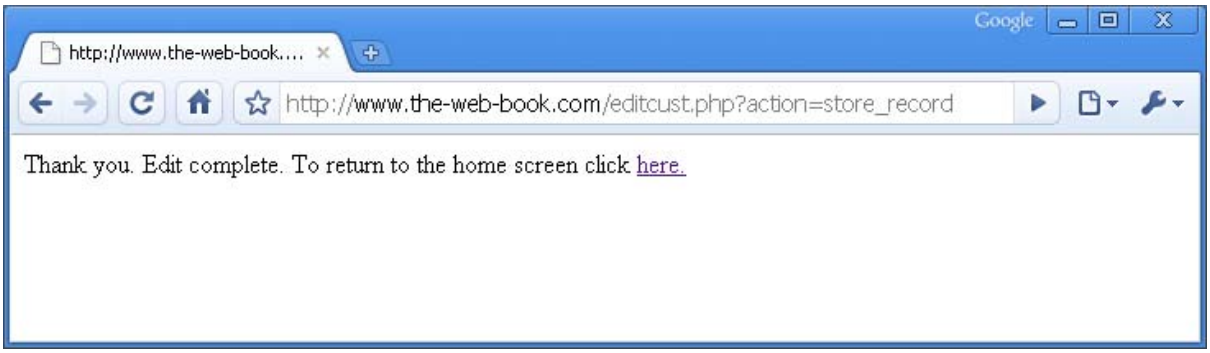
You will recall that we designed our database table so that every entry, ie every row, has a unique id number. Being able to address a particular row by its id is very useful, especially when you're editing or deleting a row and you need to be sure that you're addressing the right one.

Here's how I've chosen to design my editcust.php application. First, we display a list of every customer's firstname and surname, along with a radio button like this:



The user chooses a name to edit (Robert Schifreen in this case), clicks the button and is presented with the customer's details in a pre-filled form:

Now, we can just edit the record as required, then hit the Save button. The database is amended and a final screen appears:



As to how we make sure that the right Robert Schifreen is being edited, that's down to you. Perhaps, when displaying the list of names to edit, you should also include the address or the postcode. Anyway, here's the PHP customer editing application in full:

```
<?php

# EDITCUST.PHP - Allow the user to edit a customer record

# Connect to our database. We'll need this regardless.
$db_host = "localhost";
$db_database = "thewebbo_hms";
$db_username = "thewebbo";
$db_password = "xyzabc123";
$dbcnx = mysql_connect($db_host,$db_username,$db_password);
mysql_select_db($db_database);

# Now check the action parameter from the URL to see what we need to do
$action = $_GET["action"];

if ($action == "")  # No action specified so show the home page
{
$query = "select id,firstname,surname from customers";
$q = mysql_query($query);

echo "<form method=\"post\" action=\"editcust.php?action=show_record\">";

echo "<table border=1>";

# Create the table top row
echo "<tr>";
echo "<th>Name</th><th>Select</th>";
echo "</tr>";
```

```
while ($row = mysql_fetch_array($q))
{
echo "<tr>";
echo "<td>" . $row["firstname"] . " " . $row["surname"] . "</td>";
echo "<td>";
echo "<input type=\"radio\" value=\"" . $row["id"] . "\" name=\"id2edit\">";
echo "</td>";
echo "</tr>";
}

echo "</table>";
echo "<br><br>";
echo "<input type=\"submit\" value=\"Edit selected customer\" name=\"button\">";

echo "</form>";

exit();
} # action = ""

if ($action == "show_record")
{
# Display the customer record form.  Populate it with the details of
# the customer whose id is passed in the id2edit radio button.

# Get the contents of the id2edit form variable
$id2edit = $_POST["id2edit"];

# Now get the customer's details as we'll need them to populate the form
$query = "select * from customers where id=$id2edit";
$q = mysql_query($query);
$row = mysql_fetch_array($q); # don't need a while loop as there's only 1 row

echo "<HTML>";
echo "<body>";
echo "<form method=\"POST\" action=\"editcust.php?action=store_record\">";
echo "<table>";
echo "<tr>";
echo "<td>First Name</td>";
echo "<td><input value=\"" . $row["firstname"] . "\" type=\"text\" ";
echo "name=\"tb_firstname\" size=\"25\"></td>";
echo "</tr>";
```

```
echo "<tr>";
echo "<td>Surname</td>";
echo "<td><input value=\"" . $row["surname"] . "\" type=\"text\" ";
echo "name=\"tb_surname\" size=\"25\"></td>";
echo "</tr>";
echo "<tr>";
echo "<td>Title</td>";
echo "<td><input value=\"" . $row["title"] . "\" type=\"text\" ";
echo "name=\"tb_title\" size=\"25\"></td>";
echo "</tr>";
echo "<tr>";
echo "<td>Address</td>";
echo "<td><textarea rows=\"2\" name=\"ta_address\" cols=\"20\">";
echo $row["address"];
echo "</textarea></td>";
echo "</tr>";
echo "<tr>";
echo "<td>Phone</td>";
echo "<td><input value=\"" . $row["phone"] . "\" type=\"text\" ";
echo "name=\"tb_phone\" size=\"25\"></td>";
echo "</tr>";
echo "<tr>";
echo "<td>Email</td>";
echo "<td><input value=\"" . $row["email"] . "\" type=\"text\" ";
echo "name=\"tb_email\" size=\"25\"></td>";
echo "</tr>";
echo "</table>";
echo "<br>";
echo "<input type=\"submit\" value=\"Save\" name=\"button\">";
echo "</p>";

# Pass the id along to the next routine in a hidden field
echo "<input type=\"hidden\" name=\"id2edit\" value=\"" . $id2edit . "\">";

echo "</form>";
echo "</body>";
echo "</HTML>";
exit();
} # action = show_record

if ($action == "store_record")
{
```

```
# Retrieve the data from the form and update customer record

$id2edit = $_POST["id2edit"]; # Get the id from the hidden field
$title = $_POST["tb_title"];
$firstname = $_POST["tb_firstname"];
$surname = $_POST["tb_surname"];
$address = $_POST["ta_address"];
$phone = $_POST["tb_phone"];
$email = $_POST["tb_email"];

# Now sanitize everything
$title = sanitize_form_text($title);
$firstname = sanitize_form_text($firstname);
$surname = sanitize_form_text($surname);
$address = sanitize_form_text($address);
$phone = sanitize_form_text($phone);
$email = sanitize_form_text($email);

# Now we can update the customer's data, one column at a time.
$x = mysql_query("update customers set firstname = '$firstname' where id=$id2edit");
$x = mysql_query("update customers set title = '$title' where id=$id2edit");
$x = mysql_query("update customers set surname = '$surname' where id=$id2edit");
$x = mysql_query("update customers set address = '$address' where id=$id2edit");
$x = mysql_query("update customers set phone = '$phone' where id=$id2edit");
$x = mysql_query("update customers set email = '$email' where id=$id2edit");

echo "<HTML>";
echo "<body>";
echo "Thank you.  Edit complete. To return to the home ";
echo "screen click <a href=\"editcust.php\">here.</a>";
echo "</body>";
echo "</HTML>";
exit();
} # action = store_record

function sanitize_form_text($t)
{
$t = strip_tags($t);
$t = ereg_replace("[^A-Za-z0-9@ ._-]", "", $t);
return $t;
}
```

```
?>
```

Once again, this example is arranged in a single file (editcust.php) and all in PHP mode (ie, without flipping back into HTML mode). Remembering that the Web is a stateless system, where each page has no idea what came before it, I have used a parameter on the URL to maintain state. You can read more about this on page 295.

The program starts by performing the necessary code to connect to the database, as this will be needed regardless. Then, we use **`$_GET["action"]`** to retrieve the value of the "action" parameter from the URL which tells us what to do next. The first time around, this is not present and so we show the home page of the system, ie the screen from which the user can select a customer to edit.

We build the selection form dynamically from the database, producing a table where each row contains a customer name and a radio button. The name of the radio button is id2edit. The value of each radio button is the database id number of the customer. This is standard practice, and a very common way of using radio buttons (or dropdown boxes) to select a database record. By setting the radio button value to a database id, the code which receives our chosen name knows precisely which customer record we want to edit.

The "action" for the form is set to **`editcust.php?action=show_record`**. This means that the PHP program will be called again when the form is submitted. But this time, **`$_GET["action"]`** will be equal to **`show_record`** and so a different function will be performed.

When we detect that **`$_GET["action"]`** is "show_record", we know that we need to display the customer record form and that we also need to pre-load it with the data for the specified customer.

First, we use **`$_POST`** to retrieve the value of the id2edit radio button in the previous form. We now know which database record the user wants to edit. So, having already connected to the database at the start of the program, we retrieve that user's details (name, address, phone, email, title) into a $row array. Then we create a customer record form, just as we did in the previous addcust.php example for creating a new customer. The only difference is that, in addition to displaying each form field (textbox or textarea), we also generate HTML which fills in the contents of that field for the user of the site to edit.

Some example HTML code for displaying a text box plus some content is:

```
<input type="text" name="firstname" value="Robert" size="25">
```

As you can see, the **value=** component allows you to pre-fill the text box with whatever content you want. So in this case we do just that, but we fill it with some text that was retrieved from the database.

The HTML code to pre-fill a textarea is slightly different. It looks like this:

```
<textarea rows="2" name="address">
123 Farm Road, London W8
</textarea>
```

The **action** for this editing form is set to **editcust.php?action=store_record**. So once again, when the edit has been made and the user presses the Save button, this PHP program will be called one more time. This time, when we discover that $action is "store_record", we retrieve the edited values from the form and update the database accordingly.

One thing to note is the hidden field in the show_record form. In addition to pre-populating the form with the customer's data, we also create a hidden field called id2edit. This then gets passed to the store_record component. If we didn't include this hidden field, then the store_record part of the program would know a customer name, address, phone, email and title, because these were part of the form, but it wouldn't know which database id number they related to and thus wouldn't be able to update the table.

Deleting Data

Just one more database function to cover. We can now read, search, insert and edit data in MySQL tables. All that remains is to be able to delete records. This is very simple, using a **DELETE FROM** query. For example:

```
delete from customers where id=8
```

or

```
delete from customers where surname="Smith" or firstname="James"
```

Mostly you'll use the id-based version, in order to ensure that you're definitely deleting the correct record.

Incidentally, be very careful when using **DELETE FROM**, to ensure that you always specify a **WHERE** clause. For example, if you forget to do so, and you try:

```
delete from customers
```

you'll find that every customer in your table will be deleted And no, there's no undo facility in MySQL. You'll have to revert to the backup which you made via the export facility in phpMyAdmin. What? You mean you aren't in the habit of backing up your MySQL tables regularly? Maybe you need to consider doing so.

There is little to be gained by including a complete example application to delete a customer from our table. It would be almost identical to the editcust.php example. Just use a form and some radio buttons to allow the user to pick the id of the record they want to delete. Then, next time around, retrieve that id and do:

```
$id2delete = $_POST["id2delete"];
$query="delete from customers where id=$id2delete";
$q = mysql_query($query);
echo "Record number " . $id2delete . " has been deleted.";
```

A reminder. During this chapter and the program listings within it, we've used **`$_GET`** and **`$_POST`** to retrieve information. Remember the difference. To retrieve information that was passed as part of the URL, use **`$_GET`**. To retrieve information that was passed as a form field (textbox, textarea or a hidden field), use **`$_POST`**. Just as importantly, remember that any information that you don't explicitly pass either on the URL or in a form field will not be present.

Putting it All Together

So there you have it. You now know about databases, PHP, and how to put the two together. You have all the building blocks necessary to design and create a complete PHP/MySQL application that has the ability to allow users to create, edit, search and delete information. Whether your application uses one table or one hundred, the processes are all the same, once you understand the concepts.

The idea of using **`action=`** on the URL to allow one PHP file to contain an entire application is a good one, and one that I use all the time. For example, you could create the entire Hotel Management System in a single hms.php file. At the start, if $action is blank, you display the main menu (add customer, edit customer, and so on). If the user chooses "add customer", then the action of the menu form might be **`hms.php?action=add_customer`**. This would display the data entry form, and this form might have an action of **`hms.php?action=add_customer_store`**. Next time through the program, if $action is equal to **`add_customer_store`**, you know that the add_customer form was filled in by the visitor and your next step is to retrieve the contents of the form and store it in the database.

Of course, if you prefer, you can split your application into lots of separate files. You might have an hms.php file which does nothing more than display the main HMS menu. Then, if the visitor chooses to add a new customer, you use a form or a simple hyperlink to go to hms_addcust.php instead.

How you choose to arrange things is entirely up to you. Personally, I like to keep an entire application in a single file because it's easier to find what you're looking for during an editing session. But if you like keeping each discrete part of an application separate, then feel free to do so. This is especially useful, of course, if you're working on a project with someone else.

Having covered all the main points of developing a web database application, the remainder of this book looks at improving your general web knowledge and turning a site from a hobby computing project to a viable business idea.

Debugging and Global Variables

Professional PHP programmers don't use a simple text editor like PSPad to develop their code. Instead they install what's known as an IDE, or Integrated Development Environment, on their PC. This includes PHP, a web server, and a sophisticated debugger to help in solving problems.

There are various PHP IDEs on the market, some of which are very reasonably priced and some of which are most definitely not. However, they are best avoided by all but the most serious PHP developers, because they're large, complex, and require you to install things on your PC (specifically PHP and a web server) which pose a serious risk to stability and security unless you know what you're doing. After all, with a web server on your computer, anyone who knows its IP address could connect to your PC and browse your files if you don't configure it correctly.

All of which leaves us with one problem. If all you have is a copy of PSPad and access to a web server over the internet, how do you track down those elusive errors in your programs without the aid of a proper debugger? Hopefully, the remainder of this chapter will help.

There are two main types of error that you'll encounter. The first is a syntax error, where the code you write isn't valid PHP. For example:

```
echo "Sorry, room " . $room[1] . " is occupied"
```

You'll spend ages looking for the missing quote, or dot, only to realise that the problem is actually a missing semi-colon on the end of the line.

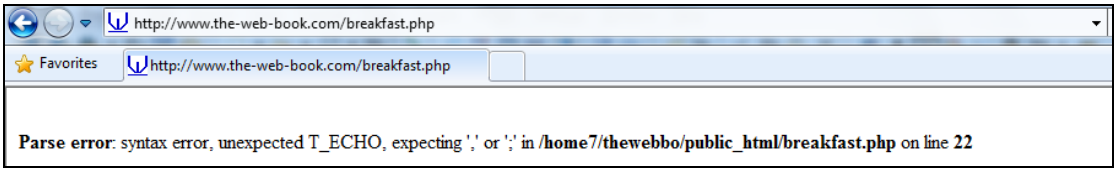
The other type of problem is where your PHP code is valid, but doesn't produce the results that you were expecting. For example, having added a semicolon to the above line, it produces:

```
Sorry, room  is occupied
```

The mistake is clearly yours rather than the server's. It's a coding error. The $room[1] variable is empty and you need to find out why.

Syntax Errors

A syntax error in a PHP file will normally prevent the entire program from running, even if the error is towards the end of the file. In most cases, you'll receive an error message from the server which will help you identify the cause of the problem. For example:



This should be enough to help you track down the problem. However, some server operators regard it as a security risk to display syntax errors, because they could provider hackers with clues as to the inner workings of a program (such as the names of variables or the locations of files). So they configure their server not to display syntax errors at all. Which means that, if there's one in your PHP file, all you'll see is a blank screen. The program won't run, but you won't be told why. In such cases, your only realistic option is to find a different server.

It's important to note that, although server-generated syntax errors tell you the line number in your code that contains the error, this number isn't always correct. It's a great starting point when you're trying to track down the problem, but it's not infallible. This is because PHP tends to assume that the error occurred on the line following the last valid line. Which isn't always the case. This often means that the indicated line number will be "out" by 1 or 2, although in rare cases (especially if you have long blocks of bracketed code and you've omitted a bracket or added an extra one) it can be even more inaccurate.

One useful technique, if you're stuck with a particularly elusive syntax error, is "the chop". Split the PHP file into two separate files, of roughly even lengths. So long as you don't break up any blocks of bracketed code, ie so long as the opening or closing bracket doesn't end up in the other file, you should be safe. The programs won't work the way you expect, but that doesn't matter. What matters is that one of them will generate a syntax error when you surf to it, and one won't. You now know for sure which half of the file contains the error. So split that file into 2 and repeat the process. Eventually, the "half" which contains the syntax error will be small enough that you should be able to recognise where the problem lies.

Coding Errors

When it comes to tracking down coding errors, your best friend a simple **`echo`** statement, possibly followed by **`exit()`**. For example, in the case mentioned above, precede the line with:

```
echo $room[1];
exit();
```

When your program reaches this point, it will display the value of the variable and then stop. Of course, the fact that the variable is blank means that you won't see anything, so you might prefer:

```
echo "XYZ" . $room[1] ;
exit();
```

You can now look on the generated web page for XYZ, and you'll know where to expect to see the room number displayed alongside.

And one final addition, which will also help. Change the code to:

```
echo "XYZ" . $room[1];  #debug
exit();                 #debug
```

This way, once the program is working properly, you can easily search for the lines with the comment in order to remove them.

If all this effort doesn't bring the bug out of the woodwork, then move your debug code further up the file, to just after the place where $room[1] is initially defined. Hopefully, its value at that point should be correct. So somewhere between those two points it's being changed. All that remains is to position your debug code, in turn, just after every line that changes (or which you suspect might change) the value of $room[1]. Chances are, the problem will be down to a simple mis-spelling, eg rooms rather than room, or "1" rather than 1.

If you want to output the values of an entire array rather than a single variable, you don't have to write a for... loop to do it. PHP has such a feature built in. Assuming there's a problem with one or more elements of the $rooms array, and you want to see the whole array rather than just element 1, use var_dump. The following code:

```
<?php
$rooms[1] = "ABC";
$rooms[2] = "DEF";
$rooms[3] = "GHI";

echo "<pre>";
var_dump($rooms);
echo "</pre>";
?>
```

Produces the following output:

```
array(3) {
  [1]=>
  string(3) "ABC"
  [2]=>
  string(3) "DEF"
  [3]=>
  string(3) "GHI"
}
```

This tells you that the array has 3 elements. Element [1] is a 3-letter string comprising ABC. Element 2 is a 3-letter string of DEF, and so on.

If this still doesn't help, you can go one step further and output the contents of every variable that the program knows about. Just do this:

```
echo "<pre>";
var_dump(get_defined_vars());
echo "</pre>";
```

This works because there's a PHP function which creates an array containing all of the variables. So we call this function, then use **var_dump** to display the array. As for the other two lines, the <pre> tag tells the web browser that it's being asked to display preformatted text, and is often used when displaying program listings and the like. Just trust me that the output of var_dump looks much more readable if surrounded by such tags.

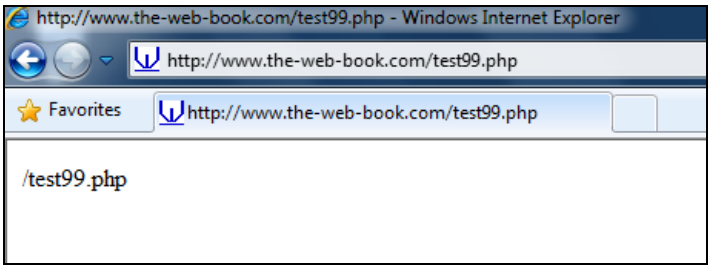
While this also produces the list of the contents of the rooms array, it comes at the end of a very long (many hundreds of lines) listing of lots of other variables as well. These are not variables that your program has created, but ones which the web server and PHP use for their own internal purposes. Some of these are very useful. For example, take a look at this extract from the list that is returned on the the-web-book.com server:

```
["_SERVER"]=>
array(31) {
  ["DOCUMENT_ROOT"]=>
  string(27) "/home7/thewebbo/public_html"
  ["GATEWAY_INTERFACE"]=>
  string(7) "CGI/1.1"
  ["HTTP_ACCEPT"]=>
  string(222) "image/jpeg, application/x-ms-application, image/gif, application/xaml+
  ["HTTP_ACCEPT_ENCODING"]=>
  string(13) "gzip, deflate"
  ["HTTP_ACCEPT_LANGUAGE"]=>
  string(5) "en-GB"
  ["HTTP_CACHE_CONTROL"]=>
  string(14) "max-age=259200"
  ["HTTP_CONNECTION"]=>
  string(10) "keep-alive"
  ["HTTP_HOST"]=>
  string(20) "www.the-web-book.com"
  ["HTTP_USER_AGENT"]=>
  string(151) "Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; Trident/4.0; SLCC2;
  ["HTTP_VIA"]=>
  string(61) "1.1 sulafat.brighton.ac.uk:8080 (squid/2.5.STABLE12-20051129)"
  ["HTTP_X_FORWARDED_FOR"]=>
  string(13) "194.83.112.39"
  ["PATH"]=>
  string(13) "/bin:/usr/bin"
  ["QUERY_STRING"]=>
  string(0) ""
  ["REDIRECT_STATUS"]=>
  string(3) "200"
  ["REMOTE_ADDR"]=>
  string(13) "194.81.199.59"
  ["REMOTE_PORT"]=>
  string(5) "64425"
  ["REQUEST_METHOD"]=>
  string(3) "GET"
  ["REQUEST_URI"]=>
  string(14) "/breakfast.php"
```

This section, as you can see from the top line, is concerned with the contents of the $_SERVER[] array, which contains 31 elements. The element called “REQUEST_URI” (remember that element names don’t have to be numeric) , right down at the bottom of the list here, seems to contain the name of our program file. Let’s just check this, by writing a program thus:

```
<?php
echo $_SERVER["REQUEST_URI"];
?>
```

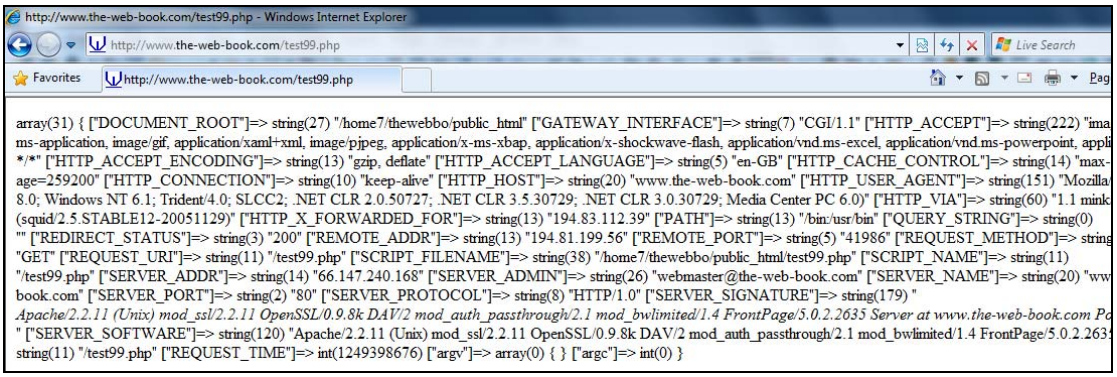
And here’s the result:



This is hugely useful, for reasons we'll come onto in the next chapter.

The $_SERVER Variables

There are lots of other useful items in the $_SERVER[] array. Let's have a look at it with a **`var_dump($_SERVER)`** call, This gives us:



Oops, that's what happens if you forget the <pre> tags! Let's try that again:

```
array(32) {
  ["DOCUMENT_ROOT"]=>
  string(27) "/home7/thewebbo/public_html"
  ["GATEWAY_INTERFACE"]=>
  string(7) "CGI/1.1"
  ["HTTP_ACCEPT"]=>
  string(3) "*/*"
  ["HTTP_ACCEPT_ENCODING"]=>
  string(13) "gzip, deflate"
  ["HTTP_ACCEPT_LANGUAGE"]=>
  string(5) "en-GB"
  ["HTTP_CACHE_CONTROL"]=>
  string(14) "max-age=259200"
  ["HTTP_CONNECTION"]=>
  string(10) "keep-alive"
  ["HTTP_HOST"]=>
  string(20) "www.the-web-book.com"
  ["HTTP_PRAGMA"]=>
  string(8) "no-cache"
  ["HTTP_USER_AGENT"]=>
```

```
  string(151) "Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1;
Trident/4.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR
3.0.30729; Media Center PC 6.0)"
  ["HTTP_VIA"]=>
  string(60) "1.1 mink.brighton.ac.uk:80 (squid/2.5.STABLE12-20051129)"
  ["HTTP_X_FORWARDED_FOR"]=>
  string(13) "191.82.12.39"
  ["PATH"]=>
  string(13) "/bin:/usr/bin"
  ["QUERY_STRING"]=>
  string(0) ""
  ["REDIRECT_STATUS"]=>
  string(3) "200"
  ["REMOTE_ADDR"]=>
  string(13) "191.82.12.56"
  ["REMOTE_PORT"]=>
  string(5) "42476"
  ["REQUEST_METHOD"]=>
  string(3) "GET"
  ["REQUEST_URI"]=>
  string(11) "/test99.php"
  ["SCRIPT_FILENAME"]=>
  string(38) "/home7/thewebbo/public_html/test99.php"
  ["SCRIPT_NAME"]=>
  string(11) "/test99.php"
  ["SERVER_ADDR"]=>
  string(14) "66.147.240.168"
  ["SERVER_ADMIN"]=>
  string(26) "webmaster@the-web-book.com"
  ["SERVER_NAME"]=>
  string(20) "www.the-web-book.com"
  ["SERVER_PORT"]=>
  string(2) "80"
  ["SERVER_PROTOCOL"]=>
  string(8) "HTTP/1.0"
  ["SERVER_SIGNATURE"]=>
  string(179) "
Apache/2.2.11 (Unix) mod_ssl/2.2.11 OpenSSL/0.9.8k DAV/2
mod_auth_passthrough/2.1 mod_bwlimited/1.4 FrontPage/5.0.2.2635 Server at
www.the-web-book.com Port 80

"
  ["SERVER_SOFTWARE"]=>
  string(120) "Apache/2.2.11 (Unix) mod_ssl/2.2.11 OpenSSL/0.9.8k DAV/2
mod_auth_passthrough/2.1 mod_bwlimited/1.4 FrontPage/5.0.2.2635"
  ["PHP_SELF"]=>
  string(11) "/test99.php"
  ["REQUEST_TIME"]=>
  int(1249398795)
  ["argv"]=>
  array(0) {
  }
  ["argc"]=>
  int(0)
}
```

That's a lot better. Although this data may be useful to you, be aware that it's server-specific. Some servers don't include as much information. Some include a lot more. Some don't actually provide anything at all in $_SERVER, citing security reasons. So use what's available on your server, while remembering that it may change if you move to a different one.

One particularly interesting server variable is **$_SERVER["HTTP_REFERER"]**, which lists the URL that the visitor to your site came from. So if, for example, your visitor was at **www.site1.com/example.html** and clicked on a link to your PHP program, **$_SERVER["HTTP_REFERER"]** would be **www.site1.com/example.html**. However, not all web servers pass on this information, so you can't always rely on it being set. In which case, you could do something like:

```
<?php
if (isset($_SERVER["HTTP_REFERER"])
{
echo "You came from " . $_SERVER["HTTP_REFERER"];
}
?>
```

Note the **isset()** function, which tells you whether a variable exists or not (ie, whether or not it is set).

You will see that there are no less than 3 server variables in the above list whose value is test99.php, ie the name of the PHP program file. These are REQUEST_URI, SCRIPT_NAME and PHP_SELF. We'll use one of these in a moment.

Application Structure Revisited

Imagine, if you will, a shopkeeper in a small store which sells a variety of grocery items. This is no self-service emporium. Everything is behind the counter, so you need to ask for an item, wait for the shopkeeper to hand it to you, then place it in your basket.

But this shopkeeper, though extremely fast and polite, has a flaw. He has no memory for names or faces. As far as he's concerned, if you ask for a can of tuna and then a loaf of bread, he won't remember that the person who wants the bread is the same person who wanted the tuna a few seconds ago. He does, though, have a secret weapon. He has a notepad, and he's very good at writing notes to help him remember what's going on around him.

The shop, too, has a flaw. There are a series of high sound-proof barriers at the counter, like the starting stalls on a racecourse. The customer has no idea, when queuing at the counter, whether any of the other stalls are occupied or whether he/she is the only patron in the shop.

To help with preparing the accounts at the end of the month, the shopkeeper records every transaction in his notebook. The writes down the item sold, the date and time, and which stall he handed the item to.

Alice, Bob and Charles are in the shop. Alice asks for some bread, which the shopkeeper duly hands over. Bob then asks for some bread, and gets it. Then Charles asks for some cheese. Then Alice wants bacon. Then Charles wants bacon too. Then Alice wants apples.

Alice then leaves. David arrives, and happens to end up in the stall previously occupied by Alice. He buys potatoes and red wine.

As far as Alice is concerned, she asked for, and got, bread, bacon and apples. She is convinced that she was the only person in the shop. The shopkeeper was serving her, and no one else.

As far as Bob is concerned, he too was the only person in the shop. He asked for bread, and received it. Charles, too, asked for cheese and bacon and was duly served. And David got his potatoes and red wine.

The shopkeeper, of course, has no idea who he served. Each time someone asked for an item, he retrieved it from the shelf and handed it over. Though he did keep a record in his notebook, so he knows what was handed over, in what order, and which of the “stalls” he handed it to. One particular stall, for example, got bread, bacon, apples, potatoes and red wine. As it happens, those 5 items went to 2 different people but the log doesn't record this.

Web Servers and the Real World

OK, you can open your eyes now. We're back in the real world.

The shop analogy describes exactly how a web server works. Requests for pages come in from all over the world and they are handled, one at a time, by the web server. Maybe Alice wants to see the home page, so we send it down the line to her. Then Bob wants to see the Contact Us page, so we send it. Then Alice clicks on the "Our New Widget" link, from the home page, so we send it to her. There is no linkage between Alice's two page requests. First, she got the home page. Then, some time later, we sent her the page about the new widget. In the mean time, 1, 2, 3 or a million other people might have requested pages. All we have to tie everything together is the server's log file, like the shopkeeper's notepad, which tells us who asked for what, and in what order.

Alice and David both used the same stall, but the log file doesn't provide any way to record this. All that the shopkeeper knows is which stall a particular item was handed to, and not the name of the person in that stall at the time. In the same way, two different users of a web server may use the same IP address, albeit not at the same time.

This lack of an ability to tie one user's chain of actions together is called statelessness. In other words, a web page (or a request from a user's browser for the web server to send a page) is stateless. It doesn't contain any information about the state of the connection between the user and the server.

When the server simply contains static HTML pages, this isn't a problem. People send in requests for pages, and we send them out as fast as possible. But with web-based, server-side applications, things are different. Here's a trivial example. The receptionist at the hotel views the list of guests, and clicks to see the full details about guest number 64. By clicking on the "View Full Info" button, he/she launches another PHP program. This program is quite capable of showing all the information about a guest, but doesn't know which one to show. Sure, 64 was chosen in the previous program, but that wasn't passed on.

Here's another example. Take the case of our hotel management system, and imagine a page that lets members of the public check availability. Brian surfs to the site, and enters "January 9th" in a drop-down box on the availability checker form. He clicks submit. A PHP program receives his date, checks the database, discovers there's one spare room, and sends back a page saying "yes, we have a room on that date". Three seconds later, Janet fills in the form. She, too, is looking for somewhere to stay on the night of January the 9th. Again, the PHP program checks the database and, what do you know, there's one vacancy. Janet and Brian both begin to fill in the booking form...

Welcome to the world of multi-user database applications.

How do you solve the problem? One way is for the system to reserve the room for Brian for, say, 15 minutes. Until then, anyone else who tries to book that room will be told it's not available. After 15 minutes, the lock is released and it goes back into the pool of available rooms, assuming Brian hasn't booked it.

How do you do this in practice? One way is to have a column in the rooms table called locked_until. To lock a room, just look up the current timestamp, add 900 to it (900 seconds is 15 minutes), and write that into the locked_until field of the room that Brian's interested in. When you come to search for availability, if there's anything in a room's locked_until field, assume the room is not available. And every 5 minutes or so, run a database query that removes every locked_until field which is less than the current timestamp.

This is just one of the concepts you need to appreciate when writing web-based applications using server-side languages such as PHP. Something else that you need to consider is how to structure your app. For example, do you keep all of the code in a single file, or do you use one single, larger file to contain your entire application? This is something we have already discussed.

Saving State

Earlier in this chapter, we talked about the way that web servers are stateless. When a server receives a request from a visitor to send out a page, the server has no idea who the user is, or what state their session is in. Have they just arrived at the site? Or have they been browsing for a while? Are they logged in, or are they an untrusted user?

One way to solve this problem is to use a PHP feature called sessions. This is an advanced topic, beyond the scope of this introductory book. If you want to know more, type "php sessions" into your favourite search engine.

How to Back Up your Web Site

Like any other collection of computer-based files, you need to back up your web site. Sure, most web hosting companies have their own backups, but that's only to protect themselves against major disasters like the loss of an entire server and all of the customers' sites that it contained. If you accidentally delete a key file from your site, or someone manages to hack one of your pages and replace its contents with porn, it's unlikely that your hosting company will be willing or able to help. Many companies don't do backups properly, and web hosting companies are no exception. So it's much better to back up your own site and be safe rather than sorry. Also, if you ever want to move your site to a different server, having a backup to upload to the new server makes the process relatively easy.

At the very least, your web site will consist of a collection of assorted files (mostly HTML and some images, probably) in a top-level folder called something like public_html. There may also be some additional folders within that top-level one too. All of these folders (ie, the entire contents of public_html and everything within it) will need to be backed up.

Unless you know for sure that you need to, it's unlikely that you'll need to back up anything that's outside of public_html. Generally speaking, these files are there to help the server rather than you, and contain non-essential things such as logs. So for backup purposes they can be ignored. One example of where data outside of public_html does need to be backed up, though, is if your host insists on .htaccess files being placed in a specific folder outside of the public_html one. This is indeed the case with hostmonster.com, as discussed on page 114.

Backing up a web site is simply a case of downloading the files to a local PC for safekeeping. To download the files you can use any FTP program, including FileZilla. Just connect to your site, navigate to a suitable empty folder on your local computer. Then find the remote folder on the server (eg, public_html), right-click it, and choose Download. The files will begin downloading. Once it's finished, either keep the files on your PC, or perhaps archive them to a CD, DVD, USB pen drive or external drive for safekeeping. Now, if anything happens to the contents of your web server, you have a safe copy.

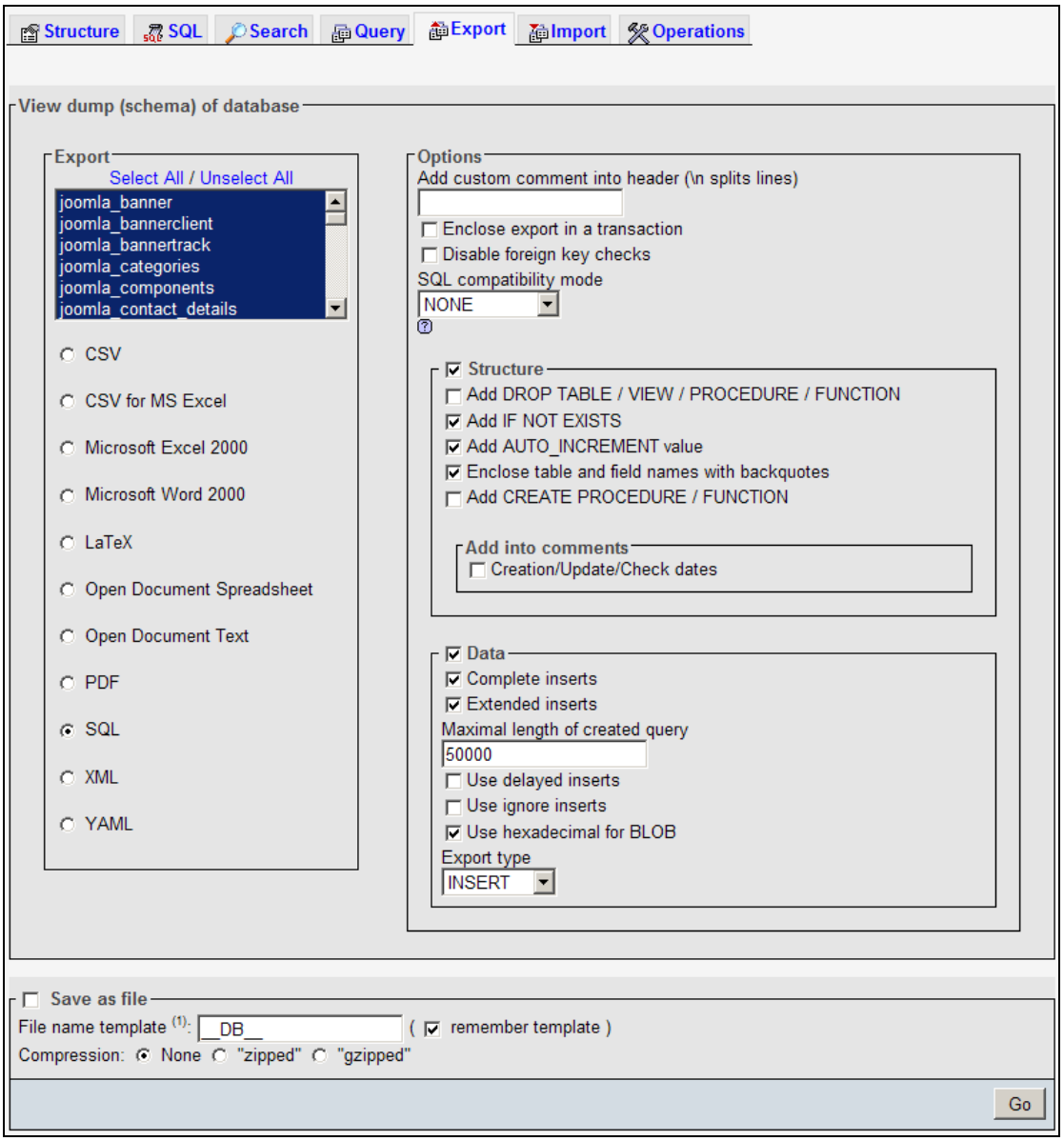
You don't have to back up the entire site at once, of course. If you just want to download a single file, the same rule applies. In FileZilla, right-click the filename on the remote server and choose the Download option.

Don't Forget the Data

Having backed up the HTML and image files from your site, you also need to back up your MySQL databases. Obviously this won't be relevant if you don't use a database. But if you

do, failing to back up the database as well as the contents of public_html means that much of the content of your site has not been backed up. Thankfully, backing up is easy with phpMyAdmin, which is a tool we first encountered on page 190 for creating database tables and inserting information into them.

To back up a database, start phpMyAdmin by surfing to its URL, or via the control panel on your web host. Select your database from the list on the left hand side, and click on Export from the menu items along the top. You'll then see something like this:



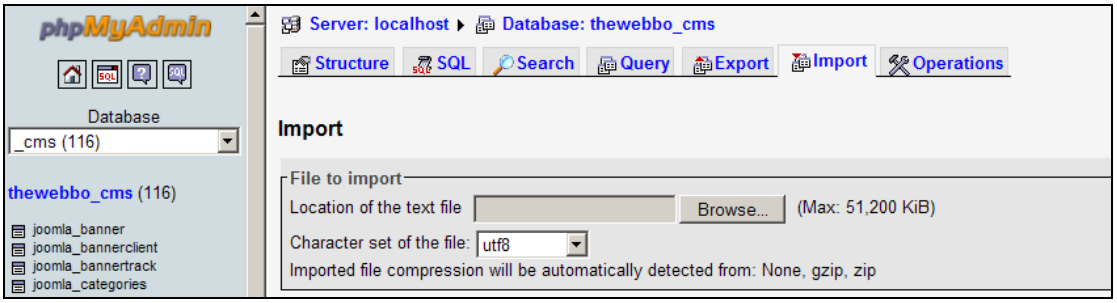
In the "Export" box, choose which tables in the database you want to back up. Assuming you'll be wanting to restore the backup onto a web server and into a MySQL database in the future, leave the export format set to SQL. But you may wish to explore the other options, such as CSV or Excel, if you want to export the MySQL database tables into a text or

spreadsheet format. This can be useful if, for example, you have been running an online survey which saves its data into a database and you then want to extract the information into a spreadsheet to analyze offline.

All that remains is to tick the "save as file" box, type a name into the filename template area, and phpMyAdmin will start the backup.

The backup itself will consist of a text file, downloaded to your PC, which contains a program script. By uploading this script to phpMyAdmin in the future, it will restore the contents of the backup into the database tables.

Note that most hosting companies place a limit on the size of a backup script that you can upload in order to restore. Check yours by clicking on the Import link and looking for any hint. Here's the screen from hostmonster.com's version of phpMyAdmin which, as you can see, has a limit of 51,200 KiB, which is around 50 megabytes.



Therefore, if you find that backing up all of the tables of a database results in a script file larger than 50 MB, re-do the backup in multiple parts, a few tables at a time, ensuring that each script is less than 50 MB.

Restoring Lost Information

Hopefully you'll never lose information from your site and so you won't ever need to restore from a backup. If you do, though, it's not particularly difficult.

In the case of static files, such as HTML files and images, just use FileZilla to re-upload the files to your site.

To restore a MySQL database that was backed up with phpMyAdmin, start phpMyAdmin, go to the database you want to restore, then click the Import link on the top menu line. Choose the script file to upload (this needs to be one that was created by phpMyAdmin's export feature) and your database is restored.

If you've never tried backing up and restoring a MySQL database table, now might be a good time to make sure that you know how to do it. Use phpMyAdmin to create a test table, insert a couple of rows, then back up the table and delete it. Now try restoring the table from the backup and check that it reappears.

Promoting and Profiting

As you begin to create Web-based applications, your attention will probably turn at some point to wondering whether you can make money from your sites. After all, Facebook and Twitter are just web-based applications, as indeed is Google. If your site could make just a thousandth of the revenue that those systems make, life would be wonderful.

If you're serious about this side of things, I strongly recommend that you also pick up a business textbook as well as this technical one. Making money from web sites is as much about knowing about marketing as being able to write programs. But for now, I'll assume you simply want an overview of the ways that you can promote your site and make money from it.

This chapter covers two main topics, which are closely linked. First, spending money to advertise your site online. Second, making money from your site, by hosting other people's advertising.

Promoting Your Site

One of the most costly mistakes that web site developers make is to assume that, if you build a site, the visitors will come in their droves. Frankly, they won't. However good your site is, your weekly total of visitors will be close to zero unless you can promote it. Attracting a healthy number of visitors is difficult. Keeping them coming back to your site, week after week, is even harder. After all, think of how many hundreds or thousands of web sites you've visited in the past. How many of them do you visit at least once a week?

Attracting visitors to your site (or, as the professionals call it, driving traffic to your site) can be done in many ways. The cheapest option is to simply plug away, relentlessly mentioning it in as many online forums and message boards as you can find. But do it too often, or mention it in places that aren't relevant, and you'll rapidly gain a reputation as a spammer.

The other end of the scale, if you have a large marketing budget, is to advertise the site on radio, TV, and in newspapers and magazines. However, regardless of the cost, advertising a web site is relatively ineffective for one simple reason: most people aren't near their computer at the time they see or hear your ad. And by the time they get to the PC, they'll have forgotten to visit your site or they'll have forgotten your URL.

Therefore, by far the best way to attract visitors to a web site is through online presence, where the customer can instantly click through to your site. It pays to spend some time reading the various documentation that Google makes available to webmasters, to ensure that your site is as far up the search results pages as you can possibly make it. Also, spend

some time researching the sort of news sites that your ideal visitors might read, and, where relevant, send a press release to the site.

Don't underestimate just how much of a difference it makes to have your web site mentioned somewhere online, where people can click straight through, rather than in print. To quote one fairly unscientific example, when I wrote an article for a UK-based national newspaper, the mention of my website in print generated a few hundred hits. When I sent a press release about a new product to about 20 different online news sites, it was published on about 5 of them and generated a quarter of a million hits.

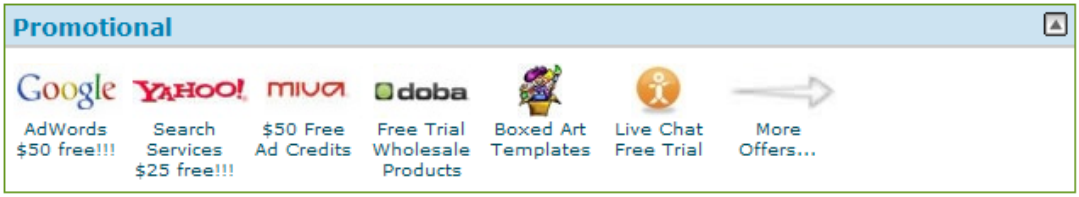
If you want to advertise your site online, and you have some money to do so, by far the most popular service is Adwords, from Google. Sign up for the service via Google's web site, give them your credit card details, then pay some money into your Adwords account. Then choose the search terms that you want to buy. For example, if your site is promoting the Star Inn Hotel in London, you'll probably want to choose terms such as "star inn hotel", "London hotel", "cheap London breaks", and so on. The skill is to think of what the typical Googler might be searching for.

If you're happy with the price that Google offers you, click a button and your advertising campaign is up and running. When someone searches for, say, a cheap London break, your ad will be shown on the right hand side of the search results page in the Sponsored Links section. It won't be shown every time, necessarily. It depends on how many other companies have bought that search term, and whether they've elected to pay more than you.

Adwords is a "pay per click", or PPC, service. Google will automatically display your ad on search results, but you only pay when someone actually clicks on the link to your sponsored site. At which point your Adword account is debited by a small amount. You'll know the amount, because you'll have agreed to it when you set up the campaign. When your account balance is empty, Google stops displaying your ads and emails you a reminder to add more credit.

The Adwords management site is very sophisticated. You can log in and view details of how often your ads are being clicked on, and you can set daily limits so that, for example, your entire balance won't be used up on day one of a campaign.

If you want to explore the use of Adwords, Google gives away millions of starter vouchers every month, entitling you to around £30 or $50 of free Adwords credit. Just look for the vouchers in your favourite computer magazine. Many Web hosts also give away similar promotions, so check on your web hosting control panel to see if yours does. In the case of Hostmonster, here's the relevant section of the control panel:



As you can see, there's $50 of free Adwords credit available, as well as some credit for spending with Yahoo and Miva. Considering that all three are free, at least to try out, it makes sense to spend the free credit and see which, if any, is worth sticking with as a way to help promote your site. However, note that the free Adwords credit offered by Hostmonster is only available to US-based customers.

Making Money

If you want to make money from your web site, there are really only two ways to do it. One is to charge visitors a subscription. The other is to place adverts on your pages, and charge the advertisers for the privilege.

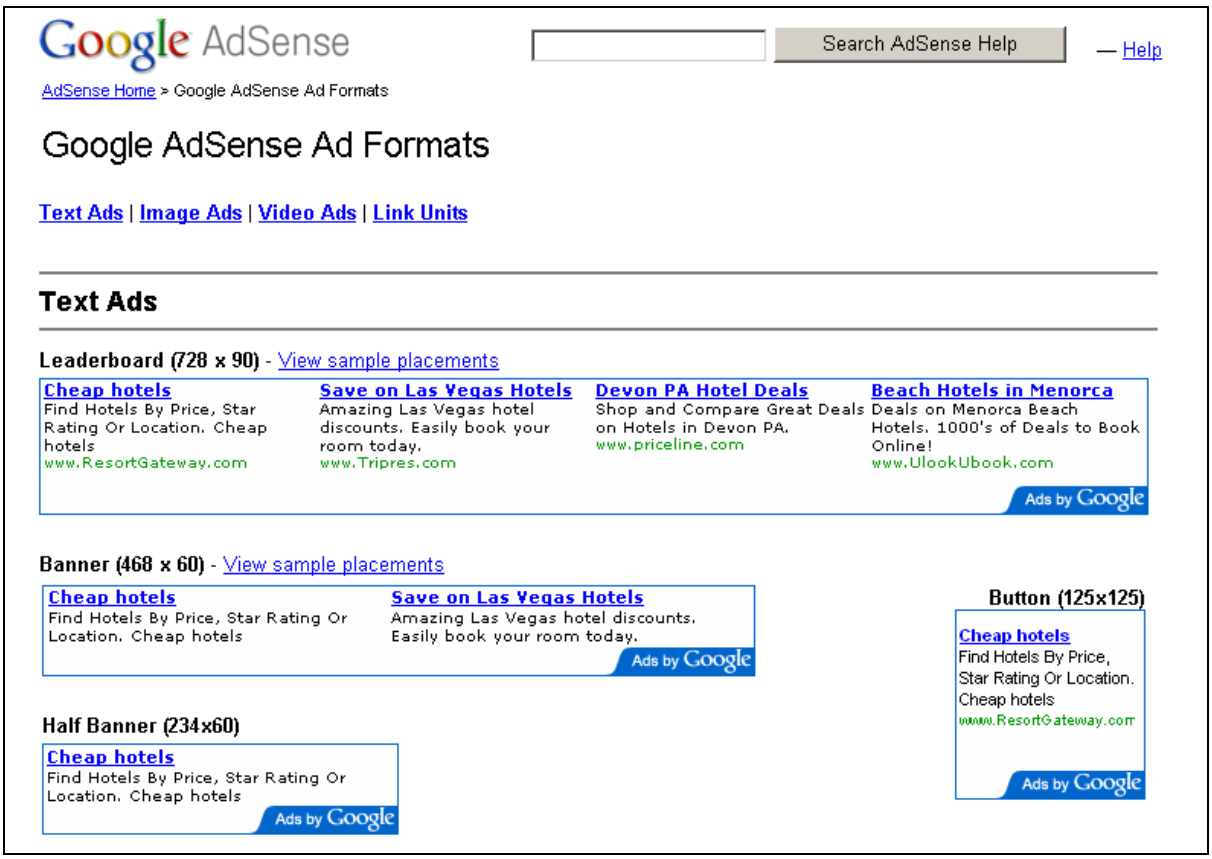
In general, it's difficult to make money from a subscription site. Most people don't like paying for access to a site, unless it's for information or other content that they can't get elsewhere for free. And even then, they'd rather search the web for a pirated username and password. Marketing subscriptions is expensive, and persuading existing customers to keep renewing every month/year is also difficult. Therefore, unless you're an experienced web marketer, charging a subscription fee is not recommended.

The other option, ie carrying ads on your site, makes more sense. Especially if you can find a company that will do all the hard work for you, to save you having to sell the advertising space and handle the payments yourself. Thankfully, there are indeed such companies, and getting everything up and running is incredibly easy. As you might expect, the leading player in this market is, once again, Google.

Google Adsense is a scheme whereby your site carries adverts that have been placed by Google's Adwords advertisers. So if, for example, an advertiser has sponsored the search term "cheap hotel break", and your web site contains information about holidays, then, in addition to the advertiser's ads being shown on Google's search results page, it will also be shown on your site. And each time someone who visits your site clicks on one of your adverts, you make money.

Adsense is well worth signing up for, even if only as a trial to see if it's worth your while. But like any other internet-based business, you need to invest some time if you want to make the most of it. This means continually tweaking the layout of your site, and the content, and the position of the ads, in order to find out what works best for you.

To sign up for Adsense, and hopefully start making money from your web site, go to **`www.google.com/adsense`** (or the local version of Google in your country, if there is one). You'll need to have a Google account to proceed, so sign up if you don't already have one. You can then choose a design for the ads, such as from the examples shown here:



You can also choose the colour scheme, font style, and whether the ads have rounded corners or not. You can even specify the names of your main business competitors to ensure that ads for those companies won't be displayed on your pages.

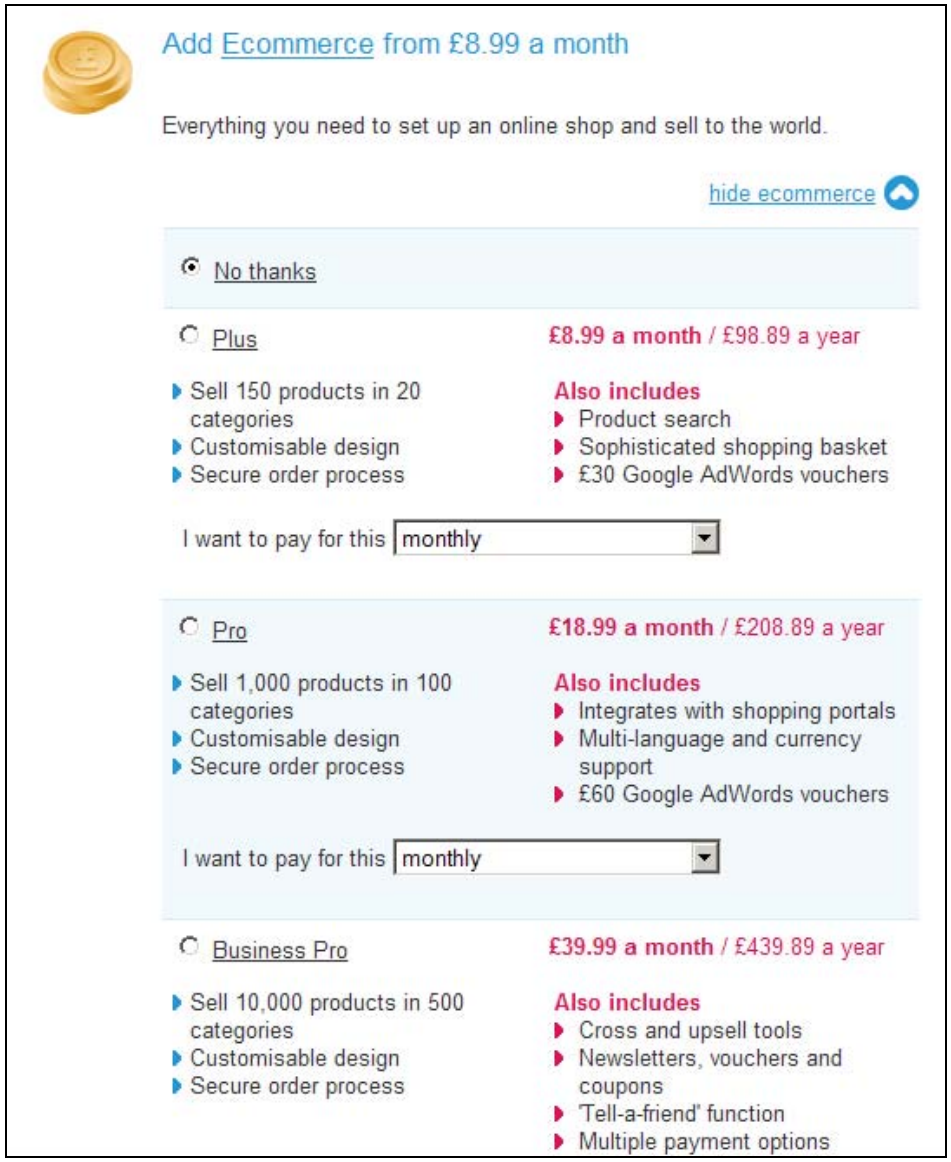
Once you've specified the ad format, Google will display a few lines of HTML code. All you need to do is copy it, and paste it into your web pages at the point where you want the ads to appear. That's all there is to it. The HTML code contains a pointer to your unique Google Adsense account, so your account will automatically be credited whenever someone clicks on an ad. All that remains is to log into Adsense every few days and check how much you've earned, and then tell Google how you'd like to receive your payment.

Incidentally, don't be tempted to click on the ads yourself, or persuade friends to do so, in order to make a quick buck. It won't work. Google will quickly notice what you're doing, and close your account.

Accepting Online Payments

Have you ever wondered how sites like Amazon and eBay allow you to buy things online with your credit card automatically? Would you like to have your web site offer the same facility, so that people can pay for your products or services online and the money automatically ends up in your account? If so, read on.

If you're looking to set up an online shop, to sell either physical goods or downloadable items, there are various ways to do it. The simplest is to find out whether your web hosting company and/or your ISP already offer such facilities. Here's the relevant part of the control panel from one web hosting company, for example. For as little as £8.99 a month, or around $15, you can have an online shop with 150 items. It just takes a few clicks, and is by far the easiest way to proceed, short of simply selling on eBay instead.



If you want to set up your own online shop, however, and do things yourself, you'll need to learn about two new bits of technology.

First, you'll probably want to find a program that can automatically manage your catalogue of items, pictures and prices, and turn it into a neat collection of browsable web pages to publish on your site. You don't have to do it this way, of course. You could create the pages yourself, but adding new products or categories can be complicated.

If you don't have a budget, check out a free package called osCommerce at **www.oscommerce.com**. If you would rather pay for a commercial product, one major player is Actinic Catalog (see **www.actinic.com**).

The second technology is how to actually handle the payments. The theory is simple. Sign up for an account with a payment handler such as Paypal and/or Google Checkout. Then add some PHP programming to your site so that visitors can pay for your goods electronically.

Shopping cart software such as osCommerce and Actinic Catalog have payment integration built in. So long as you have an account with one of the supported payment companies, and your product database contains a unique id number and a price for each item, the software will do the rest. But if you'd rather write your own integration code, it's not particularly difficult. Every payment handler publishes an integration manual, which contains complete instructions and some sample code.

In the case of Paypal, the process is relatively simple. You create (or buy some software that does it for you) an HTML form that allows the visitor to specify which item they want to buy from you. The form fields specify the item id and the price. There are also some additional hidden fields that contain your account number and an order id. You configure the form's "action" property so that its contents get sent to Paypal, rather than your own site, when the visitor clicks the "Submit" (or probably the "Buy Now") button. Paypal then takes over, asks the user for the credit card details etc, and credits your account. It then sends back a confirmation message to your site by calling your URL, eg:

```
www.yoursite.com/paypal?confirm=yes&order_id=12345
```

Your PHP code then retrieves the order_id from the URL, and therefore knows which order has been successfully processed (or not). You can then update your own records accordingly, such as by adding a record to your customer database to indicate that this customer has paid for their goods.

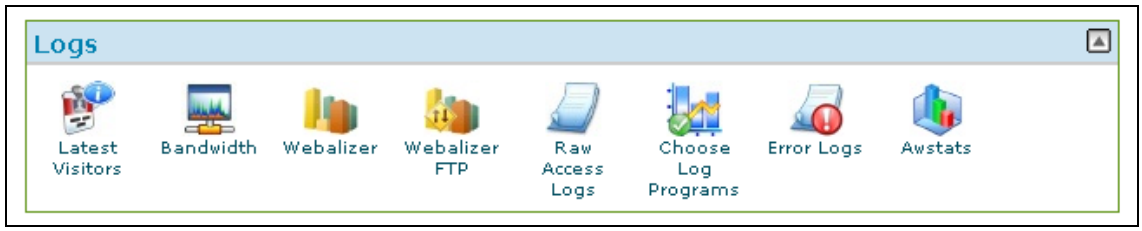
Needless to say, the process is slightly more involved than this, partly in order to ensure that it's secure, but the procedure is very much as described.

If you want to make things even easier, you don't have to go down the route of full payment integration. For example, on the www.the-web-book.com site is a button that allows you to make a donation for the book by Paypal. This took just a few minutes to set up, including me signing up for a Paypal merchant account. The lack of proper integration means that I don't have an online MySQL database containing precise details of who has paid, but I can get basic information by logging into my Paypal account so it's not a problem. If you want to see online, check out Paypal (and Google Checkout, which is similar) first. It may save you a lot of time.

Managing your Marketing

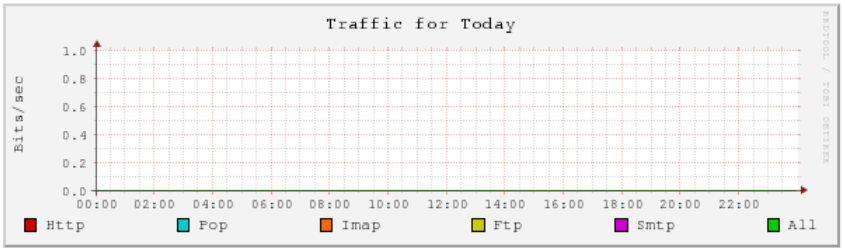
Whatever you're selling online, and however you're trying to make money from your site, you need to be able to manage your marketing effort in order to know whether it's actually working. It's not unknown for large organisations to spend tens of thousands of pounds/dollars per month advertising their business on Google Adwords. With that sort of money at risk, or even if you're just spending a tiny proportion of that, you need to know whether your money is being well spent. Equally, if you're trying your hand with a DIY approach, and sending out press releases, you need to know whether or not you're wasting your time.

At the most basic level, you can find this out by looking at the raw hit counts on your site. Most web hosting companies offer some form of hit counter, so you can always see at a glance how busy your site has been. In the case of Hostmonster, here's the relevant section of the control panel:

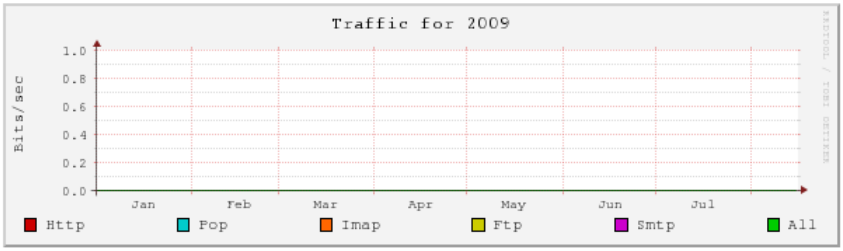


The Bandwidth option shows how much information has left your site. For example:

Today



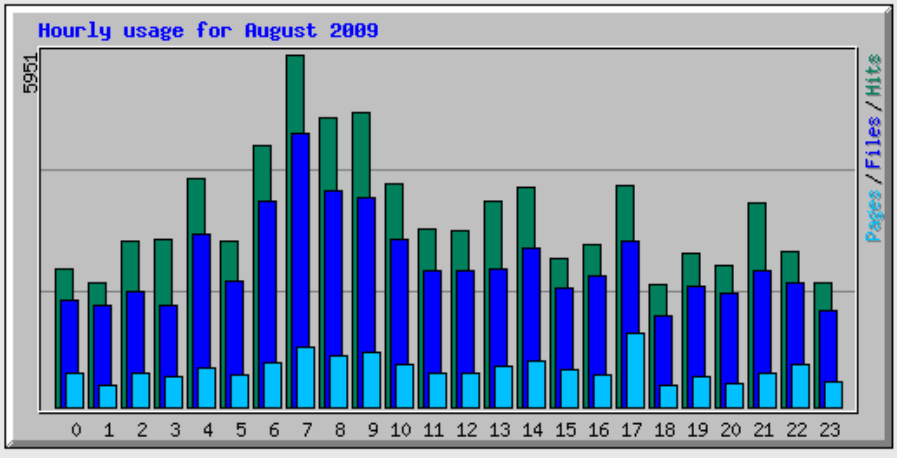
This Year



Webalizer and Awstats show graphics which indicate how busy your site has been. You can view the number of page hits for any given hour, day, month or year. Be aware that not all hosts offer the same programs, and in some cases you need to enable the logging otherwise it won't be available. This is the case with Hostmonster – if you don't click on the Choose Log Programs option and enable Awstats and Webalizer, you won't get any data to analyze.

Here are some of the graphs and reports that Webalizer can show. These are taken from the site for this book which, as you can see, was pretty popular during August 2009. The reason for the traffic peaks around 6am is that, at the time, the book was mostly being promoted on an Australian web site and yet the server is based in the USA.

Monthly Statistics for August 2009		
Total Hits		78036
Total Files		58481
Total Pages		15486
Total Visits		6100
Total KBytes		46855908
Total Unique Sites		5652
Total Unique URLs		669
Total Unique Referrers		496
Total Unique User Agents		1571
	Avg	Max
Hits per Hour	191	3669
Hits per Day	4590	33922
Files per Day	3440	26531
Pages per Day	910	5784
Visits per Day	358	2914
KBytes per Day	2756230	24634400
Hits by Response Code		
Code 200 - OK		58481
Code 206 - Partial Content		127
Code 301 - Moved Permanently		25
Code 302 - Found		2
Code 304 - Not Modified		8134
Code 400 - Bad Request		1
Code 404 - Not Found		11266



Hourly Statistics for August 2009												
Hour	Hits			Files			Pages			KBytes		
	Avg	Total		Avg	Total		Avg	Total		Avg	Total	
0	136	2327	2.98%	105	1794	3.07%	32	558	3.60%	80308	1365233	2.91%
1	122	2082	2.67%	101	1727	2.95%	21	364	2.35%	88930	1511814	3.23%
2	165	2809	3.60%	114	1946	3.33%	32	559	3.61%	99150	1685544	3.60%
3	165	2818	3.61%	101	1726	2.95%	30	516	3.33%	80417	1367090	2.92%
4	226	3858	4.94%	172	2930	5.01%	38	660	4.26%	80728	1372383	2.93%
5	164	2793	3.58%	124	2122	3.63%	32	546	3.53%	101315	1722348	3.68%
6	259	4408	5.65%	204	3476	5.94%	43	740	4.78%	157594	2679102	5.72%
7	350	5951	7.63%	270	4603	7.87%	60	1023	6.61%	245059	4166011	8.89%
8	286	4874	6.25%	214	3646	6.23%	51	868	5.61%	171834	2921179	6.23%
9	291	4963	6.36%	207	3532	6.04%	54	926	5.98%	174847	2972401	6.34%
10	221	3759	4.82%	166	2836	4.85%	41	704	4.55%	158957	2702277	5.77%
11	176	3005	3.85%	135	2305	3.94%	34	582	3.76%	116895	1987212	4.24%
12	175	2987	3.83%	134	2294	3.92%	32	558	3.60%	114060	1939023	4.14%
13	204	3468	4.44%	137	2344	4.01%	40	684	4.42%	105917	1800582	3.84%
14	218	3710	4.75%	157	2685	4.59%	45	765	4.94%	114764	1950982	4.16%
15	146	2497	3.20%	117	2001	3.42%	36	616	3.98%	94300	1603101	3.42%
16	161	2740	3.51%	129	2199	3.76%	32	555	3.58%	88846	1510387	3.22%
17	220	3746	4.80%	164	2796	4.78%	73	1241	8.01%	93429	1588298	3.39%
18	122	2076	2.66%	89	1528	2.61%	21	364	2.35%	83779	1424239	3.04%
19	152	2597	3.33%	120	2047	3.50%	30	525	3.39%	92725	1576333	3.36%
20	141	2402	3.08%	112	1913	3.27%	23	395	2.55%	104879	1782935	3.81%
21	202	3448	4.42%	135	2299	3.93%	33	567	3.66%	130358	2216083	4.73%
22	154	2627	3.37%	123	2100	3.59%	43	732	4.73%	91429	1554292	3.32%
23	123	2091	2.68%	96	1632	2.79%	25	438	2.83%	85709	1457058	3.11%

With the graphs from systems like Webalizer at your fingertips, it's easy to see whether your advertising is working. For example, try cancelling your Adwords campaign for a few days, then start it again and see if there's a noticeable increase in traffic. Also, log into your Adwords account on Google and look at your click rates, ie the number (or percentage) of ads displayed on your site which people actually click on. Then perhaps try tweaking or rearranging the site layout slightly and see if the numbers improve.

If you want even more stats and data from your site, consider signing up to an analytics service. Google Analytics lets you tracks visitors as they progress through your site, in order to build up a picture of how they use it. For example, if you have a site consisting of a home page, a product catalogue and an order form, Google Analytics will tell you what percentage of the visitors who browse your catalogue subsequently abandon their shopping cart rather than paying for its contents.

Google Analytics is free to use. Just sign up for an account, and they'll give you a couple of lines of HTML code that you need to include in the header section of every page on your site.

Once this is done, Google silently tracks all your visitors, then you just log into the control panel to view the data.

Search Engine Optimisation

Having built a web site, whether from static HTML pages or as a PHP/MySQL application, your job is not finished. You now need to attract visitors to it.

Most people locate web sites of interest by using a search engine such as Google. Therefore, by far the easiest and cheapest way to attract visitors is to ensure that your site, rather than anyone else's, is top of Google's list.

Sadly, if you're hoping that this chapter will tell you how to achieve this, you're in for a major disappointment. For obvious reasons, search engine companies don't divulge the precise details of the formulas that they use when deciding whether one site should be higher in the list than another. However, they do publish a selection of tips which can help. These, combined with anecdotal evidence from people who run sites that do achieve a good Google ranking, are clearly worth following.

The art (actually it's more like a science) of getting good search results is known as search engine optimization, or SEO. If you want to know more about it, search online, or in your favourite bookshop or library.

If, during your quest, you encounter adverts for companies or software products which guarantee your site a top-10 placing in return for a fee, avoid. As previously mentioned, search engines keep their methods highly secret, so anyone who claims to know how to defeat them is, at best, mistaken. Don't waste your money. Instead, follow the advice below.

SEO Tips

You can follow these tips to help ensure that your site gets a good search engine placement. Results are not guaranteed, but what follows is based on official advice from Google as well as anecdotal evidence from friends of mine who run high-ranking sites.

- Search engines work by automatically scanning, or crawling, the web and adding every word of every page to their enormous indexes. They normally manage to find new or changed sites automatically on their regular (every week or 2) crawl. However, you can kick-start the process for a new site by manually telling the search engine to add it to its list of sites to crawl. To do this, look for the "submit your site" link which every search engine site has. You only need to enter the URL of the front page. The search engine will then automatically follow all of the links within it to locate other pages.

- Search engines typically have between 10 and 20 billion web pages in their index. A database of this size takes a long time to build, so don't be surprised if an amendment to your site doesn't show up in Google for a few weeks.

- Ensure that every page on your site has a page title. See page 53 for how to do this.

- Make sure every page has a "description" and "keywords" meta-tag, as outlined on page 55.

- Every page on the site should be reachable by at least one hyperlink, so that the search engine can follow the links in order to find all your pages.

- Be aware that search engines are very poor when it comes to indexing information that is held in MySQL databases and which doesn't exist as a static HTML page. The reason for this should be obvious. Google can't access your MySQL database directly, but only through your website. So if you have a page which says "Please enter a name to search for", the only way for Google to extract every name from the database would be to electronically "type" every possible name in the world into that box. So, in the case of database-driven sites, it's especially important to also have a handful of "real" pages (ie, without a **?** in the URL) that contain lots of relevant information about the site.

- Google Sitemaps, often known as XML Sitemaps, are a way of helping Google and other search engines find their way around a web site in order to index it. If you have a site where the majority of content is generated dynamically from a database, rather than existing as static HTML files, it is well worth investigating these. Don't get confused with standard site maps, as mentioned below, which consist of easy-to-read collections of links, which are designed to help human visitors navigate your site. An XML or Google Sitemap isn't as easy to read, at least not by humans.

- Offer a site map to your visitors, with links that point to the important parts of your site. Even if visitors don't use it, search engines will. If the site map is larger than 100 or so links, you may want to break the site map into separate pages.

- Don't try to cheat. Some people include keywords that don't accurately reflect the content of their pages. Or they use tricks such as white text on a white background to hide extra information within a page that Google will see but which is hidden from human visitors. When Google's automatic systems detect this, your site will be removed from all of Google's indexes and it's very difficult to get reinstated.

- Think about the words users would type to find your pages, and make sure that your site actually includes those words within it.

- Use text instead of images to display important names, content, or links. The Google crawler doesn't recognize text contained in images. If you must use images (such as a logo, or a picture of a chart or graph) for textual content, also use the "alt" attribute to include a few words of descriptive text.

- Make sure there are no broken HTML links in your pages.

- Ensure that all pages validate correctly. There's more about this on page 56.

- Ensure that every page specifies a DOCTYPE. There's more about this on page 60.

- If you use a "keywords" meta-tag on a page, ensure that the page also mentions those keywords. Otherwise Google might assume that you're trying to cheat, by listing keywords that are actually nothing to do with the content of the page.

- Ensure that HTML page filenames and image filenames are, where possible, relevant to the content of the site rather than just a meaningless name. For example, in a site about a hotel, a page called vacant_rooms.html which contains an image called sample_room.jpg is better than a file called vr01.html and an image called samp.jpg.

- The more often you update your pages, the higher they'll be ranked by search engines. Google doesn't favour out-of-date sites.

- Use proper hierarchical HTML tags in your pages, regardless of whether you generate the content from a database or you use static HTML files. This is known as using semantic markup, and you'll find out more by searching for the phrase in Google. Essentially, use the built-in h1, h2, h3 etc heading tags, correctly structured. So start with an h1 tag at the top of the page, then one or more h2 tags, and perhaps also some h3 headings within the h2 ones. Don't put, say, an h3 heading directly within h1 section. This all helps the search engine understand the structure of the page.

- Google regards your page as especially important if there are lots of links to it from other sites. So encourage friends and business partners to link from their site to yours, perhaps in exchange for a link from your site to theirs.

Keeping the Crawlers Away

Most of the time, you want to encourage Google and other search engines to crawl your site and add your content to their index. But sometimes there will be areas of your site that you

would rather didn't get indexed. For example, the folder or directory that holds unfinished pages that aren't for public consumption.

There's a way of telling search engines which folders to ignore, and it's done through the use of a robots.txt file. This is a special text file that you need to place in the top-level folder of the publicly accessible part of your web site. In the case of hostmonster, that means the public_html folder. A typical robots.txt file, which you can create with any text editor such as Wordpad or PSPad (but not a word processor like Word), might look like this:

```
User-agent: *
Disallow /joomla/images
Disallow /hms
```

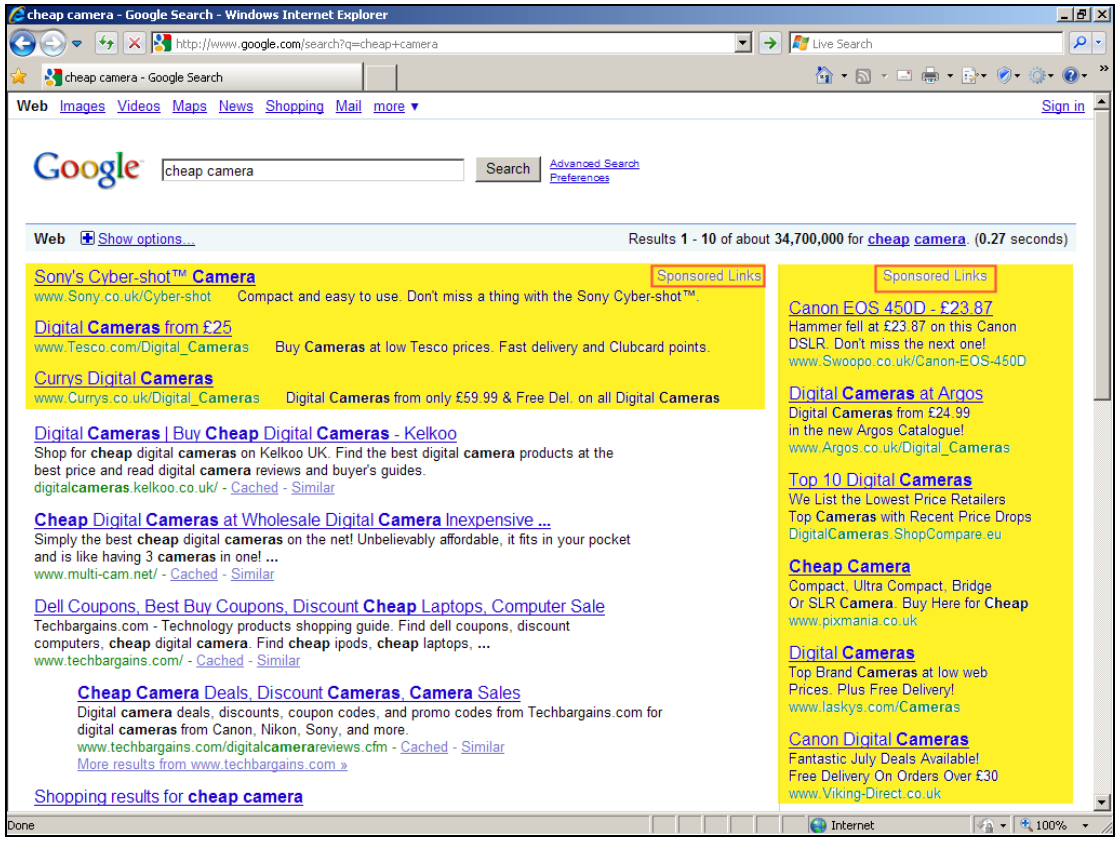
The first line is where you could, if you wanted, specify which search engines you wish to ban from your site. Chances are, you'll want to block them all, which is what the line in the above example does. Then, just list all the folders you want to block, using Disallow commands. In this case, I'm blocking Joomla's images folder and our HMS system.

Note that there's no compulsion for search engines to obey the contents of a robots.txt file, but almost all of them do.

Regardless of whether you use a robots.txt file or not, don't use a web server as a way of storing or exchanging private information. That's not what they're designed for. Even though you don't include a link to your private file area from your public web pages, that doesn't mean that Google won't already have found the files and indexed them for the rest of the world to find. This happens a lot. Try a Google search for "private and confidential" or "commercial in confidence" for some worrying examples.

If at First you Don’t Succeed, Pay

Each page of Google search results contains two sorts of information. In the image below, where I've used Google to search for a "cheap camera", the results at the very top, as well as those down the right hand side, are discretely labelled as Sponsored Links and shaded in a very pale yellow. I've made the yellow a little brighter, and highlighted the "sponsored links" text, to illustrate just how much of Google's results are not as independent and merit-based as you might think.



So how do you sponsor a link, and get your page into those yellow boxes? All you need to do is pay. It's a Google product called Adwords. You can read more about it on page 303.

The End. So, What Now?

Congratulations. That's the end of this book, and you've managed to make it all the way through. So where do you go from here? Hopefully you'll have followed some or all of the examples, and you'll have one or more web sites that do much of what you set out to achieve. But a web developer's work is never done, so there are probably more things that you're aching to do.

Sadly, this book can't cover everything about the subject. If you were to buy or borrow comprehensive books on everything we've covered here, such as HTML, CSS, Javascript, PHP, MySQL and web page design, it's likely that your collection would run to 10,000 pages or more.

This book is being expanded all the time, however, so check back at **`www.the-web-book.com`** regularly to ensure that you're always working from the latest revision. But if there's more that you want to find out, then the web, and especially a search engine, should be your next stop. Whether you need to know about one specific PHP function, or general advice about how to make your CSS-based designs look better, everything you need to know is out there if you know how to search. And because PHP is such a popular choice for writing software, there are hundreds of web sites and helpful forums devoted to it. Whatever you want to do, chances are that someone's already done much of the hard work for you and is willing to share the code.

In the future, I'm hoping to include some more advanced topics in The Web Book. These will include:

Ajax and DHTML
Database indexes
A login and authentication framework for building secure sites
MySQL table joins
PHP Sessions
Object-oriented PHP programming

The web is indeed a wonderful place. But you probably knew that already. I guess that's why you started reading this book. I just hope you enjoyed it, and that you'll tell your friends and colleagues to download their own copy too.

Appendix A – Building a Test Server

If you want to set up a test web server, and you already know the basics of things like FTP, php and mysql, this appendix will tell you how to do it. However, unless you specifically need (or want) to do this, it's perfectly acceptable to stick with the hosting space you've bought, both for hosting your live site and also for work in progress. In which case you can safely ignore this appendix. Building web servers is mostly for technophiles.

For setting up a test web server, you'll need a spare computer from which you don't mind deleting all data and programs. It needs to be Windows-compatible, even though we won't be running Windows on it. It doesn't need to be particularly new or powerful, though.

For a test web server, a server-oriented version of Linux is a good choice. Why server-oriented? Because the added GUI desktop is totally unnecessary. Why not Windows? Because this book is all about doing things as cheaply as possible, and you probably don't want to buy another copy of Windows for your test server.

The only thing you need to check is that your computer has a built-in Ethernet connector on the motherboard. If its networking capability comes via a USB or PCMCIA plug-in adaptor, chances are that it won't work without you getting involved in some substantial fiddling.

I'm going to use Ubuntu Server 7.10 for this project. It's ideally suited to the task, and it (and all the other components we'll be installing) is available totally free of charge. So if you fancy giving it a go, here's what you need to do.

Our Goal

By the end of this chapter you'll have a working Web server onto which users can safely and securely upload files via FTP. You'll also have Webmin installed, for remote admin functionality, plus Webalizer for generating web usage stats. Plus, you'll be able to host PHP/MySQL sites too.

Note that commands you need to type are in a bold Courier typeface **`like this`**.

Also, you are strongly advised to keep your new web server within the confines of your own LAN and use it purely for your own education and experimentation. Assuming your new server is connected to the internet via a broadband router, it won't be accessible by the world

in general unless you change your firewall settings in order to allow incoming connections on port 80. And frankly, that's the way it should stay. If you want to host real live web sites, open to the public, leave the hosting to the professionals.

First Install the OS

Get hold of a Ubuntu Server 7.10 CD, which you can download from http://releases.ubuntu.com/7.10/ubuntu-7.10-server-i386.iso. Yes, I know that this isn't the very latest version, but it's perfectly acceptable for this project and it works well. If you really want to get the latest version of Ubuntu Server instead then feel free, but the instructions below might not work exactly as you expect.

To get started, boot the PC from the CD-ROM. When asked, name your machine. I called mine webtest, but the precise name that you choose doesn't really matter.

Your PC is probably connected to the internet via a broadband or cable router that handles DHCP, in which case the Ubuntu installer should be able to make contact with the internet automatically and obtain from the router an IP address for your server. If it can't, you'll be asked to enter an IP address, netmask and gateway address. If this works, then that's just fine.

A word of advice. If Ubuntu can't detect a working network connection at all, that's probably because it doesn't have the necessary drivers available for your computer's ethernet socket. In which case, to be brutally frank, you should probably give up at this point. You won't have messed up your PC with a half-installed copy of Linux yet, and trying to troubleshoot Ubuntu networking is not something for amateurs.

Anyway, assuming that Ubuntu detected a network connection, you'll now be asked how to format the hard disk. Choose "guided – use entire disk".

When asked, choose a name and password for your day-to-day user account.

From the software selection menu, select only LAMP server. That's Linux, Apache, MySQL and PHP.

Next you'll be asked for a mysql root password. Be aware that you'll only be asked once (no confirmation required) and that the password you enter isn't shown on screen. So make sure you know what you're typing.

That's the basic Linux installation over. Remove the CD when prompted and the machine will restart. If you don't see a login: prompt after a minute or so, press Return and one

should appear. Remember that this is a server installation so there's no pretty graphical interface here (and thus no need to connect a mouse to this computer).

At the login: prompt, log in with the username and password you created earlier.

If the server acquired its own IP address using DHCP, you'll need to know what address it was allocated. Type ifconfig and have a look at what's listed for *inet addr*. You'll need to know this address in order to connect to the machine, unless you register a domain name that points to it or you add an entry to your company's internal DNS servers. If there's more than one entry under ifconfig, make an intelligent guess as to which is the right one. You may find, for example, that a non-existent wi-fi connection with an IP address of 127.0.0.1 is also shown.

From now on we'll assume that your server is on 192.168.1.10. Whenever you see this address mentioned below, substitute the correct address for your server.

So far, we've only got one account set up. We also need to set a password for the root (ie, administrator) account for when we need to do things that require root access. So type **`sudo passwd root`**, specify your current password when asked, then choose a password for the root username.

Linux doesn't normally allow you to log in as root directly so if/when you need to use your root privileges, log in with your normal user account and then type **`su`**, then enter the root password when prompted. In case you're wondering, it stands for super-user. If you ever forget who you're logged in as, the whoami command will tell you. Or look at the command prompt, which will end with $ for a normal user and # for a root user.

Some Useful Commands

Here are some useful commands to get you started, now that you've got a usable Linux system:

`shutdown -h now` turns off the computer.

`exit` logs you out. You'll need to do this twice if you used **`su`**. Remember that the web server is still running when you log out, so web/telnet connections to it will still work just fine. There's no need to remain logged in all the time.

`ls` shows a directory listing (that's LS, not 1S).

`ls -la` shows a better one (that's LS -LA).

`cd /` switches to the root directory.

`cd dirname` switches to the specified directory name, eg cd /etc.

`clear` clears the screen, like cls does in Windows.

`cat` is the linux version of the Windows "type" command if you want to display the contents of a text file.

`rm` deletes a file

`cp` is the linux equivalent of the DOS/Windows copy command.

`find / -name xyz.ext` will search the entire system for a file named xyz.ext

`pwd` (print working directory) tells you which directory you're currently in

Within an ls –la directory listing, lines that start with a "d" are directories, otherwise they're files. The other characters at the start of the line (such as rwxr--rw-) tell you who has permission to read, write, and execute the file. A Google search for chmod will tell you how to understand and change these.

Get Updated

Now we need to scan the internet for any important updates. The list of locations in which Ubuntu Server searches for updates is stored in a text file at /etc/apt/sources.list but the first entry in this file points to the Ubuntu Server CD-ROM. We need to remove this entry from the list, otherwise we'll keep getting prompted to insert the CD whenever we perform an update.

This file is read-only, so you'll need to be logged in as root (via su) to proceed beyond this point. In fact, everything that follows is best done as root (this is an exception rather than a rule - if you're not doing server maintenance, never log in as root).

Type **`cd /etc/apt`**
Type **`vi sources.list`**

You'll now find yourself facing vi, undoubtedly the worst editor ever invented. But without a GUI on your server you have little choice. Plus, it's very handy to know the basics of vi because it's part of every Linux and unix system.

To move the cursor up, down, left and right, use the k, j, h and l keys (I told you it was bad). To delete the character under the cursor, press x. That should be enough to allow you to delete any line that makes reference to "deb cdrom" and which isn't already commented out (ie, which doesn't have a # at the start).

If you mess up, type **:q!** and press return to abandon vi. If you manage to make it work, type **:w** to save the file and then **:q** to quit vi.

You won't have to use vi very often. Later on we'll install Webmin, which lets you maintain your server from another machine via a web browser. There's a proper file manager and editor built into Webmin, thankfully.

It's now time to update the system so that you're running the latest versions of everything.

Type **apt-get update** to update the catalogue of possible updates.
Then type **apt-get upgrade** to download and install any that need installing.

Note that apt-get may not work if your internet connection goes via a proxy server. Even if you entered the name of a proxy server when you first set up the machine and configured it with an IP address, apt-get doesn't take any notice. To fix this, type:

```
export http_proxy="http://yourproxy.com:80"
```

specifying the address (and port) of your company's proxy server. Then try the apt-get again.

Test Your Web Server

You should now have a basic working web server, although we're not finished yet. But you can test that everything is working by typing the server's IP address into a web browser on another machine on your LAN. You should see a web page with a link to apache2-default, and clicking on the link will bring up a brief message. Depending on your web browser, you may need to add http:// at the start of the address, eg type **http://192.168.1.10**.

Next we'll install a telnet server so that we can connect to the machine remotely over the LAN in command-prompt mode without the need to actually be seated at the server itself.

Install the Telnet Server

Type **apt-get install telnetd**

This will download and install the telnet server. Now we need to kick-start it, by typing:

```
/etc/init.d/openbsd-inetd restart
```

You can now log out by typing exit (you need to type it twice because you used the su command, and the first time just takes you back into non-root mode).

Everything we do from now on can be done remotely via telnet so, if you want to install the server in a hard-to-reach cupboard, that's no problem. You won't need physical access to the server again unless something goes wrong or if you need to turn it back on after a shutdown command.

To access your server type **`telnet 192.168.1.10`** (or whatever the IP address of your server is) from any machine on your LAN and you'll get a login prompt. You can do this from Windows or Linux or even a Mac.

An FTP server

Next, we need to install an FTP server so that people can upload HTML pages to your new web server. An ideal tool for this particular job is proFTPd (that's Unix-speak for the Pro FTP Daemon).

If you haven't done so already, telnet to your server and type **`su`** to get root access. Or you can work on the server directly if it's easier, of course.

We need to take a little care to set up the FTP server in a reasonably secure manner, even though this is only for test or educational purposes. We need to make sure that a user who logs into the FTP server in order to upload web pages can't browse the entire server but is locked into one directory. Also, a user who has an FTP username and password with which to upload web pages shouldn't be able to use those credentials to access the system via telnet, as that would grant them far too much power.

Type **`apt-get install proFTPd`** to download and install the FTP server. You'll be asked whether to choose an inetd installation or standalone. Choose inetd.

The basic FTP server is now up and running, and you should be able to log into it with your non-root account, using any FTP client application. But we still need to set up an account that will allow someone to upload their web pages without also having access to any other parts of the system.

First, switch to the /etc directory by typing **cd /etc**. We need to edit the file called shells and add a new line that says /bin/false to the file. Then, when we set up a new user account for our web user, we'll configure their account so that /bin/false is their command shell. Because there's no such shell, they won't be able to log in with telnet.

Type **vi shells** to edit the file. Use the cursor keys (h,j,k,l) to move the cursor to the start of a new line, then press i to enter insert mode. Press Return to insert a new line, and add /bin/false as a new line in the file. Press Esc to leave insert mode, save the file with **:w** then exit vi with **:q** and you're done.

Each user has a home directory which contains their various files. It's like My Documents in Windows and normally it resides in the /home directory. For web users, rather than setting their home directory to be somewhere within /home we'll put it under /var/www, which is the root of the web server.

Let's make an account for a user called webuser1 with a password of flintstone. These are the steps that you need to do for each web user account you want to create:

```
cd /var/www
mkdir webuser1
useradd webuser1 -p xxxx -d /var/www/webuser1 -s /bin/false chown
webuser1 webuser1
```
passwd webuser1 and, when asked, choose flintstone as the password.

Note that xxxx above is your root password, not the one that you want to assign for the webuser1 account.

Also note the chown command which changes the ownership of the webuser1 directory from root (which created it) to webuser1. If you don't do this, webuser1 won't be able to upload files.

Just to make sure that everything is working, verify that you can't telnet to the server using the webuser1 account. FTP access should work, but telnet should not.

Now create a simple index.html file and use FTP to upload it, using the webuser1/flintstone account. Then surf to http://192.168.1.10/webuser1 from any machine on your LAN and you should see the uploaded page.

Before we leave proFTPd, there are a couple of changes that we need to make to its configuration file in order to improve security and make things neater.

Type **cd /etc/proFTPd** and then **vi proFTPd.conf** to edit the config file. Move the cursor up and down with j and k until you reach the DefaultRoot line, and remove the # symbol from the start of the line by pressing the x key. This will lock all FTP users into their home directory (eg /var/www/webuser1) and won't let them view files that are further up the tree. Without this step, our webuser account holders could use their FTP software to browse the entire server's directory structure.

While you're in proFTPd.conf, add a new line near the top of the file which says:

```
IdentLookups        off
```

This will fix the problem which you'll no doubt have noticed, of a few seconds' delay when logging into the FTP server or uploading files.

You may also wish to change the ServerName entry from Debian to the name of your server, to make the welcome message more relevant. With vi, remember that typing i puts you into insert mode, for typing text, and Esc then puts you back into command mode from where you can type **:w** to save the file and **:q** to quit vi.

Webmin

Now that FTP is working, let's install Webmin so that we can remotely administer the server from anywhere on our LAN via a web browser. It's more fun and friendly than using telnet, and a great way to explore the machine.

First, make sure you're logged in as root (via your normal user account and su) then type the following, all on one line:

```
apt-get install openssl libnet-ssleay-perl libauthen-pam-perl libio-
pty-perl libmd5-perl
```

Then type (again, all on one line):

```
wget
http://prdownloads.sourceforge.net/webadmin/webmin_1.380_all.deb
```

Note that wget probably won't work if your internet connection goes through a proxy server. In which case, type:

```
export http_proxy="http://yourproxy.com:80"
```

first, and then issue the wget command.

Finally type **`dpkg -i webmin_1.380_all.deb`** and Webmin should be installed and ready to use.

From another machine on your LAN, surf to https://192.168.1.10:10000 and log in as root, using your server's root password. Note the https bit – it won't work with plain http. Also note the :10000, which is essential.

Ignore the warning about a missing SSL security certificate – you can trust this server unconditionally because it's yours. You will, though, need Java installed on the PC from which you intend to use Webmin, otherwise it won't work.

Possibly the most useful part of Webmin is the file manager, which also lets you edit files. You'll find it in the "others" category at the bottom of the left-hand menu.

Webalizer

Now we'll install Webalizer, which is a great tool that produces graphical stats to show your web site usage. Even if you're only using your server for test or educational purposes, it's useful to be able to see the sort of stats that are available with such programs.

To install webalizer type **`apt-get install webalizer`**

You need to tweak the Webalizer config file before the program will work. Type **`cd /etc/webalizer`** then **`vi webalizer.conf`** and delete the .1 from the end of the LogFile entry.

Webalizer produces its reports by analyzing the Apache web server log file on a regular basis. To make it do this, you need to set up what's called a cron job in order to run /usr/bin/webalizer regularly. Every 15 minutes should do nicely, and the easiest way to do this is via Webmin.

Go into Webmin via https://192.168.1.10:10000 from another PC and, under the System category, click on "Scheduled Cron Jobs". Then click "Create A New Scheduled Cron Job".

Choose to execute the job as root. The command to execute is **`/usr/bin/webalizer`**. Click on "Times And Dates Selected Below". Under the minutes, tick "Selected" and choose 0, 15, 30 and 45. For hours, days, months and weekdays, select "All".

Now click the Create button and close your web browser. After 15 minutes or so, surf to http://192.168.1.10/webalizer and you should see the reports and stats. Wait another 15 minutes and you should see an updated version.

PHP and MySQL

Now we need to make PHP and MySQL work, to ensure that we can host not just static HTML sites but also dynamic database-driven ones. PHP should already be working just fine, so we need to test that. Create a file called test.php which contains:

```
<?
echo "this is a test file";
?>
```

Upload it using the webuser1 account. Surf to http://192.168.1.10/webuser1/test.php and check that you see a web page containing just the message "this is a test file". If it works, PHP is working on your web server.

To allow users to create database-driven sites we'll install phpMyAdmin, which is a graphical web-based tool for managing MySQL databases. It's best if we don't allow web users to create their own databases, but we do want them to be able to manage the databases that we set up for them. PhpMyAdmin will work for both of these tasks. IE, for us to create databases and for our web users to maintain the tables within their allocated database.

As root, type **`apt-get install phpmyadmin`**

When asked which web server you're using, choose apache2.

To use phpMyAdmin, surf to http://192.168.1.10/phpmyadmin and log in with a username of root and the MySQL root password that you set up right at the start of this chapter.

On the front page of phpMyAdmin, scroll down to the Privileges link and click it. Then click "Add A New User". Enter their username (webuser1 in this case), and assign them a password. This will be used for them to log into phpmyadmin, and they'll also use it in their PHP code in order to connect to their database (using a host name of localhost). It's up to you whether you make it the same as their FTP password (flintstone). In this example, let's set the password as barney.

Click "Create database with same name and grant all privileges" and all the hard work will be done for you. A database called webuser1 will be created, with permission for the webuser1 account to do everything except creating new databases.

Log out of phpmyadmin (just close your browser), and then log in again. This time, use a username of webuser1 and a password of barney. You should see only the webuser1 database and no others, and you should find that you can create tables on the database but you can't create new databases.

You may also find that you can see a database called information_schema as well as your webuser1 database. However, this is harmless and can be ignored – it's not a security risk.

And that's it. You now have a fully working web server that you can use for test, development and training purposes.

As mentioned above, having your own web server for test purposes is fun, but isn't generally necessary. The rest of this book assumes that you have a hosting account somewhere, and that you'll use this account to hold your web sites rather than putting them on your own test server.