12
Building an Installer

Throughout the book, we have built up a CMS. This CMS works on your machine. The next step is to make sure that it works on other machines as well.

This chapter will cover topics such as:

- Creating a virtual machine using VirtualBox
- Creating the installer application
- Checking for missing features on the server

At the end of this chapter, you will be able to package and release your CMS for others to use, whether in-house or the general public.

This chapter shows how to detect various settings on the host machine. It is not a 100% complete installer, but it details how the tests can be done. As an example, I have not added a test to see that MySQL has been installed. You can add your own detection script for that.

Although it is possible to create and test the installer on your own machine, it's not ideal—you've already had the CMS installed and running, which means that your web server is already set up and compatible with the CMS's needs.

Ideally, you should create and test the installer on a machine which has not been carefully adjusted to match those needs. This way you can alert the eventual admin who installs the program of any missing requirements.

Because it is unrealistic to have a separate computer for testing each possible web server configuration, it is best to test using virtual machines.

Installing a virtual machine

A **Virtual Machine** (**VM**) is an emulation of an entire computer, which runs in a "host" machine, whether it is your laptop, a desktop, or even your company server.

There are many different VM engines out there. Examples include VMWare, QEMU, VirtualBox, Virtuozzo, Xen—and the list goes on.

In this chapter, we will use VirtualBox, because it's free, and it works on the four most popular server operating systems: Linux, Windows, Solaris, and OS X.

Installing VirtualBox

You can download a VirtualBox binary for your operating system from this place:

`http://www.virtualbox.org/wiki/Downloads`.

I will describe installation using Fedora (a Linux brand). If you are not using this brand or operating system, use the online material to properly install your program.

First, create a Yum repository configuration file. Yum is a downloader which automatically installs programs and their dependencies.

Create the file `/etc/yum.repos.d/virtualbox.repo` (from the root of your machine):

```
[virtualbox]
name=Fedora $releasever - $basearch - VirtualBox
baseurl=http://download.virtualbox.org/virtualbox/rpm/fedora/\
    $releasever/$basearch
enabled=1
gpgcheck=1
gpgkey=http://download.virtualbox.org/virtualbox/debian/\
    sun_vbox.asc
```

Next, install the program with this command:

```
[root@ryuk ~]# yum install VirtualBox
```

Note that if you are using a bleeding edge distribution, you may need to change `$releasever` to an older version number of Fedora. At the time of writing, the current release of Fedora is 13, and Sun (the makers of VirtualBox) has not updated their servers to include this, so I needed to change `$releasever` to `12` to get the installation to work:

```
baseurl=http://download.virtualbox.org/virtualbox/rpm/fedora/\
    12/$basearch
```

After installation, you may need to configure a kernel module to allow VirtualBox to run at a reasonable speed. Enter the following command:

```
[root@ryuk ~]# /etc/init.d/vboxdrv setup
```

This may require further dependencies, but any error message that shows up will tell you what those dependencies are.

After these steps, you will have a fully installed VirtualBox application. You can now start installing a guest operating system.

Installing the virtual machine

The operating system (OS) which runs inside the VM is called a **guest**. The main operating system of the machine is called the **host**.

Installing a guest OS is straightforward. It's almost the same as doing it on the real machine itself.

First, download an ISO image of the operating system you want to install. ISO is an archive format which is used to store DVDs and CDs as single files. When you download any OS from an online repository, it's usually in ISO format.

If you don't have fast bandwidth, you can also get installation DVDs from Linux magazines.

The choice of operating system is up to you, but you should keep the following in mind:

1. Don't choose an operating system that you are certain will never need to be supported. With my own CMS, I never expect it to be run on Windows. It may be run on any variety of Linux, though. And so, I won't try to run it on Windows but would choose a variant of Linux (including Unix variants such as perhaps OS X or Solaris, but with no great effort to make it work for those).
2. Don't choose a totally up-to-date operating system—the installer should work on older systems as well—such as systems that have older PHP installations, or missing Pear dependencies, or older versions of console applications such as ImageMagick.
3. Try to choose operating systems that are popular. Debian, CentOS, Fedora and, Ubuntu, are all very popular Linux variants for hosting. Don't focus all your energy on a variant that you are not likely to come across in professional life.

CentOS 5.2 fits the bill almost perfectly—it's a well-known variant of RedHat Enterprise Linux, is free to download, and has a very conservative upgrade history. While Ubuntu and Fedora push the limits of what can be done with Linux, CentOS is about stability.

You could also use Debian for exactly the same reason. I'm more comfortable with CentOS.

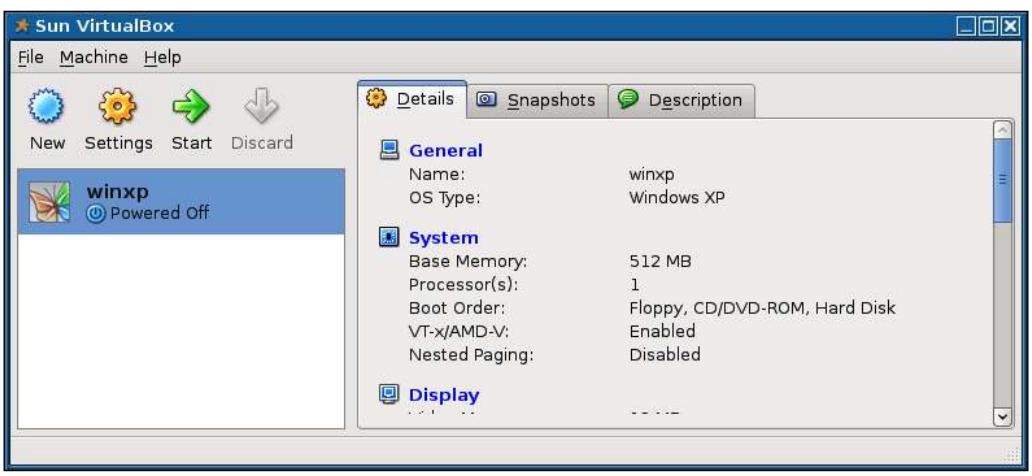
Go to `http://mirror.centos.org/centos/5/isos/` and download the appropriate ISO for your machine.

The files online at the time of writing are actually for Centos 5.5, but I have a copy of 5.2 here from a previous installation I did, and the process is identical in both cases (5.5 is more stable than 5.2, but apart from that, there are not many visible changes).

The file I have is called `CentOS-5.2-i386-bin-DVD.iso`, and the 5.5 equivalent is `CentOS-5.5-i386-bin-DVD.iso`.

Note that this is for installation on your own machine for testing purposes. If you were to use a production server, you should naturally use the latest version available, 5.5.

Okay—start up the program. In my machine, it's under **Applications | System Tools | Sun VirtualBox**:



You can see from the screenshot that I already had one virtual machine installed from earlier testing. This also illustrates that virtual machines can be used to install totally different guest operating systems than the host. The screenshot shows an installation of Windows XP which is inside a Fedora Linux host.

Click on **New**, and follow the setup wizard. Here are some suggested answers to the questions:

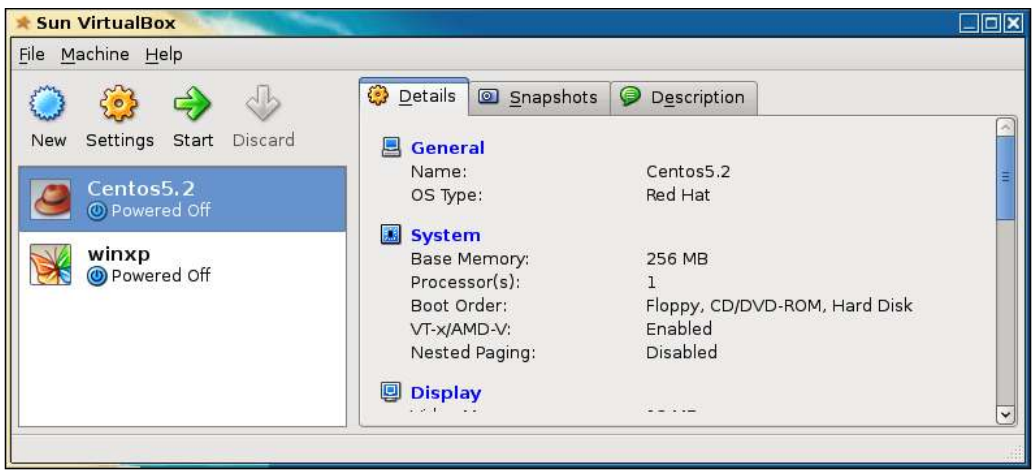
Name	Centos5.2
Operating System	Linux
Version	RedHat
Base Memory	256 MB

Boot Hard Disk should be checked, as well as **Create new Hard Disk**. Click on **Next** until the next question appears.

Dynamically expanding storage lets the fake hard disk of the VM expand if it's near full, so tick that.

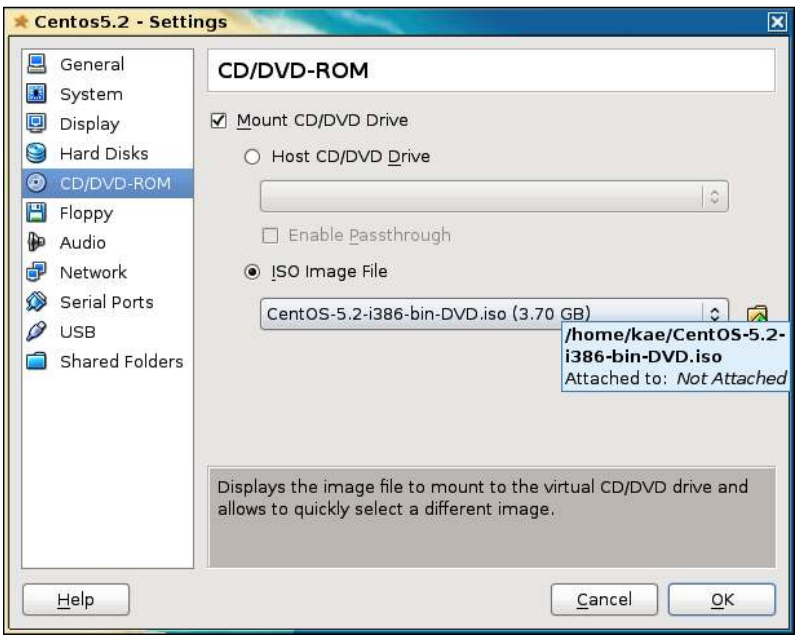
Accept the defaults for the rest of the wizard.

On completion, the screen should now appear similar to the next screenshot:



Now, make sure that **Centos5.2** is selected, then click on **Settings**.

Click on **CD/DVD-ROM**, tick **Mount CD/DVD Drive**, tick the **ISO Image file** box, and select the ISO file that you downloaded, as seen in the next screenshot:



Then click on **OK**.

Now click on **Start**.

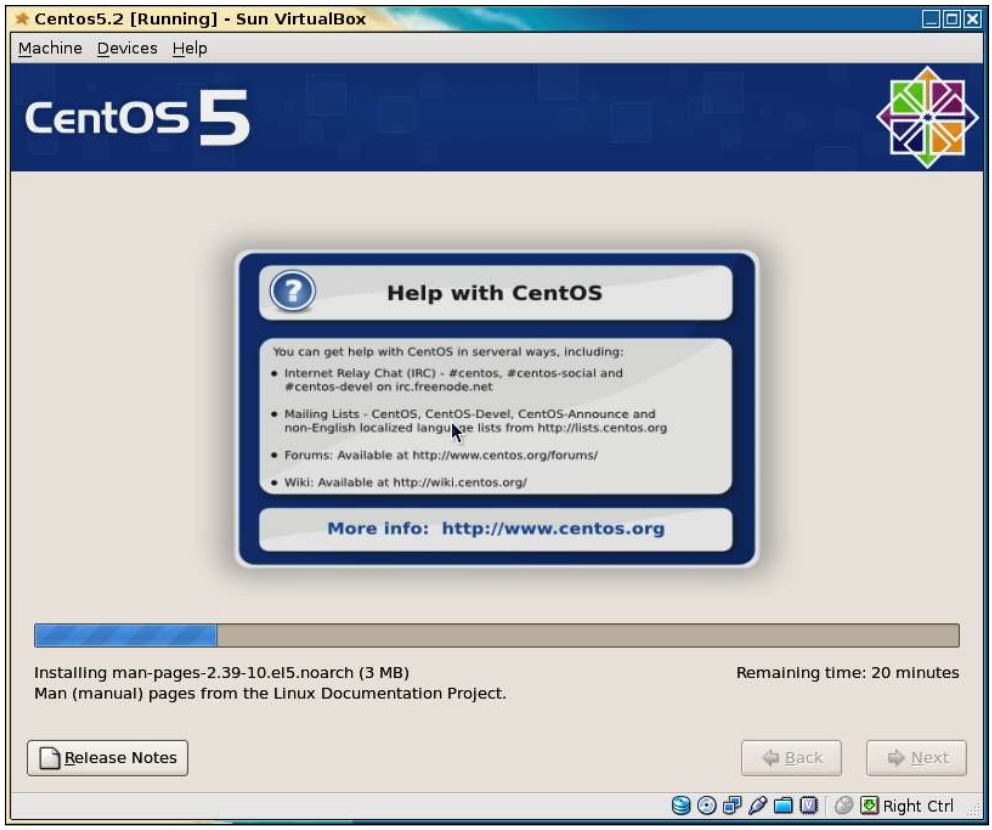
The Centos5.2 machine will boot up, acting like the ISO file you selected was a disk sitting in the virtual machine's DVD drive.

After a few moments, the installation will begin. Follow the wizard and install your guest operating system.

In most cases, the default selected option is the correct one, so if you're not sure about something, just click on OK—it's only a virtual image anyway, so if something goes wrong, you can always reinstall it.

When the wizard asks you which packages to install, accept the pre-selected packages and click on **Next**—we want a default installation which is not tailored to work with the CMS.

When the wizard is completed, the installation will take place. This can take anywhere from a few minutes up to half an hour or so:



Notice the **Right Ctrl** message in the bottom-right-hand side of the screen—when you click into the VM's window, your mouse and keyboard are trapped by the VM. To escape from it, so you can select other applications on the host, and use the *Right Ctrl* button on your keyboard to release the trap.

Another useful key combination is *Right Ctrl* + *F*, which will show the window in full screen, as if you had booted the laptop or desktop from the guest OS instead of your main OS.

When the installation is complete, click on the **Reboot** button in the guest screen. The virtual machine will reboot.

You will be asked a number of config questions—user setup and so on. Accept the defaults where possible. Create a user account when asked.

Okay—all done!

One final thing before we get to the CMS installation: When you log in, the machine will check for upgrades. Don't bother upgrading. Remember that we want the installer to work on older machines as well, and upgrading will therefore be pointless.

Installing the CMS in the VM

Now, log in to the guest OS. Open a console (**Applications** | **Accessories** | **Terminal**), `su` to `root` (use `su - root`, so we have access to `root`'s environment variables) and install the absolute minimum you would expect to find on a hosting platform—web server, MySQL database, and PHP:

```
[root@localhost: ~]# yum install httpd php-mysql php mysql\
    mysql-server
```

Note that the PHP version installed with the default CentOS installation will be out-of-date. In the case of 5.2, the version installed is 5.1.6. This is not recent enough for our CMS to run (it's missing `json_encode()`, for example), but is a good test for the installer.

When that's done, we can install the CMS code.

By default, an installation of Apache in CentOS (and most other Linux variants) will have its default webroot set to `/var/www/html`, so copy your CMS to that location using FTP, or HTTP, and so on.

Oh—before we go any further, open up the VirtualBox application again, go into settings for the CentOS 5.2 machine, and uncheck the box next to **Mount CD/DVD Drive**.

Otherwise, your VM may try to reinstall itself next time you turn it on.

Transferring the file from your laptop or desktop to the virtual machine can be done in many different ways, including setting up an FTP server on the guest or host or creating a shared directory where files are visible in both systems.

Personally, I chose to zip up the latest copy of the CMS, upload it to a file server I control, and download it through HTTP on the virtual machine. As a PHP developer, you no doubt also have a file server or web server somewhere where you can temporarily place a file while transferring it. This is probably the easiest way to transfer files without any prior preparation.

After unzipping, move all of the contents of your zipped file to `/var/www/html`, such that in that directory, you have the following files:

Note that this is from directly zipping up a working copy of the CMS.

Delete the `.private`, `f`, and `ww.cache` directories—they are specific to an installed copy of the CMS, and should not be there before the installer places them there:

```
[root@localhost html]# cd /var/www/html && rm -rf .private \

    f ww.cache
```

Your files are in place. Now we need to start up the web server and database server:

```
[root@localhost html]# /etc/init.d/httpd start && \

    /etc/init.d/mysqld start
```

And finally, let's make sure those two load up automatically the next time the virtual machine is turned on:

```
[root@localhost html]# chkconfig --add httpd && \

    chkconfig --add mysqld
```

Now that your files are in place and the web server is running, you can load up a browser (**Applications | Internet | Firefox Web Browser**), and point it at `localhost`:

The screen is blank because the server expects the `.private` directory to be there, and crashes because it's not found.

This is the point at which we can begin writing the installer.

Firstly though, make sure that your normal user account can edit the web files—you should not stay logged in as `root` while working:

```
[root@localhost html]# chown kae.kae /var/www/html -Rf
```

Change `kae` in the the command to whatever your user account is.

Creating the installer application

You can either edit the files within the VM, or edit them on the host machine and upload them to the VM after each change.

Personally, I prefer to edit things in-place, so all work in the rest of the chapter will happen within the virtual machine itself.

If you are comfortable with Vim (the console-based editor), then you're ready to go.

If not, then either install an editor you are comfortable with, or use gedit (**Applications | Accessories | Text Editor**), which is installed by default in most Linux systems.

Core system changes

The first thing we need to do is to intercept the failed loading of `/.private/config.php`, and redirect the browser to the installer.

Create the directory `/ww.installer` within `/var/www/html` (all directory references will be relative to `/var/www/html` in the rest of the chapter), and edit `/ww.incs/basics.php`:. Add the following highlighted new lines around the existing un-highlighted line:

```
if(file_exists(SCRIPTBASE . '.private/config.php')){
  require SCRIPTBASE . '.private/config.php';
}
else{
  header('Location: /ww.installer');
  exit;
}
```

This checks to see if the `.private/config.php` file exists, and if not, it is assumed that this is a fresh system, so the server redirects the browser into the `/ww.installer` directory.

And that's all we need to do to the core scripts. Now we can concentrate solely on the `/ww.installer` directory.

The installer

We need to first decide exactly what the job of the installer is.

In most systems, an installer handles three things:

1. Checking the environment to see if it is suitable for the application.
2. Choosing and entering starter configuration options.
3. Installation of the database.

Based on this, we can extend the installation process to this:

1. Check the environment, including:
    - PHP version. We need the `json_encode()` and `json_decode()` functions, so the version needs to be at least 5.2.
    - SQLite support, for KFM.
    - Writable `/f`, `/ww.cache`, and `/.private` directories.
2. Ask for any admin-entered configuration options needed for the system, including:
    - MySQL database access codes.
    - Details for the first administrator's user account.
3. Install the database, then save the configuration file to `/.private/config.php`.

You can see that the `config.php` file is created absolutely last. It is also obvious that if a `config.php` file already exists, then the installer has already been run, and should not be run again.

Therefore, the first thing we need to do is to create the `/ww.installer/index.php` file and tell it to exit if the installer has already been run:

```
<?php
if(file_exists('../.private/config.php'))exit;
?>
<html>
  <head>
    <script src="http://ajax.googleapis.com/ajax/libs/
        jquery/1.4.2/jquery.min.js"></script>
    <script src="http://ajax.googleapis.com/ajax/libs/
        jqueryui/1.8.0/jquery-ui.min.js"></script>
    <script src="/ww.installer/js.js"></script>
    <link rel="stylesheet" href="/ww.installer/css.css"
```

```
        type="text/css" />
  </head>
  <body>
    <div id="header"></div>
    <div id="content">please wait...</div>
  </body>
</html>
```

We will handle the configuration through jQuery.

Checking for missing features

First, we need to check for missing features. To do this, we will first show the list of required features, then use jQuery to ask the server about each of those features one at a time.

The reason we use jQuery here is purely visual—it looks better to have things appear to be checked and then removed from the screen, than to just have a PHP-generated page list the problems.

If all of those feature requirements are resolved, we automatically go forward to the next step, configuration. Installation on the ideal server will not display the first step at all, as there will be nothing to resolve.

Create the file `/ww.installer/js.js`:

```
function step1(){
  var tests=[
    ['php-version','PHP version 5.2 or higher'],
    ['sqlite','PDO SQLite version 3+'],
    ['f','Write permissions for user-files directory'],
    ['ww-cache','Write permissions for cache directory'],
    ['private','Write permissions for config directory']
  ]
  var html='';
  for(var i=0;i<tests.length;++i){
    html+='<div id="'+tests[i][0]+'" class="checking">'
      +'Checking: '+tests[i][1]
      +'</div>';
  }
  $('#content').html(html);
  $('.checking').each(function(){
    $.post('/ww.installer/check-'+this.id+'.php',
        step1_verify,'json');
  });
```

```
}
function step1_verify(res){
}
$(step1);
```

The last line is what starts everything off—when the page has loaded enough for JavaScript to safely run, the function `step1()` is called.

This function then draws out a list of points to the screen, as seen in the following screenshot:

After drawing the points to the screen, a separate HTTP request is called depending on the check.

For example, for the first item, where the array is `['php-version','PHP version 5.2 or higher']`, the file `/ww.installer/check-php-version.php` is called on the server.

The result is then passed to the stub function `step1_verify()`.

Let's write the first of the checking files, `/ww.installer/check-php-version.php`:

```
<?php
if(file_exists('../.private/config.php'))exit;

$vs=explode('.',phpversion());
if($vs[0]>5
  || ($vs[0]==5 && $vs[1]>=2)){
```

```
    echo '{"status":1,"test":"php-version"}';
    exit;
  }

  echo '{"status":0,"error":"'
    .'The PHP version must be at least 5.2. '
    .'It is currently '.phpversion().'",'
    .'"test":"php-version"}';
```

First, we check that the `config.php` file does not exist—we do not want people snooping on the capabilities of the server, so these checks should only be available while you're installing.

Next, we check that the version number is greater than 5.2. The version string returned by `phpversion()` is of the form "x.y.z", so the string needs to be separated into distinct numbers before it can be compared.

If the version number is higher, then a JSON object is returned containing a status variable which is true. Basically, it returns a "this is OK" message.

Otherwise, we return an error message.

Now, on the client-side, we want error messages to be highlighted to point out to the admin that something needs to be done. Edit `js.js` and replace the stub `step1_verify()` function with this:

```
function step1_verify(res){
  if(res.status==1){
    $('#'+res.test).slideUp(function(){
       $('#'+res.test).remove();
    });
    return;
  }
  $('#'+res.test)
    .addClass('error')
    .append('<p>'+res.error+'</p>');
}
```

This simple function first sees if the check was successful, in which case the displayed text for the check is removed in a sliding motion.

Otherwise, the element is highlighted as being in error. We add the class `"error"` to the displayed `<div>` and add some text explaining what went wrong. The explanation is sent by the server, allowing it to be tailored specifically to that server if you want.

The other tests are similarly done.

Here is the SQLite one, `check-sqlite.php`:

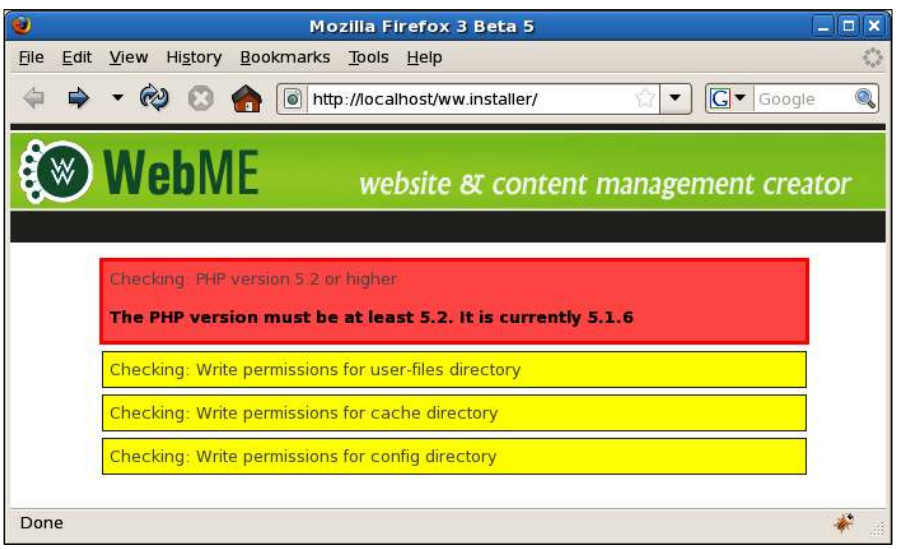
```
<?php
if(file_exists('../.private/config.php'))exit;

if(extension_loaded('pdo_sqlite')){
  echo '{"status":1,"test":"sqlite"}';
  exit;
}

echo '{"status":0,"error":"'
  .'You must have the PDO SQLite library installed. '
  .'"test":"sqlite"}';
```

The same process is followed. First, verify that the script is allowed to be run, then check to see if the requested feature can be verified, and finally, give an error message if it fails.

The default CentOS 5.2 installation we've installed does have SQLite installed as a PDO library, but doesn't have the required PHP version, as we can see here:



In this image, you can see that the PHP version has returned an error message with a clear description of the problem, and the SQLite test succeeded, so has been removed from the screen.

There are three remaining tests. They are all basically the same. Here's the first, `/ww.installer/check-f.php`:

```
<?php
if(file_exists('../.private/config.php'))exit;

$dname='f';

@mkdir('../'.$dname);
@file_put_contents('../'.$dname.'/test-permissions','test');
if(file_exists('../'.$dname.'/test-permissions')){
  echo '{"status":1,"test":"f"}';
  unlink('../'.$dname.'/test-permissions');
  exit;
}

$dir=preg_replace('/ww.installer$/',$dname,dirname(__FILE__));
if(!is_dir($dir))$error=$dir.' does not exist. '
    .'Please create it.';
else $error=$dir.' is not writable by the web server.';
echo '{"status":0,'
  .'"error":"'.$error.'",'
  .'"test":"f"}';
```

The other two are basically the same, but for `ww.cache` and `.private` respectively. Simply copy the above file to `check-ww-cache.php` and `check-private.php`, then replace the `$dname` value in both with `ww.cache` and `.private` respectively, and then change the returned `test` variable in the JSON to `ww-cache` and `private` respectively.

The next step is for you to resolve the pointed out problems for yourself.

In my case, that's done by upgrading the PHP version, then creating the user directories and making them writable by the server.

When the page is then reloaded, you end up with a blank content area.

We need to add a piece of code to watch for this and then move to `step 2`.

So, add this to the bottom of the `step1()` function (highlighted line):

```
  });
  setTimeout(step1_finished,500);
}
```

And then we need to add the `step1_finished()` function:

```
function step1_finished(){
  if($('.checking').length){
```

```
    setTimeout(step1_finished,500);
  }
  else step2();
}
function step2(){
}
```

This function loops every half-second until the checkpoints are gone. This should only loop once on any reasonable server and connection, but with slow bandwidth or a slow server, it will take a few seconds, and you'll see the valid points gradually vanish as they're checked and verified.

Adding the configuration details

Step two is to ask for information such as database and user details.

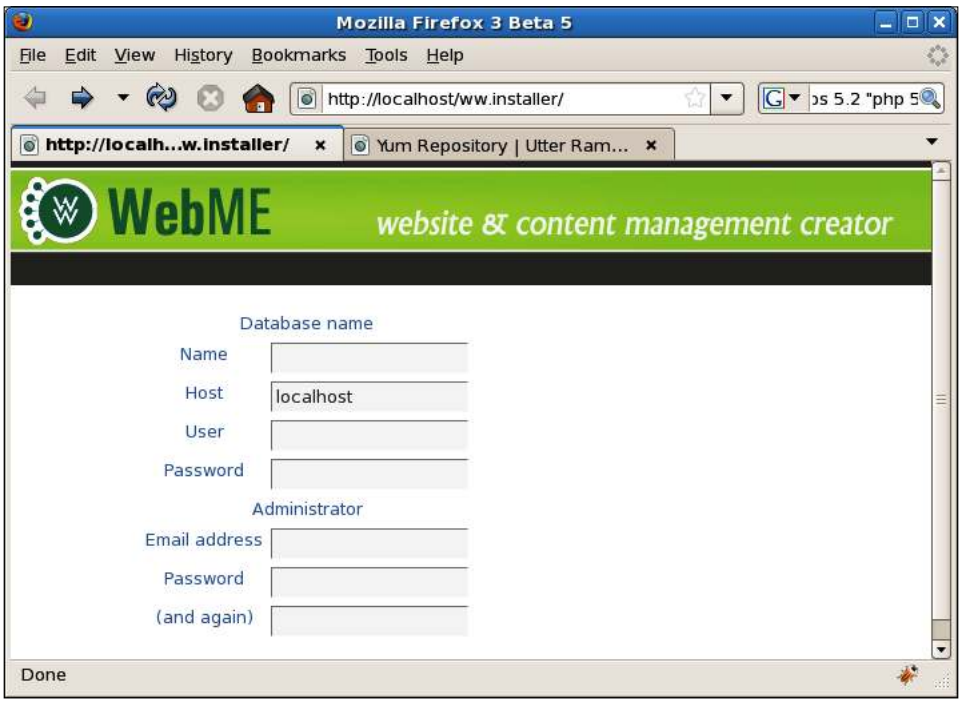
So, first, let's display the form. Replace the `step2()` stub function with this:

```
function step2(){
  $('#content').html('<table>'
    +'<tr><th id="db" colspan="2">Database name</th></tr>'
    +'<tr><th>Name</th><td><input id="dbname" /></td>'
        +'</tr>'
    +'<tr><th>Host</th><td>'
        +'<input id="dbhost" value="localhost" /></td></tr>'
    +'<tr><th>User</th><td><input id="dbuser" /></td></tr>'
    +'<tr><th>Password</th><td><input id="dbpass" /></td>'
        +'</tr>'
    +'<tr><th id="ad" colspan="2">Administrator</th></tr>'
    +'<tr><th>Email address</th><td><input id="admin" /></td>'
        +'</tr>'
    +'<tr><th>Password</th><td><input id="adpass" /></td>'
        +'</tr>'
    +'<tr><th>(and again)</th><td><input id="adpass2" /></td>'
        +'</tr>'
    +'</table><div class="error" id="errors"></div>');
  $('#content input').change(step2_verify);
}
function step2_verify(){
}
```

This will display a form with the most commonly needed configuration items. It can be expanded at a later date if you want it to include rarely changed things such as the database port and so on.

We print out the form to the page, and set a watch on the inputs such that any change to them will cause `step2_verify()` to call.

The form looks like this:



Now let's replace the `step2_verify()` stub form:

```
function step2_verify(){
  var opts={
    dbname:$('#dbname').val(),
    dbhost:$('#dbhost').val(),
    dbuser:$('#dbuser').val(),
    dbpass:$('#dbpass').val(),
    admin:$('#admin').val(),
    adpass:$('#adpass').val(),
    adpass2:$('#adpass2').val(),
  }
  $.post('/ww.installer/check-config.php',
    opts,step2_verify2,'json');
}
function step2_verify2(res){
}
```

This function just grabs all the input values and sends them to the server to be verified, which then returns its result to the `step2_verify2()` stub function.

Let's start with the verification file. Create `/ww.installer/check-config.php`:

```
<?php
if(file_exists('../.private/config.php'))exit;

$errors=array();

$dbname=@$_REQUEST['dbname'];
$dbhost=@$_REQUEST['dbhost'];
$dbuser=@$_REQUEST['dbuser'];
$dbpass=@$_REQUEST['dbpass'];
$admin=@$_REQUEST['admin'];
$adpass=@$_REQUEST['adpass'];
$adpass2=@$_REQUEST['adpass2'];

if($dbname=='' || $dbhost=='' || $dbuser==''){
  $errors[]='db requires name, hostname and username';
}
else{
  $db=mysql_connect($dbhost,$dbuser,$dbpass);
  if(!$db)$errors[]='db: could not connect - incorrect '
      .'details';
  else if(!mysql_select_db($dbname,$db)){
    $errors[]='db: could not select database "'
        .addslashes($dbname).'"';
  }
}
if(!filter_var($admin,FILTER_VALIDATE_EMAIL)){
  $errors[]='admin account must be an email address';
}
if(!$adpass && !$adpass2)$errors[]='admin password must not '
    .'be empty';
else if($adpass!=$adpass2)$errors[]='admin passwords must '
    .'both be equal';

echo json_encode($errors);
```

This checks the various submitted values, and builds up an array of error strings.

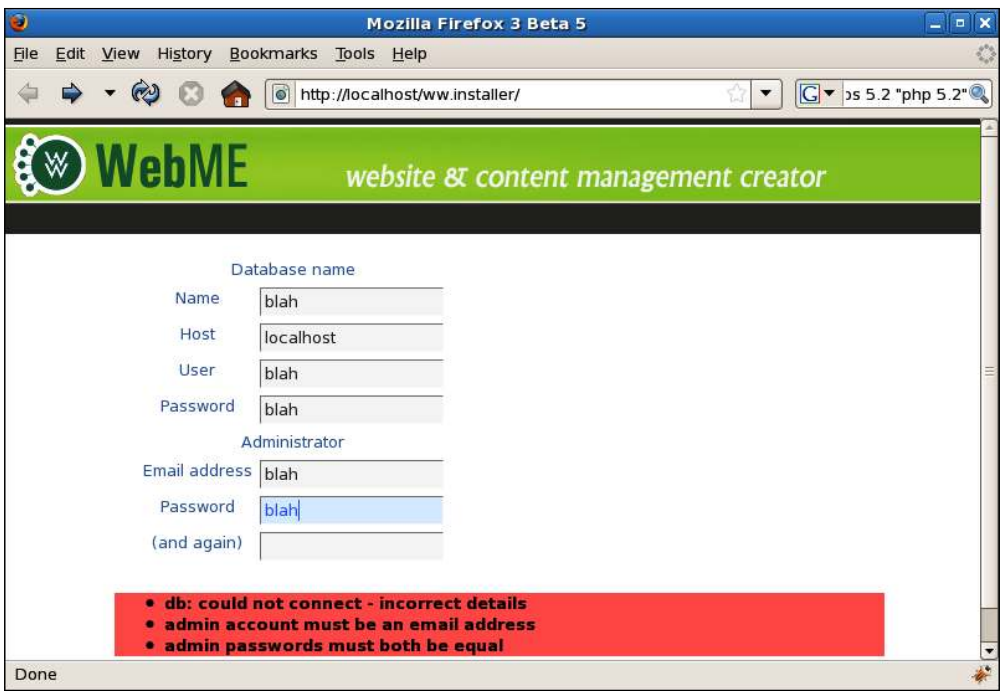
The error strings are then returned.

On the client-side, we then need to display the errors on the screen so the admin knows what to correct. Edit the `js.js` file again and replace the `step2_verify2()` stub function with this:

```
function step2_verify2(res){
  if(!res.length)return step3();
  var html='<ul>';
  for(var i=0;i<res.length;++i){
    html+='<li>'+res[i]+'</li>';
  }
  html+='</ul>';
  $('#errors').html(html);
}
function step3(){
}
```

So, if there are no errors returned, then `step3()` is called.

Otherwise, the errors are displayed in a `<ul>` list:



So what do we do when the values are finally correct?

We need to save the values to file, and then display a message explaining that we're done.

On the server-side, we know exactly when `step3` will be called, because that's where we're doing the checking, so all we need to do is to do the `step3` process (saving the values to the config file) if we find nothing wrong with the submitted values.

To do that, place the following code before the final line of `check-config.php` (highlighted):

```
if(!count($errors)){
  mysql_query('create table user_accounts(id int
      auto_increment not null primary key, email text,
      password char(32), active smallint default 0, groups
      text, activation_key char(32), extras text)default
      charset=utf8');
  mysql_query('insert into user_accounts values(1,"'
      .addslashes($admin).'", "'.md5($adpass).'",
      1, \'["_superadministrators"]\',"","")');
  mysql_query('create table groups(id int auto_increment not
      null primary key,name text)default charset=utf8');
  mysql_query('insert into groups values
      (1,"_superadministrators"),(2,"_administrators")');
  mysql_query('create table pages(id int auto_increment not
      null primary key,name text, body text, parent int, ord
      int, cdate datetime, special int, edate datetime, title
      text, template text, type varchar(64), keywords text,
      description text, associated_date date, vars text)
      default charset=utf8');
  $config='<?php $DBVARS=array('
    .'"username"=>"'.addslashes($dbuser).'",'
    .'"password"=>"'.addslashes($dbpass).'",'
    .'"hostname"=>"'.addslashes($dbhost).'",'
    .'"db_name"=>"'.addslashes($dbname).'");';
  file_put_contents('../.private/config.php',$config);
}
echo json_encode($errors);
```

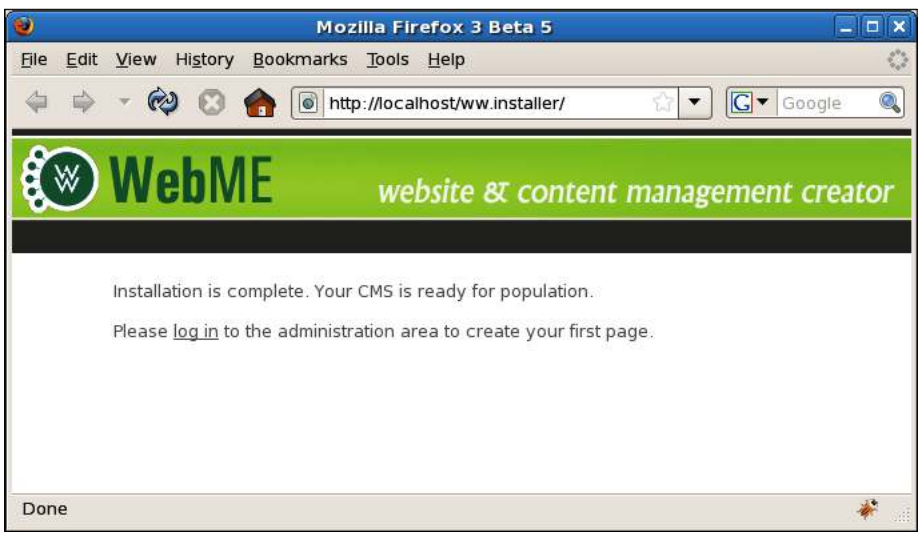
Because there are no errors, we know the configuration is complete and ready for installation, so we carry on with the installation, installing the core database tables and then recording the values to the config file `/.private/config.php`.

Finally, the result is returned to the client-side.

On the client-side, the result contains no errors, so we display a simple message saying it's all done. Replace the `step3()` stub function with this:

```
function step3(){
  $('#content').html(
    '<p>Installation is complete. Your CMS is ready for '
    +'population.</p>'
    +'<p>Please <a href="/ww.admin/">log in</a> to the '
    +'administration area to create your first page.</p>'
  );
}
```

And with that in place, the CMS is installed:



Note that this does not mean the installer is absolutely finished.

Other problems may arise later, due to unforeseen incompatibilities between the system and the CMS, which allowed the installation to complete, but don't allow some unspecified function to work.

When these things happen, all you can do is to find out what the problem is—maybe you are missing a certain PHP extension or Pear library—add a test for that problem to the installer, and re-run the installation to make sure that fixes it.

But, for the most part, this is it.