PHP Web Scraping

Get up and running with the basic techniques of web scraping using PHP

Jacob Ward

[PACKT] PUBLISHING

Instant PHP Web Scraping

Get up and running with the basic techniques of web scraping using PHP

Jacob Ward

[PACKT] open source
community experience distilled
PUBLISHING

BIRMINGHAM - MUMBAI

Instant PHP Web Scraping

First published: July 2013

Production Reference: 1220713

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham B3 2PB, UK.

ISBN 978-1-78216-476-0

www.packtpub.com

Credits

Author
Jacob Ward

Reviewers
Alex Berriman
Chris Nizzardini

Acquisition Editor
Andrew Duckworth

Commissioning Editor
Harsha Bharwani

Technical Editor
Krishnaveni Haridas

Project Coordinator
Esha Thakker

Proofreader
Elinor Perry-Smith

Production Coordinator
Kirtee Shingan

Cover Work
Kirtee Shingan

Cover Image
Abhinash Sahu

About the Author

Jacob Ward is a freelance software developer based in the UK. Through his background in research marketing and analytics he realized the importance of data and automation, which led him to his current vocation, developing enterprise-level automation tools, web bots, and screen scrapers for a wide range of international clients.

I would like to thank my mother for making everything possible and helping me to realize my potential.

I would also like to thank Jabs, Isaac, Sarah, Sean, Luke, and my teachers, past and present, for their unrelenting support and encouragement.

About the Reviewers

Alex Berriman is a seasoned young programmer from Sydney, Australia. He has degrees in computer science, and over 10 years of experience in PHP, C++, Python, and Java. A strong proponent of open source and application design, he can often be found late, working on a variety of applications and contributing to a range of open source projects.

Chris Nizzardini has been developing web applications in PHP since 2006. He lives and works in the beautiful Salt Lake City, Utah. You can follow Chris on twitter `@cnizzdotcom` and read what he has to say about web development on his blog (`www.cnizz.com`).

www.PacktPub.com

Support files, eBooks, discount offers and more

You might want to visit `www.PacktPub.com` for support files and downloads related to your book.

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at `www.PacktPub.com` and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at `service@packtpub.com` for more details.

At `www.PacktPub.com`, you can also read a collection of free technical articles, sign up for a range of free newsletters and receive exclusive discounts and offers on Packt books and eBooks.

`http://PacktLib.PacktPub.com`

Do you need instant solutions to your IT questions? PacktLib is Packt's online digital book library. Here, you can access, read, and search across Packt's entire library of books.

Why Subscribe?

- Fully searchable across every book published by Packt
- Copy and paste, print, and bookmark content
- On demand and accessible via web browser

Free Access for Packt account holders

If you have an account with Packt at `www.PacktPub.com`, you can use this to access PacktLib today and view nine entirely free books. Simply use your login credentials for immediate access.

Table of Contents

Preface

This book uses practical examples and step-by-step instructions to guide you through the basic techniques required for web scraping with PHP. This will provide the knowledge and foundation upon which to build web scraping applications for a wide variety of situations relevant to today's online data-driven economy.

What this book covers

Preparing your development environment (Simple), explains how to install and configure necessary software for development environment – IDE (Eclipse), PHP/MySQL (XAMPP) browser plugins for capturing live HTTP Headers, and Web Developer for setting environment variables.

Making a simple cURL request (Simple), explains how to request a web page using cURL, instructions and code for making a cURL request, and downloading a web page. The recipe also explains how it works, what is happening, and what the various settings mean. It also covers various options in cURL settings, and how to pass parameters in a GET request.

Scraping elements using XPath (Simple), explains how to convert a scraped page to a DOM object, how to scrape elements from a page based on tags, CSS hooks (class/ID), and attributes, and how to make a simple cURL request. It also discusses the instructions and code for completing a task, explains what XPath expressions and DOM are, and how the scrape works.

The custom scraping function (Simple), introduces a custom function for scraping content, which is not possible using XPath or regex. It also covers the instructions and code for the custom function, `scrapeBetween()`.

Scraping and saving images (Simple), covers the instructions and code for scraping and saving images as a local copy, and also verifying whether those images are valid.

Submitting a form using cURL (Intermediate), covers how to capture and analyze HTTP headers, how to submit (POST) a form, for example, a login form using cURL and cookies, or a web page with a form. It also covers the instructions on how to read HTTP headers for necessary info required to POST, instructions and code for posting using PHP and cURL, explanation of what is happening, how headers are being posted, and how to post multipart/upload forms.

Traversing multiple pages (Intermediate), explains topics such as identifying pagination, navigating through multiple pages, and associating scraped data with its source page.

Saving scraped data to a database (Intermediate), discusses creating a new MySQL database, using PDO to save the scraped data to a MySQL database, and accessing it for future use.

Scheduling scrapes (Simple), discusses how to schedule the execution of scraping scripts for complete automation.

Building a reusable scraping class (Advanced), introduces basic **object oriented programming** (**OOP**) principles to build a scraping class, which can be expanded upon and reused for future web scraping projects.

Bonus recipes covers topics such as how to recognize a pattern using regular expressions, how to verify the scraped data, how to retrieve and extract content from e-mails, and how to implement multithreaded scraping using multi-cURL. These recipes are available at `http://www.packtpub.com/sites/default/files/downloads/4760OS_Bonus_recipes.pdf`.

What you need for this book

Any basic knowledge of PHP or HTML will be useful, though not necessary

The following are the requirements:

- Eclipse
- Apache, PHP, and MySQL (XAMPP)

Download, installation, and configuration instructions are included in the *Preparing your development environment (Simple)* recipe.

Who this book is for

This book is aimed at those who are new to web scraping, with little or no previous programming experience. Basic knowledge of HTML and the Web is useful, but not necessary.

Conventions

In this book, you will find a number of styles of text that distinguish between different kinds of information. Here are some examples of these styles, and an explanation of their meaning.

Code words in text are shown as follows: " We create the `curlPost()` function, which is used to make a cURL request."

A block of code is set as follows:

```
<form action="/account" accept-charset="UTF-8" method="post"
id="packt-login-form">
```

New terms and **important words** are shown in bold. Words that you see on the screen, in menus or dialog boxes for example, appear in the text like this: " Select **Daily**, and then click on **Next**."

Warnings or important notes appear in a box like this.

Tips and tricks appear like this.

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book—what you liked or may have disliked. Reader feedback is important for us to develop titles that you really get the most out of.

To send us general feedback, simply send an e-mail to `feedback@packtpub.com`, and mention the book title via the subject of your message.

If there is a book that you need and would like to see us publish, please send us a note in the **SUGGEST A TITLE** form on `www.packtpub.com` or e-mail `suggest@packtpub.com`.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide on `www.packtpub.com/authors`.

Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

Downloading the example code

You can download the example code files for all Packt books you have purchased from your account at `http://www.PacktPub.com`. If you purchased this book elsewhere, you can visit `http://www.PacktPub.com/support` and register to have the files e-mailed directly to you.

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books—maybe a mistake in the text or the code—we would be grateful if you would report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting `http://www.packtpub.com/support`, selecting your book, clicking on the **errata submission form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded on our website, or added to any list of existing errata, under the Errata section of that title. Any existing errata can be viewed by selecting your title from `http://www.packtpub.com/support`.

Piracy

Piracy of copyright material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works, in any form, on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at `copyright@packtpub.com` with a link to the suspected pirated material.

We appreciate your help in protecting our authors, and our ability to bring you valuable content.

Questions

You can contact us at `questions@packtpub.com` if you are having a problem with any aspect of the book, and we will do our best to address it.

Instant PHP Web Scraping

Welcome to *PHP Web scraping*. Web scraping is the process of programmatically crawling and downloading information from websites and extracting unstructured or loosely structured data into a structured format.

This book assumes the reader has no previous knowledge of programming and will guide the reader through the basic techniques of web scraping through a series of short practical recipes using PHP, including preparing your development environment, scraping HTML elements using XPath, using regular expressions for pattern matching, developing custom scraping functions, crawling through pages of a website, including submitting forms and cookie-based authentication; logging in to e-mail accounts and extracting content, and saving scraped data in a relational database using MySQL. The book concludes with a recipe in which a class is built, using the information learned in previous recipes, which can be reused for future scraping projects and extended upon as the reader expands their knowledge of the technology.

Preparing your development environment (Simple)

There are a number of different IDEs available and the choice of which to use is a personal one, but for this book we will be working with Eclipse, specifically the **PHP Development Tools** (**PDT**) project from Zend. This is free to download, install, and use.

Getting ready

Before we can get to work developing our scraping tools, we first need to prepare our development environment. The essentials we will require are as follows:

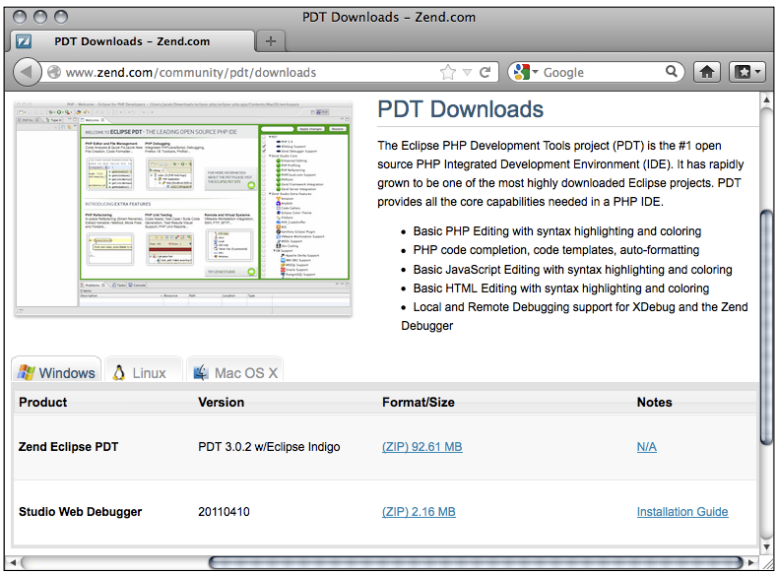
- An **Integrated development environment** (**IDE**) for writing our code and managing projects. PHP is the programming language we will be using, for executing our code.
- MySQL as a database for storing our scraped data.
- phpMyAdmin for easy administration of our databases. PHP, MySQL, and phpMyAdmin can be installed separately. However, we will be installing the XAMPP package, which includes all of these, along with an additional software, for example Apache server, which will come handy in the future if you develop your scraper further.

After installing these tools, we will adjust the necessary system settings and test that everything is working correctly.

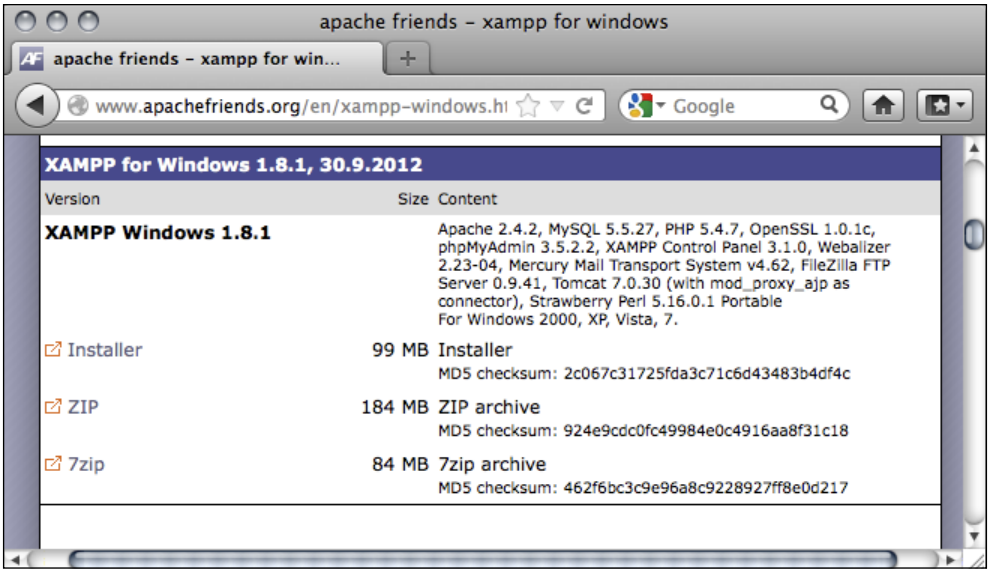
How to do it...

Now, let's take a look at how to prepare our development environment, by performing the following steps:

1. In this first set of steps, we will install our development environment, Zend Eclipse PDT.
2. Visit: `http://www.zend.com/en/community/pdt/downloads`.
3. Select the **Zend Eclipse PDT download** option for your operating system, as shown in the screenshot, and save the ZIP file to your computer.

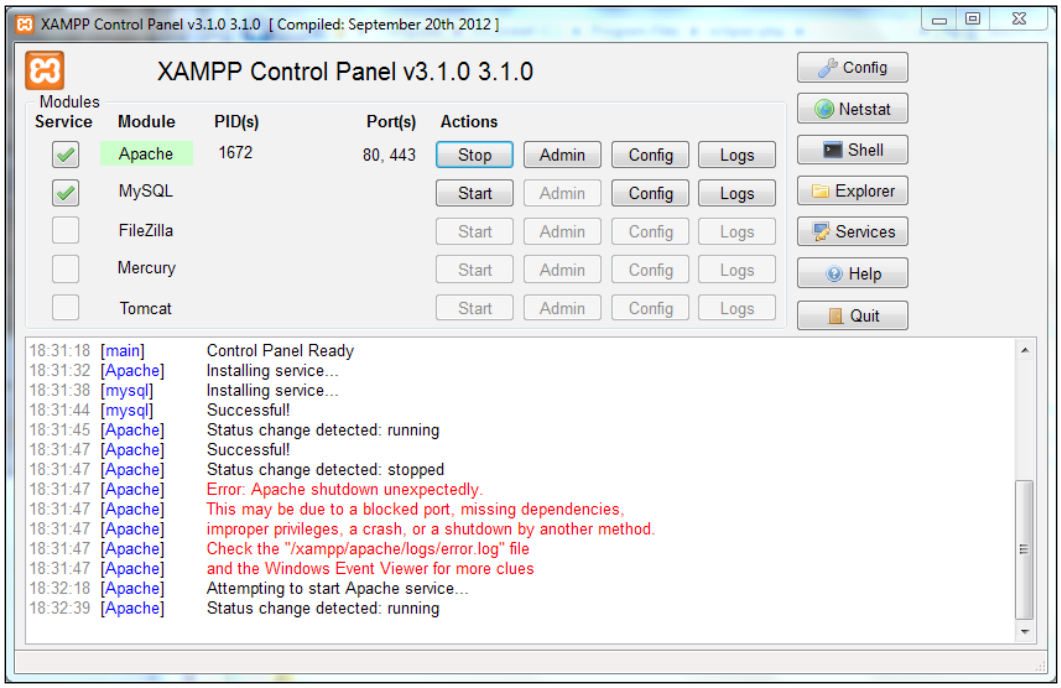


4. Once the file has been downloaded, unzip the contents. The resulting directory, `eclipse-php`, is the eclipse program folder. Drag-and-drop this into the `C:\Program Files` directory on your computer.
5. Next, we will install XAMPP, which includes PHP, MySQL, phpMyAdmin, and Apache.
6. Visit the following URL and download the latest version of XAMPP, following the installation instructions on the web page `http://www.apachefriends.org/en/xampp-windows.html`, as shown in the following screenshot:



7. Upon successful installation, start XAMPP for the first time and select the following components to install:
   - **XAMPP** – XAMPP Desktop Icon
   - **Server** – MySQL, Apache
   - **Program Languages** – PHP
   - **Tools** – phpMyAdmin
8. Save in the default destination.
9. Click on **Install** and the chosen programs will install.
10. Double-click on the XAMPP desktop icon to launch the XAMPP control panel.
11. In the XAMPP control panel start Apache and MySQL by performing the next set of steps.
12. Click on the **Start** button for **Apache**.

13. Click on the **Start** button for **MySQL**.

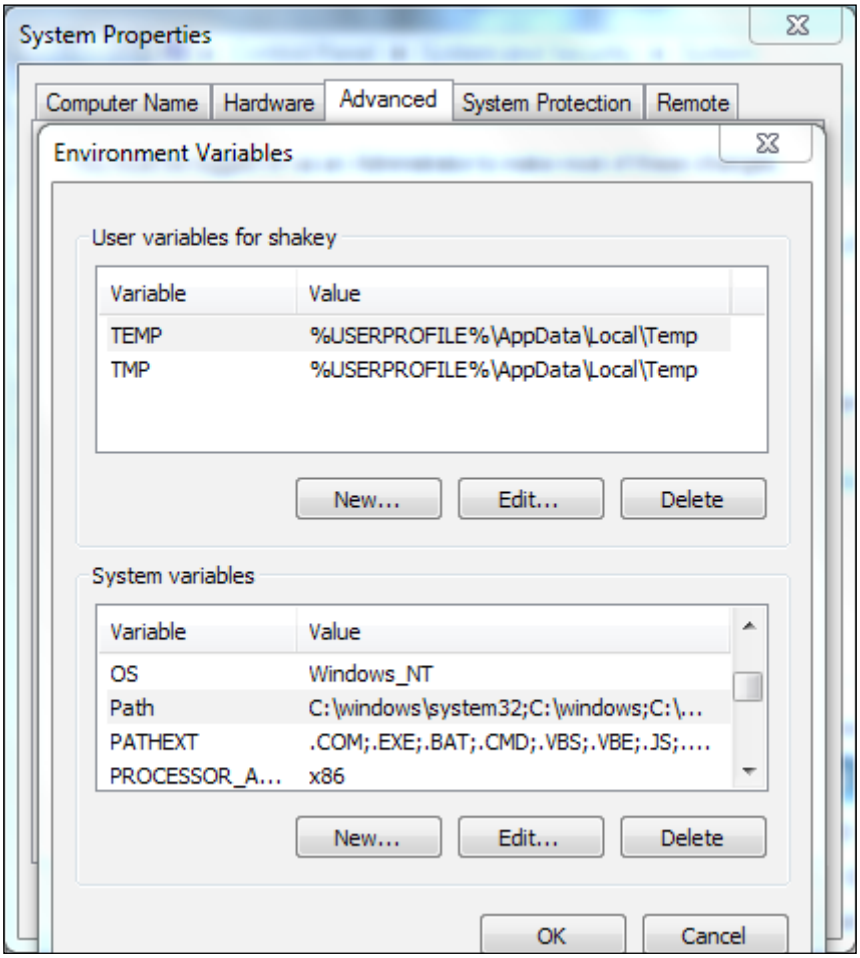


14. With the necessary software and tools installed, we need to set our PHP path variable.
15. Navigate to **Start** | **Control Panel** | **System and Security** | **System**.
16. In the left menu bar click on **Advanced system settings**.
17. In the **System Properties** window select the **Advanced** tab, and click on the **Environment variables...** button
18. In the **Environment Variables** window there are two lists, **User variables** and **System variables**. In the **System variables** list, scroll down to the row for the `Path` variable. Select the row and click on the **Edit** button.

Downloading the example code

You can download the example code files for all Packt books you have purchased from your account at `http://www.PacktPub.com`. If you purchased this book elsewhere, you can visit `http://www.PacktPub.com/support` and register to have the files e-mailed directly to you.

19. In the textbox for variable's value: add to the end of the line the directory in which PHP is installed, `C:\xampp\php`, and then click on **OK**, as given in the following screenshot:



20. The PHP directory will now be in our path variables.

21. Finally we need to ensure that cURL is enabled in PHP. Navigate to our XAMPP installation directory, then in to the `php` directory and open the file `php.ini` for editing.

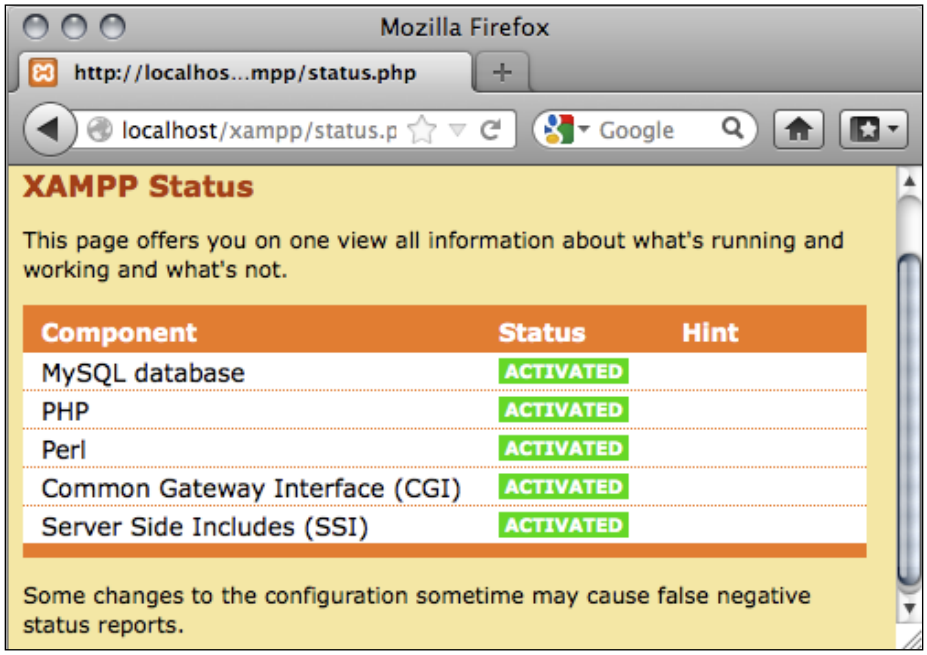
22. Find the following line and remove the semicolon from the beginning of it:

```
;extension=php_curl.dll
```

23. Save the file and close the text editor.

24. In the XAMPP control panel, restart Apache.

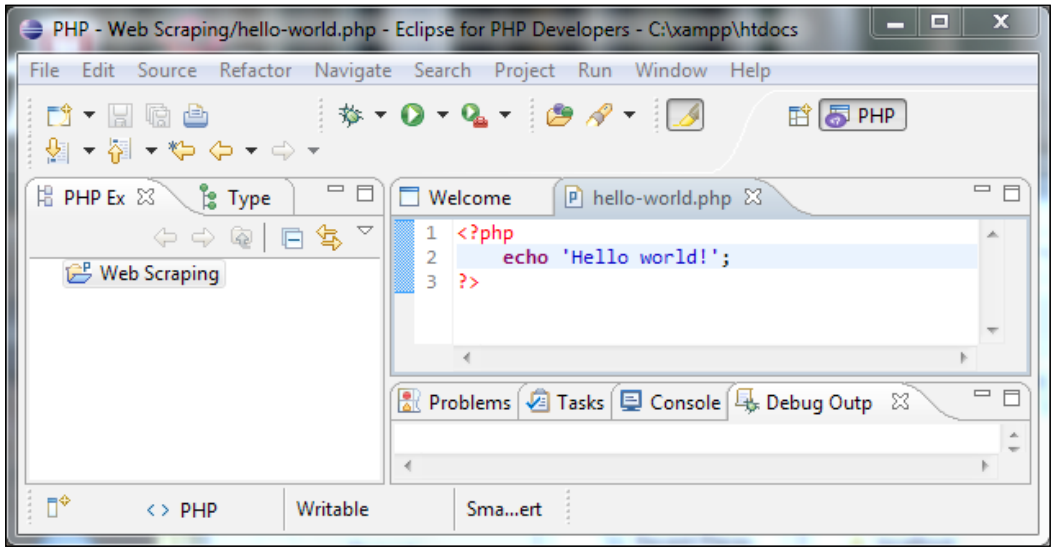
25. We can now test whether the installation is working correctly by opening our web browser and visiting `http://localhost/xampp/status.php` or `http://127.0.0.1/xampp/status.php URL` and make sure that **PHP** and **MySQL database** are both **ACTIVATED**, as shown in the following screenshot:



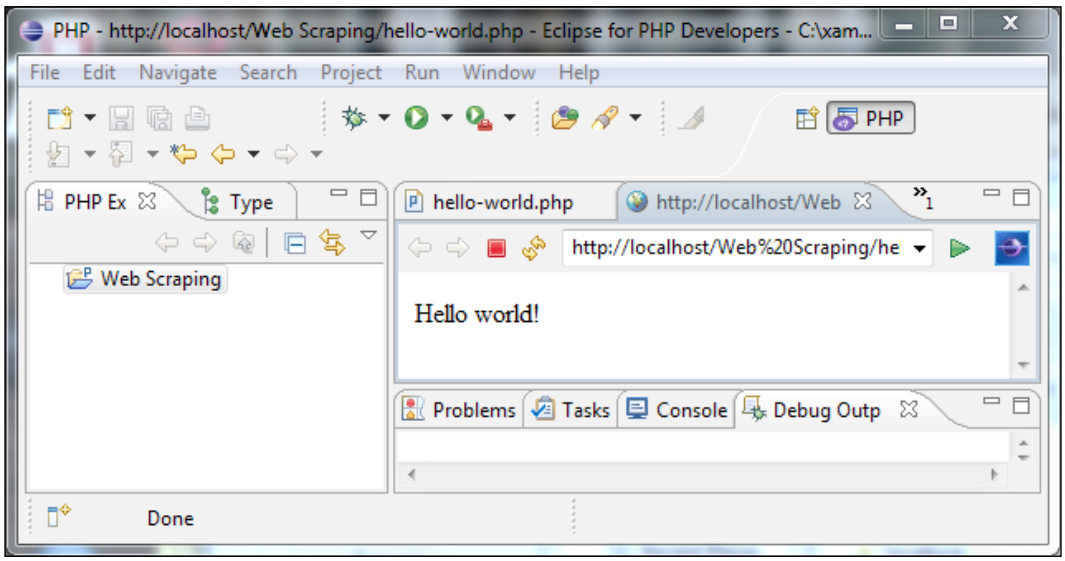
26. The final step is to create a new project in Eclipse and execute our program.
27. We start Eclipse by navigating to the folder in which we saved it earlier and double-clicking on the **eclipse-php** icon.
28. We are asked to select our **Workspace**. Browse to our `xampp` directory and then navigate to `htdocs`, for example `C:\xampp\htdocs` and click on **OK**.
29. Once Eclipse has started, navigate to **File** | **New** | **PHP Project**. Leave all of the settings as they are and name our project as `Web Scraping`. Click on **Next**, and then click on **Finish**.
30. Now we are ready to write our first script and execute it. Navigate to **File** | **New** | **PHP File**, leave the source folder as `Web Scraping` and name the PHP file as `hello-world.php`, and then click on **Finish**, and once we have created our first PHP file, be ready to type some code into it.

31. Enter the following code into Eclipse, as show in the following screenshot:

```
<?php
  echo 'Hello world!';
?>
```



Now from the top menu, click on the **Run** tab and our code will be executed. We will see the text **Hello world!** on screen as in the following screenshot:



How it works...

Let's look at how we performed the previously defined steps in detail:

1. After installing our required software, we set our PHP path variable. This ensures that we can execute PHP directly from the command line by typing `php` rather than having to type the full location of our PHP executable file, every time we wish to execute it.
2. In the next step we ensure that whether cURL is enabled in PHP. cURL is the library which we will be using to request and download target web pages.
3. We then check that everything is installed correctly by visiting the XAMPP status page.
4. Using the final set of steps, we set up Eclipse, and then create a small PHP program which echoes the text Hello world! to the screen and execute it.

Making a simple cURL request (Simple)

In PHP the most common method to retrieve a web resource, in this case a web page, is to use the cURL library, which enables our PHP script to send and receive HTTP requests to and from our target web server.

When we visit a web page in a client, such as a web browser, an HTTP request is sent. The server then responds by delivering the requested resource, for example an HTML file, to the browser, which then interprets the HTML and renders it on screen, according to any associated styling specification. When we make a cURL request, the server responds in the same way, and we receive the source code of the Web page which we are then free to do with as we will in this case perform by scraping the data we require from the page.

Getting ready

In this recipe we will use cURL to request and download a web page from a server.

Refer to the *Preparing your development environment* recipe.

How to do it...

1. Enter the following code into a new PHP project:

```
<?php

  // Function to make GET request using cURL
  function curlGet($url) {

    $ch = curl_init();     // Initialising cURL session

    // Setting cURL options
```

```
    curl_setopt($ch, CURLOPT_RETURNTRANSFER, TRUE);
    curl_setopt($ch, CURLOPT_URL, $url);

    $results = curl_exec($ch);  // Executing cURL session

    curl_close($ch);  // Closing cURL session

    return $results;  // Return the results
  }

  $packtPage = curlGet('http://www.packtpub.com/oop-php-5/book');

  echo $packtPage;

?>
```

2. Save the project as `2-curl-request.php` (ensure you use the `.php` extension!).
3. Execute the script.
4. Once our script has completed, we will see the source code of `http://www.packtpub.com/oop-php-5/book` displayed on the screen.

How it works...

Let's look at how we performed the previously defined steps:

1. The first line, `<?php`, and the last line, `?>`, indicate where our PHP code block will begin and end. All the PHP code should appear between these two tags.
2. Next, we create a function called `curlGet()` , which accepts a single parameter `$url`, the URL of the resource to be requested.
3. Running through the code inside the `curlGet()` function, we start off by initializing a new cURL session as follows:

```
$ch = curl_init();
```

4. We then set our options for cURL as follows:

```
curl_setopt($ch, CURLOPT_RETURNTRANSFER, TRUE);
// Tells cURL to return the results of the request (the source
code of the target page) as a string.

curl_setopt($ch, CURLOPT_URL, $url);
// Here we tell cURL the URL we wish to request, notice that it is
the $url variable that we passed into the function as a parameter.
```

5. We execute our cURL request, storing the returned string in the `$results` variable as follows:

```
$results = curl_exec($ch);
```

6. Now that the cURL request has been made and we have the results, we close the cURL session by using the following code:
   ```
   curl_close($ch);
   ```
7. At the end of the function, we return the `$results` variable containing our requested page, out of the function for using in our script.
   ```
   return $results;
   ```
8. After the function is closed we are able to use it throughout the rest of our script.
9. Later, deciding on the URL we wish to request, `http://www.packtpub.com/oop-php-5/book`, we execute the function, passing the URL as a parameter and storing the returned data from the function in the `$packtPage` variable as follows:
   ```
   $packtPage = curlGet('http://www.packtpub.com/oop-php-5/book');
   ```
10. Finally, we echo the contents of the `$packtPage` variable (the page we requested) to the screen by using the following code:
    ```
    echo $packtPage;
    ```

There's more...

There are a number of different HTTP request methods which indicate the server the desired response, or the action to be performed. The request method being used in this recipe is cURLs default `GET` request. This tells the server that we would like to retrieve a resource.

Depending on the resource we are requesting, a number of parameters may be passed in the URL. For example, when we perform a search on the Packt Publishing website for a query, say, `php`, we notice that the URL is `http://www.packtpub.com/books?keys=php`. This is requesting the resource `books` (the page that displays search results) and passing a value of `php` to the `keys` parameter, indicating that the dynamically generated page should show results for the search query `php`.

More cURL Options

Of the many cURL options available, only two have been used in our preceding code. They are `CURLOPT_RETURNTRANSFER` and `CURLOPT_URL`. Though we will cover many more throughout the course of this book, some other options to be aware of, that you may wish to try out, are listed in the following table:

Option Name	Value	Purpose
`CURLOPT_FAILONERROR`	`TRUE` or `FALSE`	If a response code greater than `400` is returned, cURL will fail silently.
`CURLOPT_FOLLOWLOCATION`	`TRUE` or `FALSE`	If `Location:` headers are sent by the server, follow the location.

Option Name	Value	Purpose
CURLOPT_USERAGENT	A user agent string, for example: 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10.5; rv:15.0) Gecko/20100101 Firefox/15.0.1'	Sending the user agent string in your request informs the target server, which client is requesting the resource. Since many servers will only respond to 'legitimate' requests it is advisable to include one.
CURLOPT_HTTPHEADER	An array containing header information, for example: array('Cache-Control: max-age=0', 'Connection: keep-alive', 'Keep-Alive: 300', 'Accept-Language: en-us,en;q=0.5')	This option is used to send header information with the request and we will come across use cases for this in later recipes.

A full listing of cURL options can be found on the PHP website at `http://php.net/manual/en/function.curl-setopt.php`.

The HTTP response code

An HTTP response code is the number that is returned, which corresponds with the result of an HTTP request. Some common response code values are as follows:

- **200**: OK
- **301**: Moved Permanently
- **400**: Bad Request
- **401**: Unauthorized
- **403**: Forbidden
- **404**: Not Found
- **500**: Internal Server Error

It is often useful to have our scrapers responding to different response code values in a different manner, for example, letting us know if a web page has moved, or is no longer accessible, or we are unauthorized to access a particular page.

In this case, we can access the response of a request using cURL by adding the following line to our function , which will store the response code in the `$httpResponse` variable:

```
$httpResponse = curl_getinfo($ch, CURLINFO_HTTP_CODE);
```

Scraping elements using XPath (Simple)

Now that we have requested and downloaded a web page, as mentioned in the *Making a simple cURL request* recipe we can now proceed to scrape the data that we require.

XPath can be used to navigate through elements in an XML document. In this recipe we will convert our downloaded web page into an XML DOM object, from which we will use XPath to scrape the required elements based on their tags and attributes, such as CSS classes and IDs.

How to do it...

1. Enter the following code into a new PHP project:

```
<?php

// Function to make GET request using cURL
function curlGet($url) {
  $ch = curl_init();       // Initialising cURL session
  // Setting cURL options
  curl_setopt($ch, CURLOPT_RETURNTRANSFER, TRUE);
  curl_setopt($ch, CURLOPT_FOLLOWLOCATION, TRUE);
  curl_setopt($ch, CURLOPT_URL, $url);
  $results = curl_exec($ch);      // Executing cURL session
  curl_close($ch);  // Closing cURL session
  return $results;  // Return the results
}

$packtBook = array();  // Declaring array to store scraped book
data.
// Function to return XPath object
function returnXPathObject($item) {
  $xmlPageDom = new DomDocument();      // Instantiating a new
DomDocument object
  @$xmlPageDom->loadHTML($item); // Loading the HTML from
downloaded page
  $xmlPageXPath = new DOMXPath($xmlPageDom);  // Instantiating new
XPath DOM object
return $xmlPageXPath;      // Returning XPath object
}

$packtPage = curlGet('http://www.packtpub.com/learning-ext-js/
book');  // Calling function curlGet and storing returned results
in $packtPage variable

$packtPageXpath = returnXPathObject($packtPage);  // Instantiating
new XPath DOM object
```

```
$title = $packtPageXpath->query('//h1');  // Querying for <h1>
(title of book)

// If title exists
if ($title->length > 0) {
  $packtBook['title'] = $title->item(0)->nodeValue;  // Add title
to array
}

$release = $packtPageXpath->query('//span[@class="date-display-
single"]');  // Querying for <span class="date-display-single">
(release date)

// If release date exists
if ($release->length > 0) {
  $packtBook['release'] = $release->item(0)->nodeValue;  // Add
release date to array
}

$overview = $packtPageXpath->query('//div[@class="overview_
left"]');  // Querying for <div class="overview_left">

// If overview exists
if ($overview->length > 0) {
  $packtBook['overview'] = trim($overview->item(0)->nodeValue);
// Trim whitespace and add overview to array
}

$author = $packtPageXpath->query('//div[@class="bpright"]/div[@
class="author"]/a');  // Querying for all authors

// If authors exist
if ($author->length > 0) {
  // For each author
  for ($i = 0; $i < $author->length; $i++) {
    $packtBook['authors'][] = $author->item($i)->nodeValue;  //
Add author to 2nd dimension of array
  }
}

print_r($packtBook);

?>
```

2. Save the project as `3-xpath-scraping.php`.
3. Execute the script.

4. We will see the results of our scrape displayed on the screen, as follows:

```
Array
(
    [title] => Learning Ext JS
    [release] => November 2008
    [overview] => Learn to build consistent, attractive web
interfaces with the framework components.
Integrate your existing data and web services with Ext JS data
support.
Enhance your JavaScript skills by using Ext's DOM and AJAX
helpers.
Extend Ext JS through custom components.
    [authors] => Array
        (
            [0] => Colin Ramsay
            [1] => Shea Frederick
            [2] => Steve 'Cutter' Blades
        )
)
```

How it works...

Let's look at how these steps were performed:

1. Firstly, we have included the `curlGet()` function that we created in the *Making a simple cURL request* recipe , which enables us to reuse this functionality to request the URL we are going to scrape.
2. Next, we create a new function `returnXPathObject()`, which takes a resource, in this case an HTML document, and then returns an XPath object for us to work with. The code inside the function is as follows:
    - The first line instantiates a new `DomDocument` object as follows:

```
$xmlPageDom = new DomDocument();
```

    - The next line takes our HTML resource and loads that into our DomDocument object by using the following code:

```
@$xmlPageDom->loadHTML($item);
```

Notice that when we load our HTML into the `$packtPageDom` object the statement is preceded by `@`. This instructs the procedure to execute without throwing errors. This is necessary, because in almost every case, an HTML file on the Web will contain an invalid markup. This is an unavoidable reality, so we wish to ignore any errors found that would otherwise cause our script to fail.

- From this `DomDocument` object, we then create a new XPath Dom object, which is then returned from the function for use in our scraper as follows:

```
$xmlPageXPath = new DOMXPath($xmlPageDom);
```

3. We then execute the `curlGet()` function, passing our URL, `http://www.packtpub.com/learning-ext-js/book`, as a parameter as follows:

```
$packtPage = curlGet('http://www.packtpub.com/learning-ext-js/book');
```

4. With our resource downloaded, we can now convert it to an XPath `DOM` object in order to scrape our required data from it. We do this by calling our `returnXPathObject()` function, passing our resource as a parameter by using the following code:

```
$packtPageXpath = returnXPathObject($packtPage);
```

5. With our XPath DOM object now ready, we can proceed with scraping the required data. If we take a look at the source code of the page we are scraping, `http://www.packtpub.com/learning-ext-js/book`, we can identify the data we wish to scrape, and importantly, the HTML tags between which it appears. The title appears between the `<h1></h1>` tags, the release date appears between the `<span class="date-display-single"></span>` tags, the overview between `<div class="bpright"><div class="overview"></div></div>`, and the authors each appear between the anchor (`<a></a>`) tags, all of these are enclosed in the `<div class="author"></div>` tags.

6. Firstly, we'll scrape the title of the book. In order to do this, we run the query method of our `$packtPageXpath` object, with an XPath expression tailored to return the data we require. In this case, we require the contents of `<h1></h1>`. Since there is only one occurrence of `<h1></h1>`, we can access it by using the expression `//h1`. This instructs XPath to search through all nodes `//` for the `h1` element. This is returned and stored in the `$title` object. We then use an `if` statement to check if the `$title` object contains a title, by checking that is it longer than `0` and if so, we access this at `$title->item(0)->nodeValue`, and then assign it to the array and the `$packtBook['title']` key by using the following code:

```
$title = $packtPageXpath->query('//h1');  // Querying for <h1> (title of book)

// If title exists
if ($title->length > 0) {
  $packtBook['title'] = $title->item(0)->nodeValue;  // Add title to array
}
```

7. Similarly, for the release date, we know that the required data is between the `<span class="date-display-single"></span>` tags, so we can build an XPath expression to find this, for example `//span[@class="date-display-single"]`. This is informing the query to return all nodes for any `<span>` elements that have a `class=""` attribute of `date-display-single`. We then check to see if it exists, and if so, it is added to the `$packtBook` array in the `release` key as follows:

```
$release = $packtPageXpath->query('//span[@class="date-display-
single"]');  // Querying for <span class="date-display-single">
(release date)

// If release date exists
if ($release->length > 0) {
  $packtBook['release'] = $release->item(0)->nodeValue;  // Add
release date to array
}
```

8. The book overview is scraped in exactly the preceding way, only this time the XPath query used is, `//div[@class="overview_left"]`, and when it is added to the array in `$packtBook['overview']`, we run it through the PHP's `trim()` function, which is used to strip any excessive whitespace from the beginning and end of the string as follows:

```
$overview = $packtPageXpath->query('//div[@class="overview_
left"]');  // Querying for <div class="overview_left">

// If overview exists
if ($overview->length > 0) {
  $packtBook['overview'] = trim($overview->item(0)->nodeValue);
// Trim whitespace and add overview to array
}
```

9. The author details are scraped similarly to the previous data, though, because there are multiple items, both the XPath expression and the code required to add them to our array are slightly different. The XPath expression used, `//div[@class="bpright"]/div[@class="author"]/a` searches through all nodes for `<div class="bpright">`, then within that for `<div class="author">`, then returns all of the `<a>` elements within `<div class="bpright"> <div class="author"></div></div>`. The existence of authors within the `$author` object is then tested as before, but because this time we have multiple results, we need to add each one to the array individually. To do this we use a `for` loop to iterate over each item, adding it to a second dimension array at `$packtBook['authors']`. The `for` loop works by setting a counter, `$i` to `0`, then each time the loop iterates, the value of $i is tested against the number of author results and if it is less than the number of authors, the loop iterates again each time incrementing the counter by `1` using the increment operator, `$i++`. If the condition fails, that is, `$i` being equal to or greater than the `$author` value's length, PHP breaks out of the loop and continues through the script as explained in the following code:

```
$author = $packtPageXpath->query('//div[@class="bpright"]/div[@
class="author"]/a');  // Querying for all authors
// If authors exist
if ($author->length > 0) {
  // For each author
  for ($i = 0; $i < $author->length; $i++) {
    $packtBook['authors'][] = $author->item($i)->nodeValue;  //
Add author to 2nd dimension of array
  }
}
```

10. Finally, we print the contents of the $packtBook array to the screen using the following statement:

```
print_r($packtBook);
```

There's more...

In addition to the XPath expressions discussed previously, there are many more that can be used to build specific queries. Some important expressions to know are listed in the following table:

Expression	Description
/	Gives the root node
//	Gives all nodes
.	Gives the current node
..	Gives the parent node
@	Gives attributes
[n]	Gives the nth element
[last()]	Gives the last element
[last()-n]	Gives the nth element from last
[position()<n]	Gives the first n elements
[x>n]	Gives all x elements containing an element greater than n

The custom scraping function (Simple)

There are many cases when scraping using XPath is either impractical or simply not possible. In these cases custom functions are useful for scraping our required data from the page.

The custom function, which we will create in this recipe, `scrapeBetween()`, will enable us to scrape the content from between any two known strings in a document.

In this recipe, the `scrapeBetween()` function will be used to scrape the Google Analytics ID of a website from its JavaScript implementation.

How to do it...

1. Enter the following code into a new PHP project:

```
<?php

// Function to make GET request using cURL
function curlGet($url) {
  $ch = curl_init();       // Initialising cURL session
  // Setting cURL options
  curl_setopt($ch, CURLOPT_RETURNTRANSFER, TRUE);
  curl_setopt($ch, CURLOPT_FOLLOWLOCATION, TRUE);
  curl_setopt($ch, CURLOPT_URL, $url);
  $results = curl_exec($ch);      // Executing cURL session
  curl_close($ch);  // Closing cURL session
  return $results;  // Return the results
}

// Function for scraping content between two strings
function scrapeBetween($item, $start, $end) {
  if (($startPos = stripos($item, $start)) === false) {  // If
$start string is not found
    return false;  // Return false
  } else if (($endPos = stripos($item, $end)) === false) {  // If
$end string is not found
    return false;  // Return false
  } else {
    $substrStart = $startPos + strlen($start); // Assigning start
position
    return substr($item, $substrStart, $endPos - $substrStart);
// Returning string between start and end positions
  }
}

$page = curlGet('http://www.packtpub.com');

$analyticsId = scrapeBetween($page, '(["_setAccount", "', '"])');

echo $analyticsId;

?>
```

2. Save the project as `5-custom-scraping-functions.php`.

3. Execute the script.
4. The results of the scrape will be displayed on screen as follows:

```
UA-284627-1
```

How it works...

Let's look at how these steps were performed:

1. First in our scraper is the `curlGet()` function we created in the *Making a simple cURL request* recipe.

   Next we have our scrapeBetween() function which takes the following three parameters:

   - `$item`: This parameter is the content from which we wish to scrape (the target web page downloaded using cURL), as a string
   - `$start`: This parameter is a string which indicates from which place we wish to scrape
   - `$end`: This parameter is a string which indicates until which place we wish to scrape

2. Within our `scrapeBetween()` function, first we use PHP's `stripos()` function to check if both the `$start` and `$end` strings are found in `$item`. If either of them are not, the function ends and returns `false` as shown in the following code:

```
if (($startPos = stripos($item, $start)) === false) {  // If
$start string is not found
    return false;  // Return false
  } else if (($endPos = stripos($item, $end)) === false) {  // If
$end string is not found
    return false;  // Return false
```

3. If both the `$start` and `$end` strings are found, we then assign the position from which we wish to start scraping from, and use the `substr()` function to return the string between the start and end positions as follows:

```
} else {
    $substrStart = $startPos + strlen($start);  // Assigning start
position
    return substr($item, $substrStart, $endPos - $substrStart);
// Returning string between start and end positions
  }
```

4. With our necessary functions defined, we can now do some scraping. We request our target web page using our `curlGet()` function and store the results in the `$page` variable as follows:

```
$page = curlGet('http://www.packtpub.com');
```

5. We now execute our `scrapeBetween()` function, having identified the location of the analytics code by looking at the source of our target web page as between the strings (`["_setAccount", " and "]`), we pass into the function, our target page as the `$page` variable, and our starting string as (`["_setAccount", "` and ending string as `"]`), and then store the result in the `$analyticsId` variable. This is then echoed to the screen using the following code:

```
$analyticsId = scrapeBetween($page, '(["_setAccount", "', '"])');

echo $analyticsId;
```

Scraping and saving images (Simple)

So far we have only covered scraping text content from a target web page, but often there are other contents, for example images, that we may require. In these cases we need to request the file, download it, and verify that it is an image and save it to a local directory for future use.

Using the cURL library and file functions in PHP, in this recipe we will create a function that will be used to download and save images from a target site.

How to do it...

1. Enter the following code into a new PHP project:

```
<?php



// Function to make GET request using cURL
function curlGet($url) {
  $ch = curl_init();  // Initialising cURL session
  // Setting cURL options
  curl_setopt($ch, CURLOPT_RETURNTRANSFER, TRUE);
  curl_setopt($ch, CURLOPT_FOLLOWLOCATION, TRUE);
  curl_setopt($ch, CURLOPT_URL, $url);
  $results = curl_exec($ch);  // Executing cURL session
  curl_close($ch);  // Closing cURL session
  return $results;  // Return the results
}

// Function to return XPath object
function returnXPathObject($item) {
  $xmlPageDom = new DomDocument();  // Instantiating a new
DomDocument object
```

```
  @$xmlPageDom->loadHTML($item);  // Loading the HTML from
downloaded page
  $xmlPageXPath = new DOMXPath($xmlPageDom);  // Instantiating new
XPath DOM object
  return $xmlPageXPath;  // Returning XPath object
}

$packtPage = curlGet('http://www.packtpub.com/news/experience-
amazing-gimp-photo-editing-tools-with-packts-new-ebook');  //
Calling function curlGet() and storing returned results in
$packtPage variable

$packtPageXpath = returnXPathObject($packtPage);  // Instantiating
new XPath DOM object

$coverImage = $packtPageXpath->query('//span/img/@src');  //
Querying for book cover image URL

// If cover image exists
if ($coverImage->length > 0) {

  $imageUrl = $coverImage->item(0)->nodeValue; // Add URL to
variable

  $imageName = end(explode('/', $imageUrl));  // Retrieving image
name from URL

  // If file is an image
  if (getimagesize($imageUrl)) {
    $imageFile = curlGet($imageUrl);  // Download image using cURL
    $file = fopen($imageName, 'w');  // Opening file handle
    fwrite($file, $imageFile);  // Writing image file
    fclose($file);  // Closing file handle
  }
}

?>
```

2. Save the project as `7-scraping-images.php`.
3. Execute the script.
4. The downloaded image will now be stored in the same directory as the script.

How it works...

Let's look at how these steps were performed:

1. Firstly, we have included the `curlGet()` function that we created in the *Making a simple cURL request* recipe, giving us the functionality to request a target page.
2. Next, we included the `returnXPathObject()` function that we created in the *Scraping elements using XPath* recipe, enabling us to return an XPath object of our target XML to scrape.
3. With our required functions now in place, we can go ahead and scrape an image from our target page, in this case the cover of a book. As we have done in previous recipes, we request a target page, return an XPath object of that page, and from there we scrape the URL of the image we wish to save by using the following code:

```
$packtPage = curlGet('http://www.packtpub.com/news/experience-
amazing-gimp-photo-editing-tools-with-packts-new-ebook');  //
Calling function curlGet() and storing returned results in
$packtPage variable

$packtPageXpath = returnXPathObject($packtPage);  // Instantiating
new XPath DOM object

$coverImage = $packtPageXpath->query('//span/img/@src');  //
Querying for book cover image URL
```

4. With the URL of our required image we can extract the name of the image by using the PHPs `explode()` function to separate the parts of the URL into an array, and then use the `end()` function to return only the last element, which will be the name of the image as given in the following code:

```
// If cover image exists
if ($coverImage->length > 0) {

  $imageUrl = $coverImage->item(0)->nodeValue; // Add URL to
variable
```

5. $imageName = end(explode('/', $imageUrl)); // Retrieving image name from URLWe now validate that the file we wish to download is in fact an image using the getimagesize() function.
6. If it returns true, we request the image file using the `curlGet()` function as follows:

```
if (getimagesize($imageUrl)) {
  $imageFile = curlGet($imageUrl);      // Download image using
cURL
Finally we save the image. We open a new writable file handle
using our $imageName, write the image file to disk, then close the
file handle.
  fwrite($file, $imageFile);  // Writing image file
  fclose($file);  // Closing file handle
```

There's more...

While the scraper we have created in this recipe has to download an image by changing the validation, this can even be used to download files of any type from a target website.

Submitting a form using cURL (Intermediate)

Many times while web scraping, the data which we require is located behind a form. Whether that be a login form to a members area, a search form, a file upload, or any other form submission, it is frequently implemented using a `POST` request. The process of submitting a form with a `POST` request can be easily automated using PHP and cURL.

There are a number of steps required to successfully submit a `POST` form, such as capturing and analyzing HTTP headers, submitting the form, and in case of a login form, using cookies to store session data.

How to do it...

1. Enter the following code into a new PHP project:

```
<?php

// Function to submit form using cURL POST method
function curlPost($postUrl, $postFields, $successString) {

  $useragent = 'Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10.5;
en-US; rv:1.9.2.3) Gecko/20100401 Firefox/3.6.3';  // Setting user
agent of a popular browser

  $cookie = 'cookie.txt';  // Setting a cookie file to store
cookie

  $ch = curl_init();  // Initialising cURL session

  // Setting cURL options
  curl_setopt($ch, CURLOPT_SSL_VERIFYPEER, FALSE);  // Prevent
cURL from verifying SSL certificate
  curl_setopt($ch, CURLOPT_FAILONERROR, TRUE);  // Script should
fail silently on error
  curl_setopt($ch, CURLOPT_COOKIESESSION, TRUE);  // Use cookies
  curl_setopt($ch, CURLOPT_FOLLOWLOCATION, TRUE);  // Follow
Location: headers
  curl_setopt($ch, CURLOPT_RETURNTRANSFER, TRUE);  // Returning
transfer as a string
  curl_setopt($ch, CURLOPT_COOKIEFILE, $cookie);  // Setting
```

```
cookiefile
  curl_setopt($ch, CURLOPT_COOKIEJAR, $cookie);  // Setting
cookiejar
  curl_setopt($ch, CURLOPT_USERAGENT, $useragent);  // Setting
useragent
  curl_setopt($ch, CURLOPT_URL, $postUrl);  // Setting URL to POST
to

  curl_setopt($ch, CURLOPT_POST, TRUE);  // Setting method as POST
  curl_setopt($ch, CURLOPT_POSTFIELDS, http_build_
query($postFields));  // Setting POST fields as array

  $results = curl_exec($ch);  // Executing cURL session
  curl_close($ch);  // Closing cURL session

  // Checking if login was successful by checking existence of
string
  if (strpos($results, $successString)) {
    return $results;
  } else {
    return FALSE;
  }
}

$userEmail = 'name@email.com';  // Setting your email address for
site login
$userPass = 'password123';  // Setting your password for site
login

$postUrl = 'https://www.packtpub.com/account';  // Setting URL to
POST to

// Setting form input fields as 'name' => 'value'
$postFields = array(
  'email' => $userEmail,
  'password' => $userPass,
  'destination' => 'account',
  'form_id' => 'packt_login_form'
);

$successString = 'You are logged in as';

$loggedIn = curlPost($postUrl, $postFields, $successString);  //
Executing curlPost login and storing results page in $loggedIn

?>
```

2. Save the project as `8-submitting-form.php`.
3. Execute the script.
4. We are now logged in to the website.

How it works...

If you have followed the recipes in this book, specifically the *Making a simple cURL request* recipe, then the first part of this script should look familiar. We create the `curlPost()` function, which is used to make a cURL request, this time rather than making a `GET` request as we did previously, we're making a `POST` request. The parameters passed to the function is the URL to the form fields to POST, and a success string which will be used to check whether the form is submitted correctly by following the given steps:

1. Inside the `curlPost()` function, we first set a user agent string. This is used to identify our client to the server. We can enter any string here to use as our user agent, which would usually be the name of our crawler. In this case, we have chosen to emulate a popular browser, though this can sometimes be seen as deceitful when crawling parts of the Web, which we do not own as explained in the following code:

```
$useragent = 'Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10.5;
en-US; rv:1.9.2.3) Gecko/20100401 Firefox/3.6.3';  // Setting
useragent of a popular browser
```

2. Next, as we are logging into a website which uses cookies for authentication, we need to set a file to store our cookies in by using the following code:

```
$cookie = 'cookie.txt';    // Setting a cookie file to store cookie
```

3. As discussed in previous recipes, we first initialize a cURL session, set the cURL options, execute, then close the cURL session. We have introduced some new cURL settings here as follows:
    - `CURLOPT_COOKIESESSION`: This setting instructs cURL to start a new cookie session
    - `CURLOPT_COOKIEFILE`: This setting tells cURL, our cookie file
    - `CURLOPT_COOKIEJAR`: This setting tells cURL, our cookie jar
    - `CURLOPT_USERAGENT`: This setting tells cURL, our user agent
    - `CURLOPT_POST`: This setting tells cURL that we wish to make a `POST` request
    - `CURLOPT_POSTFIELDS`: This setting receives an array of the fields, we wish to `POST` as `'name' => 'value'`

```
$ch = curl_init();  // Initialising cURL session

// Setting cURL options
curl_setopt($ch, CURLOPT_SSL_VERIFYPEER, FALSE);  // Prevent cURL
```

```
from verifying SSL certificate
curl_setopt($ch, CURLOPT_FAILONERROR, TRUE);  // Script should
fail silently on error
curl_setopt($ch, CURLOPT_COOKIESESSION, TRUE);  // Use cookies
curl_setopt($ch, CURLOPT_FOLLOWLOCATION, TRUE);  // Follow
Location: headers
curl_setopt($ch, CURLOPT_RETURNTRANSFER, TRUE);  // Returning
transfer as a string
curl_setopt($ch, CURLOPT_COOKIEFILE, $cookie);  // Setting
cookiefile
curl_setopt($ch, CURLOPT_COOKIEJAR, $cookie);  // Setting
cookiejar
curl_setopt($ch, CURLOPT_USERAGENT, $useragent);  // Setting
useragent
curl_setopt($ch, CURLOPT_URL, $postUrl);  // Setting URL to POST
to

curl_setopt($ch, CURLOPT_POST, TRUE);  // Setting method as POST
curl_setopt($ch, CURLOPT_POSTFIELDS, http_build_
query($postFields));  // Setting POST fields as array

$results = curl_exec($ch);  // Executing cURL session
curl_close($ch);  // Closing cURL session
```

4. Finally, the function with which we check if the login was successful. To do this, we first need to identify a string on the page that is displayed upon successful login. The existence of this is then checked using `strpos()`, and if it is found, then login has been successful and the page is returned; if it is not found then the function returns `FALSE` as given in the following code:

```
if (strpos($results, $successString)) {
    return $results;
} else {
    return FALSE;
}
```

With our function coded, we are now ready to submit the form.

5. We first assign our username and password to variables as follows:

```
$userEmail = 'name@email.com';  // Setting your email address for
site login
$userPass = 'password123';  // Setting your password for site
login
Next we build an array of the fields we are required to submit.
// Setting form input fields as 'name' => 'value'
$postFields = array(
  'email' => $userEmail,
```

```
    'password' => $userPass,
    'destination' => 'account',
    'form_id' => 'packt_login_form'
);
```

6. We now set the URL to receive the `POST` request as given in the following code:

```
$postUrl = 'https://www.packtpub.com/account'; // Setting URL to
POST to
```

7. A success message to be displayed for a successful login is now set as follows:

```
$successString = 'You are logged in as';       // Setting success
string to test
```

8. Finally, we execute the `curlPost()` function, passing `$postUrl`, `$postFields`, and `$successString` as parameters, as given in the following code:

```
$loggedIn = curlPost($postUrl, $postFields, $successString);  //
Executing curlPost login and storing results page in $loggedIn
```

There's more...

The recipe has made the assumptions that we know the URL we need to POST to and we know the form fields we need to post. Now, we will cover how to identify these and how to submit forms with additional inputs, for example file uploads.

Identifying the POST URL

The URL which we need to POST to is often not the same URL as the web page on which the form is located. To identify the URL to use in our script, we need to look at the form being submitted and find the `action=""` attribute. It's value is the URL we require. In this example the action is `/account`, this being located in the root directory of the domain (`https://www.packtpub.com`), we can build the full URL as `https://www.packtpub.com/account` using the following code:

```
<form action="/account" accept-charset="UTF-8" method="post"
id="packt-login-form">
```

Identifying the fields to POST

The fields that we require or are available to `POST` when submitting a form are not always obvious. For example, there may be fields which are not visible when looking at the form in a browser, such as `hidden` fields.

One way to identify the fields that are required or available is to view the source code of the page and find all of the `<input>` tags for the `name` and `value` attributes.

Posting upload forms

Some web forms, in addition to the preceding inputs, allow the upload of files for submission. For the following forms, uploading of a file is simple:

- We need to identify the full path to the local file we wish to upload:

  ```
  /Users/jacobward/Documents/upload_file.png
  ```

- This string is then added to the `POST` fields array as before, only this is prepended by an `@` as given in the following code:

  ```
  'file' => '@/Users/jacobward/Documents/upload_file.png'
  ```

Traversing multiple pages (Intermediate)

Up to this point, everything we have worked with has been scraping content from a single page, but where scraping really provides the most valuable results is in crawling and scraping multiple pages.

In the following recipe we will access the paginated result of a search query, visit each page of search results, and scrape all of the results.

How to do it...

1. Enter the following code into a new PHP project:

```
<?php

// Function for making a GET request using cURL
function curlGet($url) {
  $ch = curl_init();  // Initialising cURL session
  // Setting cURL options
  curl_setopt($ch, CURLOPT_RETURNTRANSFER, TRUE);  // Returning
transfer as a string
  curl_setopt($ch, CURLOPT_FOLLOWLOCATION, TRUE);  // Follow
location
  curl_setopt($ch, CURLOPT_URL, $url);  // Setting URL
  $results = curl_exec($ch);  // Executing cURL session
n the results
}

// Funtion to return XPath object
function returnXPathObject($item) {
  $xmlPageDom = new DomDocument();  // Instantiating a new
DomDocument object
  @$xmlPageDom->loadHTML($item);  // Loading the HTML from
```

```
downloaded page
  $xmlPageXPath = new DOMXPath($xmlPageDom);  // Instantiating new
XPath DOM object
  return $xmlPageXPath;  // Returning XPath object
}

// Function for scraping content between two strings
function scrapeBetween($item, $start, $end) {
  if (($startPos = stripos($item, $start)) === false) {  // If
$start string is not found
    return false;  // Return false
  } else if (($endPos = stripos($item, $end)) === false) {  // If
$end string is not found
    return false;  // Return false
  } else {
    $substrStart = $startPos + strlen($start);  // Assigning start
position
    return substr($item, $substrStart, $endPos - $substrStart);
// Returning string between start and end positions
  }
}

// Declaring arrays
$resultsPages = array();
$bookPages = array();

$initialResultsPageUrl = 'http://www.packtpub.com/books?keys=php';
// Assigning initial results page URL to work from
$resultsPages[] = $initialResultsPageUrl;  // Adding initial
results page URL to $resultsPages array

$initialResultsPageSrc = curlGet($initialResultsPageUrl);  //
Requesting initial results page

$resultsPageXPath = returnXPathObject($initialResultsPageSrc);  //
Instantiating new XPath DOM object

$resultsPageUrls = $resultsPageXPath->query('//ul[@class="pager"]/
li/a/@href');  // Querying for href attributes of pagination

// If results exist
if ($resultsPageUrls->length > 0) {
  // For each results page URL
  for ($i = 0; $i < $resultsPageUrls->length; $i++) {
    $resultsPages[] = 'http://www.packtpub.com' .
```

```
$resultsPageUrls->item($i)->nodeValue;  // Build results page URL
and add to $resultsPages array
  }
}

$uniqueResultsPages = array_values(array_unique($resultsPages));
// Removing duplicates from array and reindexing

// For each unique results page URL
foreach ($uniqueResultsPages as $resultsPage) {

  $resultsPageSrc = curlGet($resultsPage);  // Requesting results
page

  $booksPageXPath = returnXPathObject($resultsPageSrc);  //
Instantiating new XPath DOM object

  $bookPageUrls = $booksPageXPath->query('//div[@class="view-
content"]/table/tbody/tr/td/div/div[@class="field-content"]/a/@
href');  // Querying for href attributes of books

  // If book page URLs exist
  if ($bookPageUrls->length > 0) {
    // For each book page URL
    for ($i = 0; $i < $bookPageUrls->length; $i++) {
      $bookPages[] = 'http://www.packtpub.com' . $bookPageUrls-
>item($i)->nodeValue;  // Add URL to $bookPages array
    }
  }

  $booksPageXPath = NULL;  // Nulling $booksPageXPath object
  $bookPageUrls = NULL;  // Nulling $bookPageURLs object

  sleep(rand(1, 3));  // Being polite and sleeping

}

$uniqueBookPages = array_values(array_unique($bookPages));  //
Removing duplicates from array and reindexing

print_r($uniqueBookPages);

?>
```

2. Save the project as `9-traversing-multiple-pages.php`.

3. Execute the script.
4. The results of the scrape will be displayed on screen as follows:

```
Array
(
    [0] => http://www.packtpub.com/phplist-2-e-mail-campaign-
manager/book
    [1] => http://www.packtpub.com/mastering-phpmyadmin-3-4-for-
effective-mysql-management/book
    [2] => http://www.packtpub.com/yii-1-1-using-php-framework-
application-development-cookbook/book
    [3] => http://www.packtpub.com/couchdb-and-php-web-
development-beginners-guide/book
    [4] => http://www.packtpub.com/learning-ext-js/book
    [5] => http://www.packtpub.com/rapidweaver-5-beginners-guide/
book
… and so on …
```

How it works...

1. First in our script, we've included our previously coded functions, `curlGet()`, `returnXPathObject()`, and `scrapeBetween()`.
2. We then assign the URL of the first page of results to the `$initialResultsPageUrl` variable and to the `$resultsPages` array, which we will be using to store all of the results page URLs by using the following code:

```
$initialResultsPageUrl = 'http://www.packtpub.com/books?keys=php';
// Assigning initial results page URL to work from

$resultsPages[] = $initialResultsPageUrl;  // Adding initial
results page URL to $resultsPages array
```

3. We use cURL to request the initial results page and return an XPath DOM object of it, as given in the following code:

```
$initialResultsPageSrc = curlGet($initialResultsPageUrl);  //
Requesting initial results page

$resultsPageXPath = returnXPathObject($initialResultsPageSrc);  //
Instantiating new XPath DOM object
```

4. This page contains links to all of the results page URLs (2-6), which we will need to scrape in two places, before and after the results. We build an XPath query to access these and return them into the `$resultsPageUrls` object by using the following code:

```
$resultsPageUrls = $resultsPageXPath->query('//ul[@class="pager"]/
li/a/@href');// Querying for href attributes of pagination
```

5. We now add all of these URLs to the `$resultsPages` array and make sure they are all unique values, as given in the following code:

```
if ($resultsPageUrls->length > 0) {
  // For each results page URL
  for ($i = 0; $i < $resultsPageUrls->length; $i++) {
    $resultsPages[] = 'http://www.packtpub.com' .
$resultsPageUrls->item($i)->nodeValue;  // Build results page URL
and add to $resultsPages array
  }
}
$uniqueResultsPages = array_values(array_unique($resultsPages));
// Removing duplicates from array and reindexing
```

6. With our array of URLs, which we need to traverse, we are ready to begin. We use a `for each` loop to iterate over each of the URLs in the array as given in the following code:

```
foreach ($uniqueResultsPages as $resultsPage) {
```

7. We use cURL to request the URL and return an `XPath DOM` object.

```
$resultsPageSrc = curlGet($resultsPage);       // Requesting
results page
$booksPageXPath = returnXPathObject($resultsPageSrc); //
Instantiating new XPath DOM object
```

8. We then build an XPath query to return the URLs of all of the books listed on the page and add the URLs to the `$bookPages` array as follows:

```
$bookPageUrls = $booksPageXPath->query('//div[@class="view-
content"]/table/tbody/tr/td/div/div[@class="field-content"]/a/@
href');       // Querying for href attributes of books

if ($bookPageUrls->length > 0) {
  // For each book page URL
  for ($i = 0; $i < $bookPageUrls->length; $i++) {
    $bookPages[] = 'http://www.packtpub.com' . $bookPageUrls-
>item($i)->nodeValue;  // Add URL to $bookPages array
  }
}
```

9. The objects used are then nulled to ensure they are free to use again in the next iteration as follows:

```
$booksPageXPath = NULL;  // Nulling $booksPageXPath object
$bookPageUrls = NULL;  // Nulling $bookPageURLs object
```

10. Before the next iteration of the `for each` loop, we tell the script to sleep for a random period of time between 1 and 3 seconds. This is both to be polite to the server and to ensure that we do not get blocked. Many requests in quick succession to the server can be misinterpreted as a **Denial of service** (**DOS**) attack:

```
sleep(rand(1, 3));  // Being polite and sleeping
```

11. With our iteration completed, we can now clean the array to ensure all of the URLs are unique, as given in the following code:

```
$uniqueBookPage = array_values(array_unique($bookPages));  //
Removing duplicates from array and reindexing
```

12. Finally, we print the array of scraped URLs to the screen by using the following statement:

```
print_r($uniqueBookPages);
```

There's more...

The output of this scraper is an array of URLs for all of the books returned from the search. We can use the same iterative method that we did for visiting each of the results pages to visit each of the book pages in the array, and then continue scraping from there.

Saving scraped data to a database (Intermediate)

In all of the recipes so far, the results of our scrapes are simply been printed to the screen. While this is fine for small projects, where the data may only be required only one time; but if we are scraping a large amount of data, which needs to be organized and saved for future access, we will need to create a database.

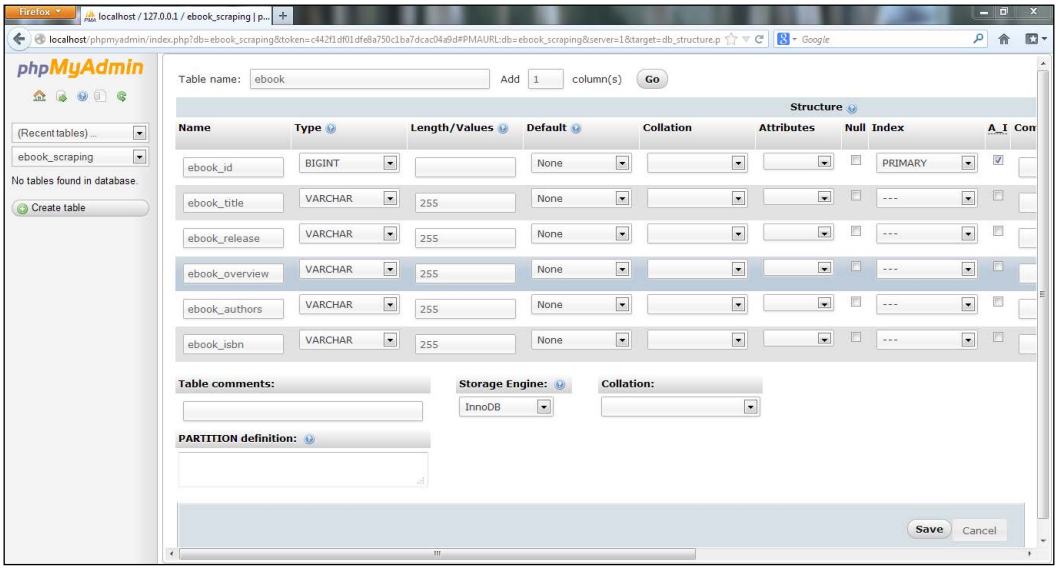
Getting ready

In this recipe, we will create a database using MySQL, into which we can save our scraped data. We will then modify the previous recipe, which prints the scraped data to the screen, to access the database, and save the data.

How to do it...

1. Start the XAMPP Control Panel.
2. Click on the **Admin** button for MySQL, and phpMyAdmin will open in our web browser.
3. Select the **Users** tab and click on **Add user**.
4. Enter the user name as `ebook_scraping` and password as `scr4p1ng`. You may of course use your own username and password, and then substitute that later in the code.

5. Select the radio button to create database with the same name and grant all privileges.
6. Click on **Add user** to create the new user and database.
7. Select our newly created database, **ebook_scraping**, in the left panel.
8. In the **Create table** dialog, enter the name as `ebook` and the number of columns as `6`, and then click on **Go**.
9. Enter the details as shown in the following screenshot, and then click on **Save**:



10. Open our previously saved scraper from the *Multithreaded scraping using multi-cURL* recipe available at `http://www.packtpub.com/sites/default/files/downloads/4760OS_Bonus_recipes.pdf` in Eclipse.
11. Remove the following line:

```
print_r($packtEbooks);
```

12. Append the following code to the script:

```
<?php

$dbUser = 'ebook_scraping';  // Database username
$dbPass = 'scr4p1ng';  // Database password
$dbHost = 'localhost'; // Database host
$dbName = 'ebook_scraping';  // Database name

$tableName = 'ebook';      // Table name to store ebooks

// Try to create a new database connection
try {
```

```
  $cxn = new PDO('mysql:host=' . $dbHost . ';dbname=' . $dbName,
$dbUser, $dbPass);
  $cxn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
// Changing default error mode from PDO::ERRMODE_SILENT to
PDO::ERRMODE_EXCEPTION
} catch(PDOException $e) {
  echo 'Error: ' . $e->getMessage();  // Show exception error
}

$insertEbook = $cxn->prepare("INSERT INTO $tableName (ebook_
isbn, ebook_title, ebook_release, ebook_overview, ebook_authors)
VALUES (:ebookIsbn, :ebookTitle, :ebookRelease, :ebookOverview,
:ebookAuthors)");  // Preparing INSERT query

// For each ebook in array, add to database
foreach ($packtEbooks as $ebookIsbn => $ebookDetails) {
  // Executing INSERT query
  $insertEbook->execute(
    array(
      ':ebookIsbn' => $ebookIsbn,
      ':ebookTitle' => $ebookDetails['title'],
      ':ebookRelease' => $ebookDetails['release'],
      ':ebookOverview' => $ebookDetails['overview'],
      ':ebookAuthors' => implode(', ', $ebookDetails['authors'])
    )
  );
}

$selectEbooks = $cxn->prepare("SELECT * FROM $tableName");  //
Preparing SELECT query

$selectEbooks->execute();  // Executing SELECT query

echo '<table><tr><th>ISBN</th><th>Title</th><th>Overview</
th><th>Author(s)</th><th>Release Date</th></tr>';  // Opening
table and headers

// While there are rows returned, echo table data
while ($row = $selectEbooks->fetch()) {
  echo '<tr>';

  echo '<td>' . $row['ebook_isbn'] . '</td>';
  echo '<td>' . $row['ebook_title'] . '</td>';
  echo '<td>' . $row['ebook_overview'] . '</td>';
  echo '<td>' . $row['ebook_authors'] . '</td>';
  echo '<td>' . $row['ebook_release'] . '</td>';
```

```
  echo '</tr>';
}

echo '</table>';  // Closing table

?>
```

13. Save the project as `12-saving-database.php`.
14. Execute the script.
15. The data will be scraped as before, added to the database, and then retrieved and displayed on screen in a table.

How it works...

The first part of this recipe is the same as multithreaded scraping using multi-cURL, after this we introduce some new code to perform the database operations, which is where we'll pick it up here.

1. The first five lines we assign our database settings to variables for using in the following script:

```
$dbUser = 'scraping_db';  // Database username
$dbPass = 'scr4p1ng';  // Database password
$dbHost = 'localhost'; // Database host
$dbName = 'ebook_scraping';  // Database name
$tableName = 'ebook';  // Table name to store ebooks
```

   We then use PDO to create a new database connection passing our stored database variables, change the *Multithreaded scraping using multi-cURL* recipe available at `http://www.packtpub.com/sites/default/files/downloads/4760OS_Bonus_recipes.pdf` default error mode and show an error should an exception be thrown.

```
try {
  $cxn = new PDO('mysql:host=' . $dbHost . ';dbname=' . $dbName,
$dbUser, $dbPass);
  $cxn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
// Changing default error mode from PDO::ERRMODE_SILENT to
PDO::ERRMODE_EXCEPTION
} catch(PDOException $e) {
  echo 'Error: ' . $e->getMessage();  // Show exception error
}
```

2. We now prepare the SQL query to insert the scraped data into our database. The first part of the query tells MySQL that we wish to `INSERT` data into the `$tableName` table. This is then followed by a comma-separated list of the table's column names, followed by `VALUES` and a comma-separated list of the data we which to insert into those columns, respectively. Please note the use of double quotes (`""`), which are used to encapsulate this string, this enables us to use variables directly in the quotes, rather than having to concatenate with the concatenation operator (`.`) as follows:

```
$insertEbook = $cxn->prepare("INSERT INTO $tableName (ebook_
isbn, ebook_title, ebook_release, ebook_overview, ebook_authors)
VALUES (:ebookIsbn, :ebookTitle, :ebookRelease, :ebookOverview,
:ebookAuthors)");  // Preparing INSERT query
```

3. With the query ready, we can now execute it and insert the data into our database. To do this we use a `foreach` loop to iterate over the array of scraped data, each time executing the query and passing an array of the data to insert as follows:

```
// For each ebook in array, add to database
foreach ($packtEbooks as $ebookIsbn => $ebookDetails) {
  // Executing INSERT query
  $insertEbook->execute(
    array(
      ':ebookIsbn' => $ebookIsbn,
      ':ebookTitle' => $ebookDetails['title'],
      ':ebookRelease' => $ebookDetails['release'],
      ':ebookOverview' => $ebookDetails['overview'],
      ':ebookAuthors' => implode(', ', $ebookDetails['authors'])
    )
  );
}
```

4. The data is now stored in our database and we can go ahead and access this data to work with, in this case printing to the screen in a nicely formatted table. The first step is to prepare the query to select the data from the database. `SELECT` tells MySQL that we wish to select data. `*` tells MySQL that we wish to select all the available column data from `$tableName` as follows:

```
$selectEbooks = $cxn->prepare("SELECT * FROM $tableName");  //
Preparing SELECT query
```

5. We then execute the preceding query as follows:

```
$selectEbooks->execute();  // Executing SELECT query
```

6. We now open our HTML table and echo out the headers as follows:

```
echo '<table><tr><th>ISBN</th><th>Title</th><th>Overview</
th><th>Author(s)</th><th>Release Date</th></tr>';
```

7. We iterate over the returned results from our `SELECT` query for each of the row fetching an array with the array keys as the column titles, using the following code:

```
while ($row = $selectEbooks->fetch()) {
```

8. We create a new HTML table, rows and cells for each column, and echoing the returned results into each using the following code:

```
echo '<tr>';
echo '<td>' . $row['ebook_isbn'] . '</td>';
```

```
echo '<td>' . $row['ebook_title'] . '</td>';
echo '<td>' . $row['ebook_overview'] . '</td>';
echo '<td>' . $row['ebook_authors'] . '</td>';
echo '<td>' . $row['ebook_release'] . '</td>';
echo '</tr>';
```

9. Finally we close the table using the following statement:

```
echo '</table>';
```

Scheduling scrapes (Simple)

Using all of the recipes we have worked through so far, we can perform a number of useful scraping tasks. The one thing holding us back, given that our goal should be complete automation, is the need to manually execute our scrapers when we need them to be executed repeatedly.

In this recipe we will cover the process of scheduling the execution of our scrapers using tools already available in our operating system. This will make our scraping processes truly automated.

Getting ready

Identify the scraper script which needs to be scheduled. In this recipe we will work with a scraper from the *Retrieving and extracting content from e-mails* recipe available at `http://www.packtpub.com/sites/default/files/downloads/4760OS_Bonus_recipes.pdf`, `retrieving-email.php`, and have it scheduled to execute every day at 6:00 p.m.

How to do it...

1. Open the Task Scheduler by navigating to the path, **Start** | **Control Panel** | **System and Security** | **Administrative Tools** | **Task Scheduler**.
2. In the **Task Scheduler** menu bar navigate to **Action | Create Basic Task...**.
3. In the **Name** field, type `Scheduled Email Scraper`, and then click on **Next**.
4. Select **Daily**, and then click on **Next**.
5. Set the time to `18:00:00` to recur every 1 day, and then click on **Next**.
6. Select **Start a program**, and then click on **Next**.
7. In the **Program/script** field enter `php C:\xampp\htdocs\Web Scraping\10-retrieving-email.php`,and then click on **Next**.
8. In the dialog box, click on **Yes**.

9. Click on **Finish**.
10. The script will be executed everyday at 18:00.

Building a reusable scraping class (Advanced)

Given the recipes we have completed so far, we are in a position where we can take on most scraping tasks that we may come across by developing a solution specific to any problem we may encounter. While this is a perfectly suitable approach, and is necessary for very large projects; for the types of small projects we will usually find ourselves feeling that it is overly time-consuming, and will see us often repeating ourselves.

In this recipe we will introduce some basic **Object oriented programming** (**OOP**) techniques to build a scraping class, which can be easily expanded and reused for any projects we may embark on in the future.

How to do it...

1. Enter the following code into a new PHP file:

```
<?php

class Scrape {

  // Declaring class variables and arrays
  public $url;
  public $source;
  public $baseUrl;
  private $baseUrl;
  private $parsedUrl = array();

  // Construct method called on instantiation of object
  function __construct($url) {
    $this->url = $url;     // Setting URL attribute
    $this->source = $this->curlGet($this->url);
    $this->xPathObj = $this->returnXpathObject($this->source);
    $this->parsedUrl = parse_url($this->url);
    $this->baseUrl = $this->parsedUrl['scheme'] . '://' . $this-
>parsedUrl['host'];
  }

  // Method for making a GET request using cURL
  public function curlGet($url) {
    $ch = curl_init();     // Initialising cURL session
    // Setting cURL options
```

```
    curl_setopt($ch, CURLOPT_RETURNTRANSFER, TRUE);  // Returning
transfer as a string
    curl_setopt($ch, CURLOPT_URL, $url);  // Setting URL
    $results = curl_exec($ch);  // Executing cURL session
    curl_close($ch);  // Closing cURL session
    return $results;  // Return the results
  }

  // Method to return XPath object
  public function returnXPathObject($item) {
    $xmlPageDom = new DomDocument();  // Instantiating a new
DomDocument object
    @$xmlPageDom->loadHTML($item);  // Loading the HTML from
downloaded page
    $xmlPageXPath = new DOMXPath($xmlPageDom);  // Instantiating
new XPath DOM object
    return $xmlPageXPath;  // Returning XPath object
  }


}
?>
```

2. Save the file as scrape.class.php
3. Enter the following code into a new PHP file:

```
<?php

require_once('scrape.class.php');

$cakePhpBook = new Scrape('http://www.packtpub.com/cakephp-
application-development/book');  // Instantiating new instance of
Scrape class

$cakePhpBook->title = $cakePhpBook->xPathObj->query('//h1')-
>item(0)->nodeValue;  // Assigning book title

$cakePhpBook->release = $cakePhpBook->xPathObj->query('//span[@
class="date-display-single"]')->item(0)->nodeValue;  // Assigning
book release date

$cakePhpBook->overview = $cakePhpBook->xPathObj->query('//div[@
class="overview_left"]')->item(0)->nodeValue;  // Assigning book
overview

$cakePhpBook->author = $cakePhpBook->xPathObj->query('//div[@
class="bpright"]/div[@class="author"]/a')->item(0)->nodeValue;  //
Assigning book author
```

```
$cakePhpBook->coverUrl = $cakePhpBook->baseUrl . $cakePhpBook-
>xPathObj->query('//div[@class="cover-images"]/div/div/a/@href')-
>item(0)->nodeValue;       // Assigning cover image URL

$cakePhpBook->eBookPrice = $cakePhpBook->xPathObj->query('//div[@
class="price-choice "]/div/div/span[@class="larger"]')->item(0)-
>nodeValue;  // Assigning eBook price

$cakePhpBook->bundlePrice = $cakePhpBook->xPathObj->query('//div[@
class="price-choice bundle"]/div/div/span[@class="larger"]')-
>item(0)->nodeValue;  // Assigning eBook price

// Echoing out attributes
echo $cakePhpBook->title . '<br />';
echo $cakePhpBook->url . '<br />';
echo $cakePhpBook->release . '<br />';
echo $cakePhpBook->overview . '<br />';
echo $cakePhpBook->author . '<br />';
echo $cakePhpBook->coverUrl . '<br />';
echo $cakePhpBook->eBookPrice . '<br />';
echo $cakePhpBook->bundlePrice . '<br />';

?>
```

4. Save the file as `14-oop-scraping-class.php` and execute it.
5. The results will be displayed on screen as follows:

```
CakePHP Application Development

http://www.packtpub.com/cakephp-application-development/book

July 2008

Develop cutting-edge Web 2.0 applications, and write PHP code
in a faster, more productive way Walk through the creation of a
complete CakePHP Web application Customize the look and feel of
applications using CakePHP layouts and views Make interactive
applications using CakePHP, JavaScript, and AJAX helpers Ready for
the forthcoming release of CakePHP 1.2

Ahsanul Bari

http://www.packtpub.com/sites/default/files/bookimages/1847193897.
jpg

£8.99

£24.99
```

How it works...

Before we go into the specifics of how our script works, we will need to cover some basic OOP principles.

OOP is a style of programming, which is used to group similar tasks and data into classes. From classes, any number of instances of a class may be instantiated as a unique object, each with its own attributes (data unique to a specific object) and methods (functions to perform actions on that data). This can be a difficult concept to understand, having only worked with procedural programming. One easy way to understand the concept is to think in real world terms. For example, we could have a human class, which would define a human being, with attributes such as name, age, hair color, eye color, language spoken, and methods such as speaking and walking. From this human class, we could instantiate a number of instances of this class into a set of objects, each representing a unique person, with their own unique attributes. For example, each of the following would represent a unique person instantiated from a single human class as follows:

```
Name: Bob
Age: 48
Hair: Blonde
Eyes: Blue
Language: English

Name: Sam
Age: 34
Hair: Brown
Eyes: Green
Language: French

Name: Margaret
Age: 23
Hair: Black
Eyes: Brown
Language: Welsh
```

With this in mind, we can move on to understanding our script.

The following are the steps to be performed for building a reusable scraping class:

1. First, we define our class, which we have called `Scrape` as given in the following code:
   ```
   class Scrape {
   ```
2. Inside our class we have added two methods, `curlGet()` and `returnXPathObject()`, which you should recognize as functions we created in previous recipes so we won't cover them in detail. These are the first two methods of our class.

3. The final method of our class is the `__construct()` method. This is implicitly executed each time we instantiate a new object from the class. As such, it is used to prepare our object in the state we wish it to be in, when we begin.
4. The first thing this method does is accepts a URL as a parameter and assigns this URL to an attribute. An attribute can be thought of as a class variable. It is accessed by calling the identifier followed by `->`, and then the name of the attribute. For accessing attributes from within an instantiated class in PHP we can use `$this` as follows:

```
function __construct($url) {
    $this->url = $url;
```

5. Next, we call the `curlGet()` method to retrieve the source code of the page and assign it to the `source` attribute. Notice how methods are also accessed by the class' identifier, followed by the method and its parameters as follows:

```
$this->source = $this->curlGet($this->url);
```

6. We then call the `returnXpathObject()` method, passing to it the source code as follows:

```
$this->xPathObj = $this->returnXpathObject($this->source);
```

7. Finally in our `__construct()` method we use the URL we passed to it to create the base URL, parsing our URL using the `parse_url` function, which returns an array of the different sections from which we take what we need and concatenate into the base URL as follows:

```
$this->parsedUrl = parse_url($this->url);
    $this->baseUrl = $this->parsedUrl['scheme'] . '://' . $this-
>parsedUrl['host'];
```

8. This is then saved as `scrape.class.php` to be included in our next script. This file could be named whatever we like, though naming it as the name of the class it contains (`Scrape`) followed by `class.php` makes it easy to identify as a class file and the class it contains in the future.
9. In our next script, the first thing we do is include our class file using the `require_once()` function. This ensures that our script will only run if it includes our class file as given in the following code:

```
require_once('scrape.class.php');
```

10. Next, we instantiate a new instance of the Scrape class as an object, `$cakePhpBook`, and passing as a parameter the URL of the book we will be working with (CakePHP Application Development), as given in the following code:

```
$cakePhpBook = new Scrape('http://www.packtpub.com/cakephp-
application-development/book');  // Instantiating new instance of
Scrape class
```

11. Over the next lines we assign to our object's attributes the data we scrape from the web page.

```
$cakePhpBook = new Scrape('http://www.packtpub.com/cakephp-
application-development/book');  // Instantiating new instance of
Scrape class

$cakePhpBook->title = $cakePhpBook->xPathObj->query('//h1')-
>item(0)->nodeValue;  // Assigning book title

$cakePhpBook->release = $cakePhpBook->xPathObj->query('//span[@
class="date-display-single"]')->item(0)->nodeValue;  // Assigning
book release date

$cakePhpBook->overview = $cakePhpBook->xPathObj->query('//div[@
class="overview_left"]')->item(0)->nodeValue;  // Assigning book
overview

$cakePhpBook->author = $cakePhpBook->xPathObj->query('//div[@
class="bpright"]/div[@class="author"]/a')->item(0)->nodeValue;  //
Assigning book author

$cakePhpBook->coverUrl = $cakePhpBook->baseUrl . $cakePhpBook-
>xPathObj->query('//div[@class="cover-images"]/div/div/a/@href')-
>item(0)->nodeValue;  // Assigning cover image URL

$cakePhpBook->eBookPrice = $cakePhpBook->xPathObj->query('//div[@
class="price-choice "]/div/div/span[@class="larger"]')->item(0)-
>nodeValue;  // Assigning eBook price

$cakePhpBook->bundlePrice = $cakePhpBook->xPathObj->query('//div[@
class="price-choice bundle"]/div/div/span[@class="larger"]')-
>item(0)->nodeValue;  // Assigning eBook price
```

12. Finally, we echo out all of the attributes of our object.

```
echo $cakePhpBook->title . '<br />';
echo $cakePhpBook->url . '<br />';
echo $cakePhpBook->release . '<br />';
echo $cakePhpBook->overview . '<br />';
echo $cakePhpBook->author . '<br />';
echo $cakePhpBook->coverUrl . '<br />';
echo $cakePhpBook->eBookPrice . '<br />';
echo $cakePhpBook->bundlePrice . '<br />';
```

There's more...

Using the class, which we have defined previously, we can instantiate a virtually unlimited number of instances, each with its own unique attributes.

We could extend the functionality of this class further by adding additional methods such as the functions from previous recipes in this book.

Thank you for buying
Instant PHP Web Scraping

About Packt Publishing

Packt, pronounced 'packed', published its first book "*Mastering phpMyAdmin for Effective MySQL Management*" in April 2004 and subsequently continued to specialize in publishing highly focused books on specific technologies and solutions.

Our books and publications share the experiences of your fellow IT professionals in adapting and customizing today's systems, applications, and frameworks. Our solution based books give you the knowledge and power to customize the software and technologies you're using to get the job done. Packt books are more specific and less general than the IT books you have seen in the past. Our unique business model allows us to bring you more focused information, giving you more of what you need to know, and less of what you don't.

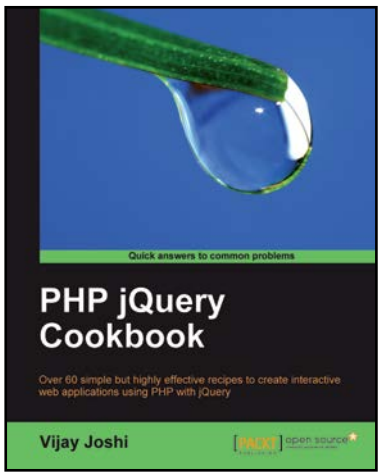
Packt is a modern, yet unique publishing company, which focuses on producing quality, cutting-edge books for communities of developers, administrators, and newbies alike. For more information, please visit our website: `www.packtpub.com`.

Writing for Packt

We welcome all inquiries from people who are interested in authoring. Book proposals should be sent to `author@packtpub.com`. If your book idea is still at an early stage and you would like to discuss it first before writing a formal book proposal, contact us; one of our commissioning editors will get in touch with you.

We're not just looking for published authors; if you have strong technical skills but no writing experience, our experienced editors can help you develop a writing career, or simply get some additional reward for your expertise.

[PACKT]
PUBLISHING



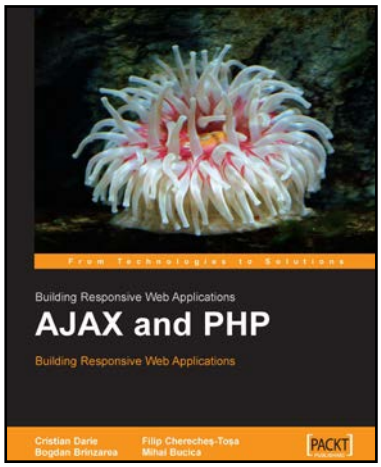
PHP jQuery Cookbook

ISBN: 978-1-84951-274-9 Paperback: 332 pages

Over 60 simple but highly effective recipes to create interactive web applications using PHP with jQuery

1. Create rich and interactive web applications with PHP and jQuery
2. Debug and execute jQuery code on a live site
3. Design interactive forms and menus
4. Another title in the Packt Cookbook range, which will help you get to grips with PHP as well as jQuery

---



Building Responsive Web Applications AJAX and PHP

ISBN: 978-1-90481-182-5 Paperback: 284 pages

Building Responsive Web Applications

1. Build a solid foundation for your next generation of web applications
2. Use better JavaScript code to enable powerful web features
3. Leverage the power of PHP and MySQL to create powerful back-end functionality and make it work in harmony with the smart AJAX client

Please check **www.PacktPub.com** for information on our titles