Snaphe Documentation

Florian Eckerstorfer

Matrikelnummer: 0725781
Rotenmühlgasse 10/4/1/2, 1120 Wien

181.196 Projektpraktikum

Wien, 20th of May 2011

Abstract

Snaphe is a Web Data Extraction library written in PHP. The library includes several tools to make writing and executing of wrappers much simpler but does not offer any wrapper generation functionality.

The basic idea behind Snaphe is that a human can write more robust wrappers than an algorithm and that this is a not a very time consuming task if the developer has advanced knowledge of web technologies and if there are only few different templates. The second import aspect of Snaphe is that installation and deployment is fairly simple and fast. Wrappers can run on every server and client which has PHP installed.

Contents

1 Introduction

Snaphe is a PHP 5[6] library to extract data from websites written by the author of this paper. The basic idea of Snaphe is that a human programmer with knowledge of web extraction technologies can write better and more robust wrappers than a machine can do just knowing some examples. Because it is written as a PHP library it can be integrated in any PHP application without much effort. Thus it is easy to write a wrapper which converts a machine unreadable website into an API. Snaphe already comes with a HTTP frontend and output adapters for XML and JSON. It is also possible to execute Snaphe from the command line or to invoke the extraction process from another PHP script. These frontends are called applications and it is possible to develop own applications.

The library contains a wrapper for PHPs cURL library, which makes it extremely easy to fake the user agent or HTTP headers and to route all requests through a proxy server. It is also possible to add multiple proxy servers and for every request, Snaphe randomly chooses one.

Listing 1 shows a wrapper which searches for a given query on Google and returns a list of results. The code of this example will be explained later ins this documentation.

```php
class GoogleSearchWrapper extends Snaphe_Wrapper_Abstract
{

  /** @var array */
  protected $models;

  /** @var array */
  protected $definedParameters = array('q');

  /** @var array */
  protected $requiredParameters = array('q');

  public function __construct(array $parameters)
  {
    $this->initializeParameters($parameters);
  }

  public function execute()
  {
    $this->setDefaultClient(new Snaphe_HTTP_Client());

    $page = $this->getPage('http://www.google.com/search?q=' . urlencode($this->
        getParameter('q')));

    $model = new Snaphe_Model_List();
    $titleRegExp = new Snaphe_Selector_RegExp('/<h3(.*)>(.*)<\/h3>/Us', 2);
    $titleRegExp->registerCallback('my_strip_tags');
    $descriptionRegExp = new Snaphe_Selector_RegExp('/<div class=\"s\">(.*)<br>/Us');
    $descriptionRegExp->registerCallback('my_strip_tags');

    $model->addValue('title', $titleRegExp);
    $model->addValue('description', $descriptionRegExp);

    $page->extract(array($model));
    $this->models[] = $model;
  }

}
```

Listing 1: Google Search wrapper

2 Installation

There is no need for an installation, because Snaphe is just a PHP library which can be copied to a directory. Although there are some things to do, depending on the frontend you want to use. In every case you need to download the source code from Github[1].

2.1 Command Line Interface

First of all, you should copy Snaphe to your hard disk and navigate into its base directory. You need to add execution rights to the Snaphe command line tool.

```
  chmod +x snaphe
```

Listing 2: Execution rights

By default Snaphe searches for wrappers in the `./wrappers` subdirectory of the Snaphe base directory. However you can change this by setting an environment variable.

```
  export SNAPHE_WRAPPER_DIRECTORIES="/path/to/your/wrappers"
```

Listing 3: Wrapper directory

2.2 HTTP frontend

You can also run wrappers through a HTTP frontend. Snaphe already contains a front controller, so you only need to point your webserver to the `./public` directory. The configuration in Apache would look like this:

```
<VirtualHost *:80>
  ServerName snaphe.example.com
  DocumentRoot /path/to/snaphe/public
</VirtualHost>
```

Listing 4: Wrapper directory

3 Execution

3.1 Command Line Interface

There are already some example wrappers in the `./wrappers` directory. For example the GoogleSearch-Wrapper performs a search request on Google and extracts the result page.

```
./snaphe --wrapper=GoogleSearch --q="web data extraction"
```

Listing 5: Execute GoogleSearch wrapper

The first argument is the name of the wrapper, the second argument is the search query.

By default Snaphe prints the result in a format which makes it easy to read by a human. When you are using the CLI tool (which is powered by the class `Snaphe_Application_Cli`) this format is cli. The HTTP application (`Snaphe_Application_HTTP`) uses by default the html format. If you want to reuse the data in another application you can use the xml or json format.

```
./snaphe --wrapper=GoogleSearch --q="web data extraction" --format=cli
./snaphe --wrapper=GoogleSearch --q="web data extraction" --format=html
./snaphe --wrapper=GoogleSearch --q="web data extraction" --format=xml
./snaphe --wrapper=GoogleSearch --q="web data extraction" --format=json
./snaphe --wrapper=GoogleSearch --q="web data extraction" --format=text
```

Listing 6: Formatting options

You can save the results by adding the argument file:

```
./snaphe --wrapper=GoogleSearch --q="web data extraction" --format=xml --file=/home/data
    /googlesearch.xml
```

Listing 7: Save to file

If you just want to execute a wrapper, you can surpress the output be choosing no as format. Please note that this also affects the file argument.

```
./snaphe --wrapper=GoogleSearch --q="web data extraction" --format=no
```

Listing 8: No output

3.2 HTTP frontend

To use the HTTP frontend you need to configure your webserver and then open it in a web browser. All options which can be used with the CLI can also be used in the HTTP frontend. They need to be passed as query string. For example, the following URL would execute the GoogleSearch wrapper, with the input parameter *web data extraction* and would display the results in HTML format: `http://snaphe.example.com/?wrapper=GoogleSearch&q=web+data+extraction&format=html`.

4 Components

Basically a wrapper is a class that implements the `Snaphe_Wrapper_Interface` interface. In practice every wrapper will extend the class `Snaphe_Wrapper_Abstract` because this class implements some helper methods which are quite useful when developing a new wrapper. The code to extract data must be placed in a method called `execute()`.

4.1 Selectors

A selector is an expression which is able to describe a single item on a page or a list of items. By default Snaphe comes with three different types of selectors.

XPath Extract a data value based on a XPath expression.

CSS selectors CSS selectors are much simpler to write than a XPath expression, but they are nearly as expressive. Internally they are transformed into XPath expressions.

Regular expressions Can be any Perl-Compatible regular expression and they should be used if it is not possible to parse a given website into XML.

4.2 Models

Models are grouped pieces of data values. Every data value is associated with a selector which extracts that piece of data, but there are also methods to manually define data values. In general they can be of any data type, but the standard output adapters shipped with Snaphe do not handle objects. Because models can contain array elements it is possible that a data value is a list of values. This is useful for example when someone wants to extract a list of books and every book is available in different editions. With nested lists it is possible to create relations between different items.

Snaphe contains two different types of models. A normal model is a single item, for example a product. But there is also a list model, which can contain multiple items, where every item has the same characteristics, for example a list of products.

4.3 Output adapters

After executing the wrapper and extracting data, Snaphe invokes an output adapter. An output adapter takes a list of models and other properties of a wrapper, like URLs, the name and the input parameters and returns a rendered representation in the desired format.

Snaphe contains the following output adapters:

Cli A format which is suitable for showing it in a command line interface.

HTML A HTML representation of the extracted data, which is readable by a person.

JSON This format is useful if the size of the output should be as small as possible or if the data should be processed with JavaScript.

No This adapter does not output or process anything.

Text A simple text-based format, which is readable by a person.

XML Creates a structured and valid XML representation of the extracted data.

In addition it is also possible to write custom output adapters and invoke them at runtime. Thus it is for example possible to write an output adapter which writes extracted data directly into a database or send it via email or text message. Let's consider the following scenario: A company sells products online and acquires most of their customers through search engines or search engine marketing. If a competitor suddenly lowers his prices, most customers would choose the cheaper competitor over the company. Thus the product manager wants to know as soon as possible if their competitors lower or raise their praises. We could write an output adapter which, instead of saving the extracted values into a file or database, loads the prices from our products database, compares them with the prices of our competitors and warn the product manager via email or text message if they are too high or low.

4.4 Applications

An application is a frontend for Snaphe.

5 Architecture

Snaphe was designed to work with any PHP version above 5.2.4, which is also the only dependency. If it should be possible to execute wrappers through HTTP requests, some HTTP server like Apache[2] or ngnix[8] is required. Snaphe was built that it uses native PHP libraries as often as possible. Because PHP is a highly popular scripting language for the web it is extremely good tested and, most of the time, very fast. Snaphe tries to avoid 3rd party libraries which are not sufficiently tested and actively maintained. Therefore it uses the cURL extension provided by PHP[3] and wraps them in a class called `Snaphe_HTTP_Client`. This class is kind of a web browser and tries to automate or at least improves handling of cookies, sessions, proxy servers and other tasks required to access web pages. To parse a HTML page (for example to execute XPath or CSS selectors) Snaphe uses the Document Object Model extension[4], which is enabled by default in every PHP installation. Because the DOM extension is not able to parse every HTML page (for example when the code is very invalid) it is sometimes not possible to use XPath or CSS selectors and the developer is required to fallback to regular expressions to extract data. Whenever it is possible to parse the HTML code Snaphe uses the SimpleXML extension[5] (another extension which is enabled by default in every PHP installation) to apply XPath expressions or walk through the DOM tree.

Figure 1 shows an UML diagram of the most important classes of Snaphe. Every wrapper is executed by an application, which should be a subclass of `Snaphe_Application_Abstract`. There is no interface because it is not required, but the abstract class offers some methods which are required by most applications. The application instantiates the wrapper class, which must be an implementation of `Snaphe_Wrapper_Interface` . There is also an abstract class `Snaphe_Wrapper_Abstract` which implements some common methods. This concept can be found throughout Snaphe. For every interface there is an abstract class which implements the most common methods. A wrapper can extract data from one or more pages. Every page is represented by an `Snaphe_Page` object. This object requires a `Snaphe_HTTP_Client` object to execute HTTP requests. `Snaphe_Wrapper_Abstract` offers the possibility to define a default HTTP client object which is used in all pages of a wrapper. When someone wants to extract data from a page, he has to invoke the `extract()` method of `Snaphe_Page` providing an array of models. A model is an implementation of `Snaphe_Model_Interface`. Each field in a model is defined by a unique name and a selector. A selector is an implementation of `Snaphe_Selector_Interface`. During the extraction the extracted value is stored in the model. After the wrapper is executed the application retrieves the models and passes them to an implementation of `Snaphe_Output_Interface`. The output adapter renders the models in the desired format and outputs or saves them.

6 Writing a Wrapper

At first we need to create a class which implements `Snaphe_Wrapper_Interface`

```
class GoogleSearchWrapper extends Snaphe_Wrapper_Abstract
{


```

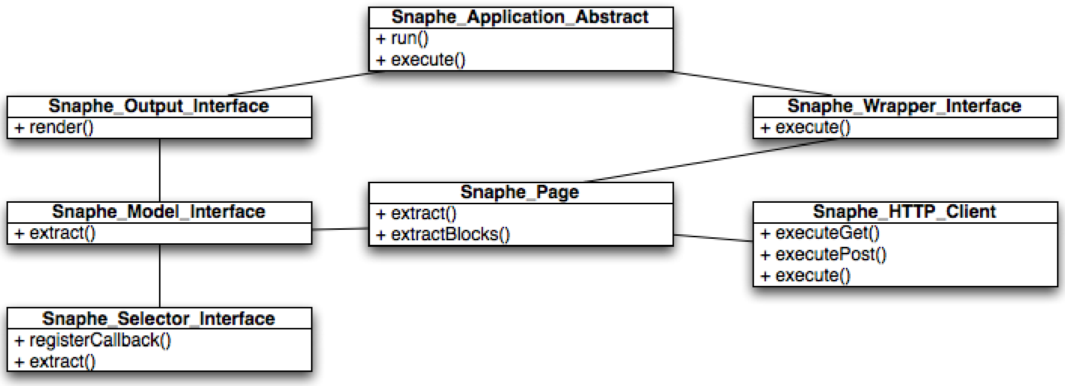

Figure 1: UML diagram showing the relation between the most important classes of Snaphe.

```
    public function __construct(array $parameters)
    {
    }

    public function execute()
    {
    }

}
```

Listing 9: Empty wrapper class

Next we need to define which parameters the wrapper accepts. By default a parameter is optional, but you can make them required:

```
protected $definedParameters = array('foo', 'bar');
protected $requiredParameters = array('foo');
```

Listing 10: Defining parameters

In this example there are two parameters foo and bar and foo is required, while bar is optional.

Snaphe passes the parameters to the wrapper through the constructor. `Snaphe_Wrapper_Abstract` offers a method to check these parameters:

```
public function __construct(array $parameters)
{
    $this->initializeParameters($parameters);
}
```

Listing 11: Initialize parameters

The goal of this first wrapper is to extract the title and description from the Google search result page.

At first we need to create a `Snaphe_HTTP_Client` object, which will be used as a HTTP client. We can set these client as the default client.

```
public function execute()
{
    $client = new Snaphe_HTTP_Client();
    $this->setDefaultClient($client);
}
```

Listing 12: Create HTTP client

Next we need to create a page object. We could instaniate `Snaphe_Page` and then specify the HTTP client, the URL and so on but it is simpler to use the `getPage()` method of `Snaphe_Wrapper_Abstract`.

```
$page = $this->getPage('http://www.google.com/search?q=' . urlencode($this->getParameter
    ('q')));
```

Listing 13: Get a page

Next we need to define a model and some selectors. We want to define two selectors using regular expressions.

```
$model = new Snaphe_Model_List();
$titleRegExp = new Snaphe_Selector_RegExp('/<h3(.*)>(.*)<\/h3>/Us', 2);
$titleRegExp->registerCallback('my_strip_tags');
$model->addValue('title', $titleRegExp);
$descriptionRegExp = new Snaphe_Selector_RegExp('/<div class=\"s\">(.*)<br>/Us');
$descriptionRegExp->registerCallback('my_strip_tags');
$model->addValue('description', $descriptionRegExp
```

Listing 14: Define the models

Let's take a closer look at the first selector:

```
$titleRegExp = new Snaphe_Selector_RegExp('/<h3(.*)>(.*)<\/h3>/Us', 2);
$titleRegExp->registerCallback('my_strip_tags');
```

Listing 15: Defining a selector

The first argument of the constructor is a standard PHP regular expression. The second argument is the index of the match in the regular expression. In that example it would extract the inner HTML of H3 tag and not the attributes. Next we define a callback function which is applied to every extracted value. `my_strip_tags()` is a wrapper for the standard PHP function `strip_tags()`, but it also takes an array as a argument. Here is the code:

```
function my_strip_tags($value)
{
    if (is_array($value))
    {
        for ($i = 0; $i < count($value); ++$i)
        {
            $value[$i] = strip_tags($value[$i]);
        }
        return $value;
    }
    return strip_tags($value);
}
```

Listing 16: Strip tags function.

There is one more thing to do. Extract the models and save them in the property `\$models`:

```
$page->extract(array($model));
$this->models[] = $model;
```

Listing 17: Initializing the extraction

Note: The `extract()` method takes an array as argument. You can define is much models as you want. Also, the source code of the defined website is fetched on demand (if `extract()` is never called, no HTTP request is executed).

You can now execute the wrapper:

```
./snaphe --wrapper=GoogleSearch --q="web data extraction"
```

Listing 18: Executing the wrapper

7 Practical Experiences with Snaphe

Snaphe was developed because of the need for a light weight web data extraction system for a small Internet company. The main requirement was that the system integrates into the company's infrastructure, thus PHP was chosen as scripting language. Because wrappers would always be generated by skilled PHP developers, PHP was also chosen as wrapper generation language. Experience in the company's industry had shown that competitors change the layout or the structure of their HTML content at most once a year and because there is a very limited number of different websites to watch, it was possible to use a manual wrapper generation approach. The company and their competitors operate in a field where a customer visits their website, selects a start and end date and a location. Every location contains one or more shops which offer a limited range of products. The products are very similar in all shops in all locations, but some products are not available in all shops. Each product also can contain some options which change the price of the product. While the base price is nearly the same in every shop, in every location for every competitor, the interesting part is the discount. Each company applies a different discount for different locations and sometimes even for different shops and for time ranges.

The main motivation for extracting the product data of its competitors was for the company to get an overview of the different prices and discounts in different time ranges. Therefore it was essential to compare the extracted prices with the prices in the company's database. To accomplish this, a custom output adapter was written which mapped the extracted models to the models in the database. Before Snaphe was available, the marketing division of the company manually extracted their competitor's prices and wrote them into an Excel spreadsheet. In the first stage of implementing the new automated data extraction system the result was a CSV file which can be imported in their existing Excel spreadsheets. The next stage of implementation will contain a tight integration with the Symfony Framework[7] based backend of the company's website.

Some notes of experiences while developing wrappers with Snaphe:

- Snaphe does not support any form of JavaScript execution. Therefore it was not possible to navigate through Ajax only interfaces. But it was possible to simulate the asynchronous requests and because most sites respond to these requests in a XML or JSON based format it was easy to extract the desired data. And it is very likely that these formats do not change, even if the website gets a new layout.
- The developer of a wrapper should take a close look at the source code of a website. Sometimes the desired data is also available hidden in the source code, for example in a hidden form field. This is especially true for JavaScript (but not Ajax) based interfaces.
- Most of the time XPath or CSS selectors are the easiest way to select values to extract.
- But sometimes it is easier to work directly on the string model and use regular expressions.
- Often it is useful if you are able to use the power of a real, full powered scripting language instead of the functionality provided by a graphical interface.
- For small data extraction tasks it is easier to combine the actual extraction with data cleaning and mapping instead of using different tools for each task.
- It was easy to integrate Snaphe into the very popular Symfony Framework. Only a very small script was required to use the Symfony command line tool instead of Snaphes.

All in all Snaphe is a very powerful tool if it is in the hands of a skilled PHP developer and if applied to the correct task. It is especially useful if the environment is already prepared for PHP applications.

8 Advantages of Snaphe

While there are some disadvantages, Snaphe has a lot advantages when we look at it from the right point of view. First of all it is directed to advanced web developers, but not towards web data extraction specialists. Therefore the target group has knowledge about web technologies like HTML, CSS, XPath, PHP, JavaScript and others, but no experience with specific web data extraction technologies. Because wrappers are developed in native PHP code the developer does not need to learn a new programming or scripting language, but rather only very few new methods. Another advantage of Snaphe is that it does not require any specific execution software. Any server or client with PHP installed can execute Snaphe wrappers. We should also keep in mind that Snaphe was not designed to extract content from thousands of sites. Snaphe was designed for personal and small or medium business usage. Thus when the task is to extract data from a limited number of websites it is easier and cheaper for a developer to use a well known technology like PHP. It is also not required to buy, configure and maintain a server dedicated to the web data extraction task, but the developer can use any web server with PHP installed. He can use an established technology like cron jobs to schedule the wrappers. If the extraction tasks do not require to run repeatedly on a specific date it is also possible to run the wrappers from a desktop

PC or notebook. As mentioned before, these characteristics make Snaphe ideal for small and medium web data extraction tasks.

References

[1] Florian Eckerstorfer. Snaphe. https://github.com/florianeckerstorfer/Snaphe.

[2] The Apache Software Foundation. Apache http server project. http://httpd.apache.org/.

[3] The PHP Group. Php: Client url extension. http://php.net/manual/en/book.curl.php.

[4] The PHP Group. Php: Document object model extension. http://php.net/manual/en/book.dom.php.

[5] The PHP Group. Php: Simplexml extension. http://php.net/manual/en/book.simplexml.php.

[6] The PHP Group. Php. http://php.net/.

[7] Sensio Labs. Symfony. http://www.symfony-project.org/.

[8] Igor Sysoev. nginx. http://nginx.org/.