university of groningen

faculty of mathematics and natural sciences

Automatic website slicing: An open specification and implementation

Bachelor thesis

July 5, 2012

Student: Alex Jeensma

Primary supervisor: Arnold Meijster

Secondary supervisor: Michael Wilkinson

Acknowledgements and copyright

A lot of people have contributed, directly or indirectly to this thesis. First of all I would like to thank my girlfriend Marjan for her unconditional support, both in my personal as well in my professional life. And of course I want to thank my first supervisor Arnold Meijster for his support and supervision.

With regards,
Alex Jeensma

Abstract

This thesis introduces a specification, to structure websites designed in Photoshop files [1], so that they can be converted to HTML files. This is a tedious process that can take a long time for a human being (thus costing a lot of money).

The process is mostly based on heuristics and best practices. Our goal is to specify these heuristics and best practices in a specification that covers a lot of the design techniques used today.

We also introduce a parser that can read Photoshop files that follow the specification, and define the bounds that the specification is limited by. We will use the parser on several designs and comment on its merits and shortcomings. Later on in the thesis we suggest some options to improve the parser.

Contents

Chapter 1

Introduction

Websites have evolved from being a static resource into dynamic pages designed in graphic programs. Most agencies design their websites in Photoshop. Subsequently these Photoshop files are ”sliced” by a programmer.

The process of slicing is firstly dividing the design up in parts, and secondly embedding it into an HTML page. The slicing process is a process that has formed itself over the years, introducing various common practices to cope with the limitations of the popular CSS2 standard. It also aims to avoid bugs presented in various internet browsers.

There have been several initiatives to automate the slicing process, but none of them have really caught on. This is mostly due to dynamic standards and the ever changing way people design websites.

That is why we want to introduce an open specification, that will define a lot of metadata that allows us to interpret Photoshop files in such a way that we can semi-automatically realize them in HTML. The specification will be open and free for anybody to use (**not** for commercial use, this is to stimulate the development of the framework). This way we will try to set a platform for a wide support of automatic website slicing.

Furthermore, we will introduce pseudocode for a parser, and implement a initial prototype that implements this parser. This parser will follow the open specification strictly and will be hosted on Github (for easy collaboration).

Chapter 2

Web and its terminology

In this chapter we will describe most of the used techniques for designing and building websites. If you are already familiar with Photoshop and HTML it is safe to skip this chapter.

2.1 HTML, CSS and JavaScript

Three open standards are the building blocks of any webpage: HTML represents the structure of a page, CSS represents the style of a page and JavaScript defines the interaction with the webpage.

2.1.1 HTML

HyperText Markup Language (HTML) is the main language to mark up webpages. A tag is a delimiter for an element, which can be nested and contains the content for each element. Each tag has a begin ($<$ `tag` $>$) and an end tag ($<$ /`tag` $>$). Some tags are self closed (for instance, $<$ `img`/ $>$). These are the so called void tags [7].

Tags also have optional attributes; there are some global attributes and attributes that depend on the type of tag. Attributes control the behavior of the tag. For instance, attaching an id attribute defines that this element has a certain unique identifier within the document.

2.1.2 CSS

Cascading Style Sheets (CSS) defines the style of the webpage. It has been introduced to separate the document content from the style of the website. It uses so called selectors to target elements on a webpage. For instance: `div{background` : `#FF0000;}` changes the background colour of all `div` elements (colors prepended with a hashtag are hexadecimal colors in RGB format).

Our open source project will use the CSS2 standard, because the CSS3 standard is not supported by some major browsers yet.

2.1.3 JavaScript

JavaScript offers some realtime dynamic functionality for webpages. For instance, creating a nested menu that animates in and out. There are several libraries that overcome differences between browsers and offer some extended selectors. One of those libraries is jQuery [6].

jQuery and other JavaScript libraries introduced an easy way to work with AJAX (Asynchronous JavaScript and XML, allows developers to retrieve data in asynchronous way. The web is mostly synchronous). This allowed developers to communicate with the backend of their applications (for instance, saving data without refreshing the webpage).

2.1.4 Web browsers

Web browsers interpret the webpages and render them to the screen. They implement the standards defined for each language (most of the time). At the moment the main 5 popular browsers are Google Chrome, Firefox, Internet Explorer, Safari and Opera. Internet Explorer has a notable track record of not always following the specifications to the letter.

Most browsers are open source, several of them are based on the WebKit platform (Safari and Google Chrome, which itself is based on Chromium).

2.2 Photoshop

2.2.1 About Photoshop files

The Photoshop file format is the native file format for Adobe Photoshop. The specification of its file-format is open. This allows other programs to interpret the format. The buildup of a Photoshop file is shown in Figure 3.1.

There are several readers for the format, most of them are open. For instance, ImageMagick is able to read the format, and exports all the layers in PNG format [5].

We will use the ImageMagick library in our initiative. The fact that it can only export PNG files is actually an advantage as the lossless compression of PNG does not result in any degradation of quality. PNG files that are too big, will be converted to JPG automatically by the parser.

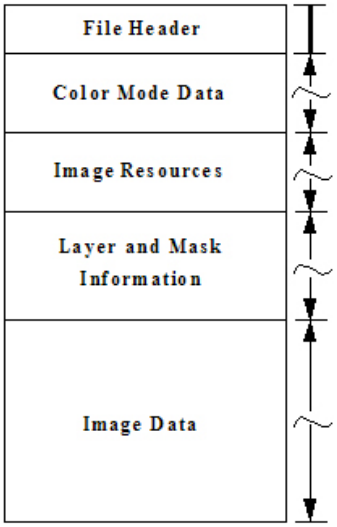

Figure 2.1: Specification of a Photoshop file

2.2.2 Layers and Groups

Photoshop itself groups image data by so called layers and groups. We will use this to our advantage when writing our parser.

The layers also allow the designer to make complex designs by using the hierarchy of layers. Groups are used to group layers. They also have a hierarchy. Layers have a name that is used to identify the layer. We will use the layer name to our advantage when defining our specification.

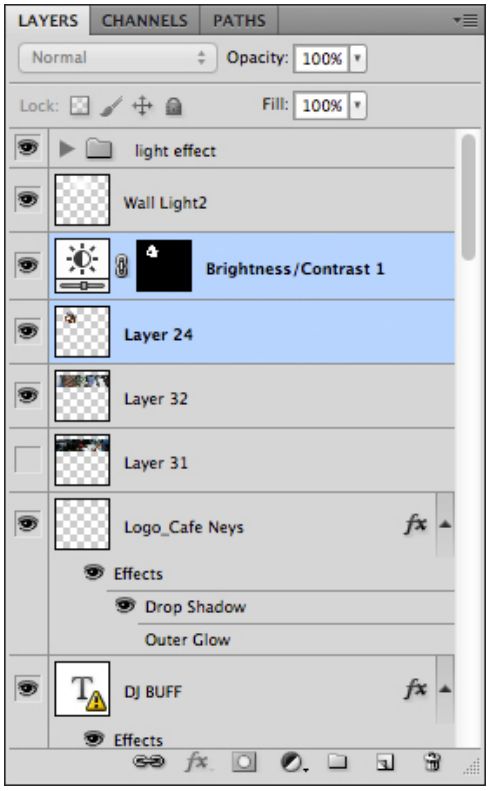

Figure 2.2: The layer panel in Photoshop

2.2.3 Why Photoshop for designing websites?

Websites have evolved from basically plain text towards heavy graphical pages that try to entice the user in to undertake action (placing an order, contacting a company). This is mostly why Photoshop is used to design powerful graphic websites.

Companies want to be unique. For most companies the website is a powerful mean of communication. That is why they want a graphical page to outbeat their competition.

Chapter 3

Related work

There are some initiatives that touch on this subject, and even implement a parser. These parsers all accept files that conform to different specifications. Some parts of these specifications are also embedded in our parser and specification.

3.1 The slice tool in Photoshop

When Adobe Creative Suite 3 was released, the slice tool was first introduced under the name Imageready [2]. Adobe discontinued Imageready and embedded a lot of functionality from Imageready in Photoshop. That is why we will now refer to it as the slice tool (in Photoshop).

Although, HTML does not have any semantics, the structure and usage of tags is very important. Early versions of the slice tool could only generate HTML table based code. The use of tables to build up a webpage is not considered as a good practice [4]. Therefore, use of the slice tool is generally discouraged.

There are some positive features Imageready introduced: the ability to link a slice to a layer reduced the amount of work needed when a design changed. Furthermore, it introduced a way for developers to see how optimizing an image for the web would change its quality in realtime.

However the practical use of the tool was little, especially because it forced layouts to correspond exactly to what the designer made (it did not feature scaling). Moreover it also introduced compatibility issues with different browsers.

3.2 Sitegrinder 3

Sitegrinder is a commercial solution that introduces a limited specification, and a parser that is able to parse Photoshop files and convert them into fully functional HTML pages. It also implements their own CMS (Content Management System).

The specification is limited in the sense that it has some limitations that are used in a lot of designs (custom fonts, no use of CSS sprites, incorrect separation of markup and functionality,

deprecated javascript). Another limitation is the readability of the code, the code is not indented and the start and finish tags are not documented.

Although, Sitegrinder is a good start, it would also benefit from a complete and universal specification. Furthermore, Sitegrinder is a commercial package. Which implies that you also pay for the content management system behind it.

3.3 psd2cssonline.com

Another good initiative is called psd2cssonline.com. This free web service allows you to upload Photoshop files. It will then generate all the necessary HTML and CSS, based on the specification embedded in the Photoshop file.

However this initiative is mostly limited by its specification. The specification is bulky and has no real practical use. It is also limited by the maximum file size (10 MB). Although, many designs do not exceed that limit, still a lot of them do.

Furthermore, it does not implement some common practices that have became standard over the years (CSS sprites, CSS sliding doors). These and other limitations exclude this initiative for professional use.

3.4 Adobe Fireworks

Adobe Fireworks was mainly implemented as a vector graphics editor. The product has been created as an alternative for fast mockups. It supposedly has a better compression rate than Photoshop [3]. This means that the images that Fireworks produces are of a smaller size than images produced by Photoshop. Moreover the product has some more options when working with slices.

Because the Slice tool in Photoshop and Adobe Fireworks are almost similar, the same arguments apply to why the process is far from ideal.

3.5 Conclusion

Although, there are several initiatives that introduce the concept of automatic slicing, they are limited in their specification and often produce marginal results. After all these years, and several initiatives, does this mean that the problem is unsolvable? In this thesis we will demonstrate by implementation that this is not necessarily true.

Chapter 4

Guidelines for webpage design with Photoshop

In this section we will provide a short overview of the languages and file formats that are involved with the open specification. After that we will define a specification for automatic web slicing. All of these mentioned languages have an open specification and standard.

4.1 Layer metadata and specification

We will express our specification through the group and layer names. For instance, a layer named $< div > hello\_world$ will result in the following HTML: <**div id='hello_world'**></**div**>.

That way we can attach all the information we need in the Photoshop file, without the need of any external configuration files. Our intention is to mold the layers and groups in Photoshop, so that we can extract all information from it.

4.1.1 The top level groups, and special definitions

We will define two groups on the top level, one with an arbitrary name that encapsulates the website without any metadata, and a other group with a fixed name called HTML. This way, designers and slicers can work in one shared Photoshop file. The only disadvantage of this method is that it will double the size of a Photoshop file. But in a world where the price per uploaded GB is 8 eurocents, this is not considered a problem.

There is an optional group inside HTML, named Configuration. This group will not be parsed and contains key and values for specifying the global options for the parser and its generated HTML.

The configuration group does have multiple options, for instance, specifying the way how start and end tags get defined, or what the title attribute of the website should be.

4.1.2 Groups and layers

In our specification, we will consider groups and layers, to be essentially the same. The only difference between layers and groups, is that groups can have children, while layers can not. Each layer or group should encapsulate exactly one HTML element. The specifications are given in a regular expression format.

The specification per layer will be:

$(<\texttt{tagname}>)?\texttt{layer\_name}([\texttt{ns}:\texttt{key}=\text{“}\texttt{value}\text{”},\texttt{ns}:\texttt{key}=\text{“}\texttt{value}\text{”})$

$<\texttt{tagname}>$ is optional and will default to the div element.

`layer_name` is mandatory and must comply to the allowed characters that can be used in either the id or class attribute. The regular expression that the name should match is:

$['\texttt{A}'-'\texttt{z}']\{1\}['\texttt{A}'-'\texttt{z}','\texttt{0}'-'\texttt{9}','\_','-']*$

Although, the ' : ' and '.' character are allowed in names according to the official HTML4 specification, we omit them here because it causes trouble with certain JavaScript libraries. Also note that the name cannot begin with a number or a special character.

The last part of the specification is metadata; the format of the metadata will be specified in the next subsection. Metadata is optional, in the sense that if no metadata items are specified, all keys will be set to their default value.

4.1.3 Layer metadata

The metadata of layers, is a map with the following key format:

`namespace : key`

The namespace is a global identifier for a key. This allows an easy and clear way to add new keys and values. It also allows keys with the same name to exist (limited to their namespace). The keys and the namespace should conform to the following regular expression:

$['\texttt{a}'-'\texttt{z}','-']+$

This regular expression limits the complexity of keys and namespaces, but will improve its readability.

For a full specification of the metadata, we refer to the namespaces and their keys chapter. It also helps us when a special namespace is introduced in subsection 4.2.1. Multiple metadata pairs should be separated by a comma.

4.1.4 Examples of usage

In this section we will give some examples of the specification. Do not expect the examples to be normative for what you would expect in a real HTML webpage.

These examples also omit metadata. A full specification of metadata will be given in a next section.

Layer name	Results in the following HTML	Notes
`test_5`	< `div id` = '**test_5**' > .. < /`div` >	
< `table` >	< `table` > .. < /`table` >	
< `ul` > `nice_ul`	< `ul id` = '**nice_ul**' > .. < /`ul` >	
< `li` > `active_item`	< `li class` = '**active_item**' > .. < /`li` >	class attribute instead of id attribute
`1_division`		invalid layer name, starts with a number

Example 4 (< `li` > `active_item`) only occurs when there are clashing layer names.

Note that the parser that reads files that comply to the specification, must conform to the self closing of elements if it is a void tag [7]. A void tag is a tag without an explicit end tag. For instance, $< img/ >$.

4.2 Namespaces and their keys

Namespaces have an identification, this description can be any string. All of the namespaces also have a shorthand notation, that needs to conform to the given namespace regular expression.

4.2.1 Attribute: a special namespace

First off, we will define a special namespace. This namespace acts a little different from its counterparts. The full name of the namespace is Attribute, and it is shortened to `attr`.

This namespace allows developers to directly add attributes to the given element. This attribute is one of the reasons we chose to allow the '-' character in our key names; there are 2 attributes in HTML that contain a '-' (`accept` − `charset` and `http` − `equiv`).

Allowing the '-' character also means that we can accept the so called data-* attributes introduced in HTML5 [8].

Some examples are:

Layer name	Results in the following HTML
`test`[`attr` : `title` = '**value**']	< `div id` = '**test**' `title` = '**value**' > .. < /`div` >
< `table` > [`attr` : `cellpadding` = '**0**']	< `table cellpadding` = '**0**' > .. < /`table` >
< `a` > [`attr` : `href` = '**link**']	< `a href` = '**link**' > .. < /`a` >

As shown above, the `attr` namespace is a very flexible namespace, that provides users more control over their elements.

Values can be any string, it must be enclosed by double quotes. Double quotes inside the string must be escaped with a \.

4.2.2 Namespace: Background

The background namespace controls parts of the CSS property ”background”, it will be abbreviated by `bg`.

The following properties are supported:

Key	Value	Default
repeat	`x` | `y` | `xy`	None

The **repeat** key controls the repetition of a background. For instance, if you want this element to repeat itself horizontally (of course limited by its parent element), you would set this value to x. The default value is repeating over the x and y axis.

4.3 The configuration group

The configuration group only includes transparent layers. The layer name should be as follows:

(`optional_namespace`) : `key` = ‘**value**′

The key must adhere to exactly the same regular expression given in subsection 4.1.3. The value must adhere to exactly the same demands as specified in subsection 4.2.1

Note that a namespace for a configuration group is not always necessary. Since most of the configuration options are set with one key; it would mean that we have to specify a namespace which is essentially purposeless.

4.3.1 Auto comments after elements

Configuration option	Value	Default
comment-template	value start tag | value end tag	None

Notes: `tag_id` and `tag_name` will be replaced with the tag identification (.class or #id) and with the tag name (for instance, div).

Motivation: Most webpages introduce a lot of code. For instance, nested divisions and complex structures. That is why it is easy to delimit code by using HTML comments. For instance, without this configuration option, the structure would look like this:

Example: The following example shows how this tag can be used.

```
<div id="encapsDiv">
    <div id="anotherDiv">
        <div id="nestedAgain">
            Contents..
        </div>
        <div class="nestedAgain">
            Contents..
        </div>
    </div>
</div>
```

When enabling this option, the HTML would look like this:

```
<div id="encapsDiv"><!-- / #encapsDiv \ -->
    <div id="anotherDiv"><!-- / #anotherDiv \ -->
        <div id="nestedAgain"><!-- / #nestedAgain \ -->
            Contents..
        </div><!-- \ #nestedAgain / -->
        <div class="nestingClass"><!-- / .nestingClass \ -->
            Contents..
        </div><!-- \ .nestingClass / -->
    </div><!-- \ #anotherDiv / -->
</div><!-- \ #encapsDiv / -->
```

This increases the length of the HTML document (thus the size), but improves readability of the document drastically.

4.3.2 Metadata

The metadata namespace controls the meta tags in the head tag, it is prepended by *meta*. These are mostly used to provide hints to search engines.

The key of a metatag is used as the name, the value is the value of the metatag.

Configuration option	Value	Default
meta:robots	Value of the meta robots tag	None
meta:keywords	Value of the keywords tag	None
meta:description	Value of the meta description tag	None

Some examples:

meta:robots='index,follow', **meta:keywords='key,words'** and **meta:description='Description that will be shown in the Google Search results.'**

Results in these tags (within the head tag):

```
<meta name="robots" content="index,follow" />
<meta name="keywords" content="key,words" />
<meta name="description" content="Description that will be shown in
    the Google Search results." />
```

4.3.3 Head data

Prepended by *head* these configuration options control hints to browsers and search engines. Examples of usage are the page title and doctype of the webpage.

Configuration option	Value	Default
head:title	Value of title tag	Title of the webpage
head:doctype	Full Doctype (including <html>)	XHTML 1.0 Transitional
head:encoding	Encoding type	UTF-8
head:base	Full base path including trailing slash	ϵ

With HTML5 some new properties for the head were introduced. The parser should easily be extended to accept those new properties.

4.3.4 Scripts

If you want to include some Javascript in your webpage, you can specify multiple paths separated by a comma to the scripts directive, for example:

Configuration option	Value	Default
scripts	comma separated list of paths to .js files	ϵ

So if you were to give this command: scripts=**'js/jquery.js,js/script.js'** the parser would include the files with a script directive in the correct order.

Chapter 5

A parser in Java

5.1 Introduction

The parser that accepts files that comply to our proposed specification is written mainly in Java. We expect that there are a lot of Java developers around that are willing to assist on an OpenSource project that will eventually help to produce better webpages.

Organizing a program like this takes a lot of thinking about the structure. In section 5.2. We will give an overview of the specific structure and the paradigms that have been used.

Java Package	Class	Description
(default package)	*Main*	Runs the Parser with command line arguments
firebolt	*Configuration*	Holder for all the configuration options
firebolt	*Document*	Basically holds the document, including its top element (body), reference to the stylesheet, etc
firebolt	*Element*	Holds a HTML element, its attributes, parent element, also handles building of the Elements style
firebolt	*CommonPractice*	Abstract class for managing common practices
firebolt	*Parser*	Parses the Photoshop document to HTML
firebolt.css	*Stylesheet*	Holds all the stylesheet selectors for the Document
firebolt.css	*StylesheetSelector*	Single Stylesheet selector
firebolt.metadata	*Attribute*	Implementation of Metadata, adds a attribute to an element
firebolt.metadata	*CSS*	Implementation of Metadata, adds a stylesheet definition to a selector
firebolt.metadata	*Metadata*	Parent class of all metadata, does some basic cleanup before parsing the definition
firebolt.metadata	*MetadataParser*	Parses the metadata and calls the right Parsing class

5.2 Parser overview

5.2.1 Summary

First off all we will ignore the Java packages starting with PSD. This is because they contain a lot of irrelevant code to our application. They parse the Photoshop file so that we can read it with Java. The interesting classes are mentioned in the table above.

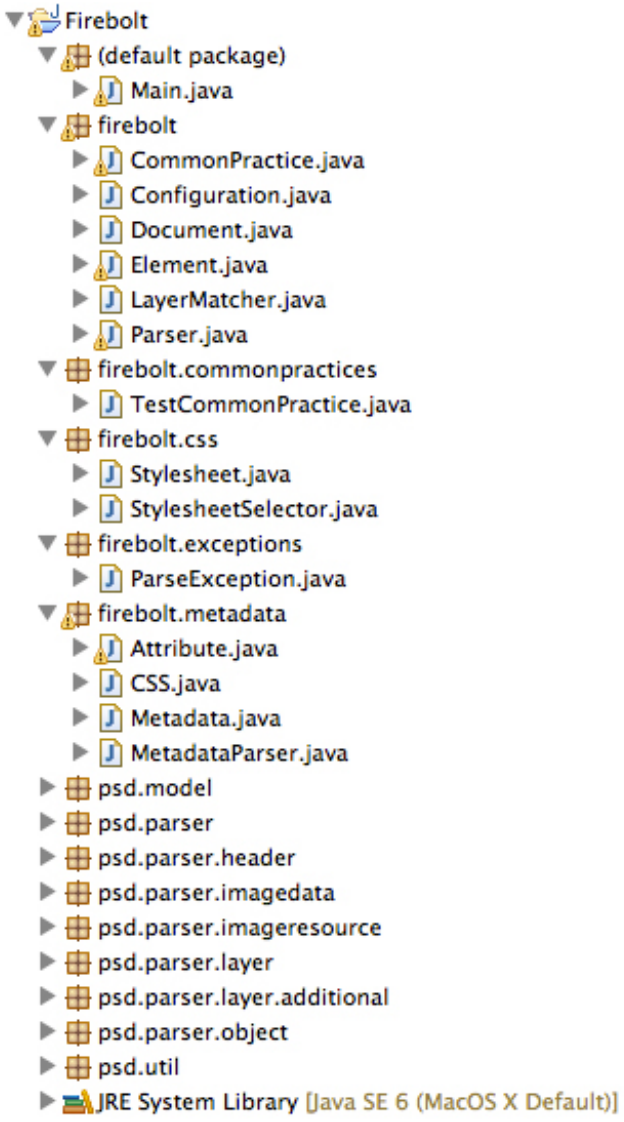

Figure 5.1: Interesting Parser classes

5.2.2 Configuration

The Configuration class is a wrapper for the Configuration options specified in Chapter 4.3. It uses a HashMap to store the key-and-value pairs.

5.2.3 Document

The Document class holds the outer layer of the whole HTML document, including its resources like images and stylesheets. It holds a reference to the Configuration object, and to the top most element of every HTML page (the body tag). It uses the print method to generate the final HTML. It also includes the DOCTYPE [9] and references to the stylesheet(s).

5.2.4 Element

The Element class is the most complex class. It handles everything you would expect from a Parser. It deals with the tag name, attributes, prints them, determines the style based on its own attributes and occasionally the siblings of the Element. Due to its complexity we will explain the most important methods of this class (the methods we ignore are methods that are simply getting and setting properties):

Constructor

The Constructor adds the ID attribute (if it is available). It also creates a new StylesheetSelector for attaching styles to the Element. After doing the actions mentioned, it passes the attributes to the MetadataParser class together with the current Element so that the MetadataParser can do its work on the current Element (if necessary).

recursiveBuildStyle

This method is called once for the body tag (from the Parser). Basically it traverses the body tag and all its children (recursively) and determines and builds the style (CSS) of the element. This is executed by calling the buildStyle method explained below.

buildStyle

This method divides the problem of styling an Element based on a Parsed Photoshop file into several parts:

- Scaling the element according to the Box Model
- Determining the Element's background
- Positioning according to flow order and potential floating
- Calling defined common practices (for example: the SlidingDoors common practice)

This method will be improved over time, eliminating edge cases and implementing more common practices.

getNextInFlow

Gets the next Element in the so called Document Flow [10]. It does this by detecting the position in the siblings list.

getPrevInFlow

Gets the previous Element in the so called Document Flow [10]. It does this by detecting the position in the siblings list.

merge

This method is necessary because Photoshop does not allow us to attach a style to a Group type. Thus we merge the Layer and the Group with exactly the same name. This allows us to style the Group element.

buildAppend

This builds the string that we append to the element if the Configuration option comment-template (see section 4.3.1) is set.

printRecursive

Method that prints the element and its children (by calling printRecursive in turn). It also handles indentation.

5.2.5 CommonPractice

The CommonPractice class is an abstract class that functions as a building block for a common practice. As mentioned in the introduction, slicing a webpage is based on a lot of common practices, e.g. the Sliding Door common practice [11]. This common practice allows us to create a variable sized element with so called caps. These common practices can be formalized and implemented by creating a Child class that implements the CommonPractice class.

5.2.6 Parser

This class takes an input file and an output folder. It parses and extracts all the information from the Photoshop file. The parseLayer method is the most interesting part, it takes a layer name and uses regular expressions to extract all the metadata from the layer name.

5.2.7 Stylesheet

The Stylesheet class holds all the StylesheetSelectors for the whole document. It also holds the print method for the concatenation of all Selectors.

5.2.8 StylesheetSelector

The StylesheetSelector holds the complete selector, and all its properties. It implements a print method (that in turn is used in the Stylesheet print method).

5.2.9 Attribute

This implementation of Metadata is simple and just adds the key and value to the Element its HTML attributes.

5.2.10 CSS

Implementation of Metadata class adds a key and a value to the StylesheetSelector.

5.2.11 Metadata

Basic class that holds all the Metadata clean up methods. It is the parent of the above classes.

5.2.12 MetadataParser

Calls the right Metadata class for the job.

Chapter 6

Future work

6.1 Circumventing Photoshop limits

Photoshop allows a maximum depth of 5 nested groups. A strange decision as Photoshop files are flat files. The most portable way of specifying a nested relationship would be a parent and child strategy (building a tree is relatively cheap in computation and memory). Adobe chose an XML like technique that results in these kind of limitations.

This is the main reason that complex Photoshop files can not be converted into HTML files. A workaround could be to define the structure of the HTML document into another file. Or creating theoretical symlinks (linking a group to a group on a lower level).

6.2 Enhancing the parser

The implemented parser has several shortcomings at the moment. Firstly, it can not recognize patterns that repeat in a direction and it can not handle text layers correctly. Moreover, it also has a basic way of determining the size and position of an Element (it is based on the layer dimensions).

6.3 Implementing common practices

The basics of all future common practices are encapsulated in the CommonPractice class. A good first Common Practice class would be the Sliding Doors common practice [11].

6.4 Writing a compare application

To keep backwards compatibility a compare application would be necessary; this application is able to compare sets of HTML and Photoshop files to correct references. This would prevent new features from breaking a different part of the parser.

Chapter 7

Conclusion

We have barely scratched the surface on the surface of automatic Photoshop to HTML conversion.

Even though it is a very complex subject. We have demonstrated that with the right specification a lot can be achieved. Perfect conversion will probably be impossible. However that does not mean that we can take the mind numbing part out of the manual slicing.

However there are a lot of hurdles to take. The current specification does allow a wide range of Photoshop files to be converted; the current parser does not yet support a lot of features that the specification introduces.

In the end, we think that we have shown that it is possible to automatically convert a Photoshop file to its matching HTML. Using a strict specification that is based on practice and common practices.

Appendix A

Summary of specification

In this Appendix the current specification and the source code of the parser are listed. The full source code is on GitHub for easy collaboration. The full link is *https://github.com/alexjeen/Firebolt.git*.

A.1 Layer Specification

(< `tagname` >)?`layer_name`([`ns` : `key` = "`value`", `ns` : `key` = "`value`")

< `tagname` > is optional and will default to the div element.

`layer_name` is mandatory and must comply to the allowed characters that can be used in either the id or class attribute. The regular expression that the name should match is:

['A' − 'z']{1}['A' − 'z', '0' − '9', '_', ' − ']∗

A.2 Metadata in a layername

The metadata of layers, is a map with the following key format:

`namespace` : `key`

The namespace is a global identifier for a key, this allows an easy and clear way to add new keys and values. It also allows keys with the same name to exist (limited to their namespace). The keys and the namespace should conform to the following regular expression:

['a' − 'z', ' − ']+

Appendix B

Formal grammar

Formal grammar for every layer name. Hierarchy is expressed through the group mechanism in Photoshop. The start symbol is always the <body> layer.

⟨*layername*⟩→ ⟨*tag*⟩?⟨*name*⟩⟨*metadata*⟩?

⟨*tag*⟩→ '<'⟨*html_tag*⟩'>'

⟨*html_tag*⟩→ div | area | ...

⟨*name*⟩→ ['A'-'z']{1}['A'-'z','0'-'9',' _','-']*

⟨*metadata*⟩→ [⟨*multiple_values*⟩] | [⟨*value_format*⟩]

⟨*multiple_values*⟩→ ⟨*multiple_values*⟩','⟨*value_format*⟩ | ⟨*value_format*⟩

⟨*value_format*⟩→ ⟨*ns_key*⟩'='⟨*value*⟩

⟨*value*⟩→ '(.*)'

⟨*ns_key*⟩→ ⟨*ns_key_format*⟩':'⟨*ns_key_format*⟩

⟨*ns_key_format*⟩→ ['a'-'z','_']?

Appendix C

Source code parser

C.1 Main.java

```
1  import firebolt.*;
2
3  /**
4   *
5   * Main class (entry point Firebolt)
6   *
7   * @author Alex Jeensma
8   */
9  public class Main {
10   /**
11    * Takes a .psd file as input and parses it
12    *
13    * @param args
14    */
15   public static void main(String[] args) {
16
17     long start = System.currentTimeMillis();
18
19     if(args.length <= 1) {
20       System.out.println("Usage: firebolt path/to/design.psd /path/to/
             output_folder");
21       System.exit(0);
22     }
23
24     try
25     {
26       Parser p = new Parser(args[0], args[1]);
27       long end = System.currentTimeMillis();
28       System.out.println("Parsed Photoshop file in " + (end - start) + " ms..\
             nOutput is in " + p.getOutput());
29       System.exit(0);
30     }
31     catch(Exception e) {
32       System.out.println("Something went wrong: " + e.getMessage());
33       e.printStackTrace();
```

```
34      }
35    }
36  }
```

C.2 CommonPractice.java

```
1  package firebolt;
2
3  import java.util.LinkedList;
4
5  import firebolt.commonpractices.*;
6
7  /**
8   * The CommponPractice abstract class allows us to easily add some common
        practices to our project
9   * Some of them are developed over time by seeing patterns in the HTML
10  *
11  * @author Alex Jeensma
12  */
13 public abstract class CommonPractice {
14   /**
15    * Changes a element based on the different properties by
16    * using a Common Practice..
17    *
18    * @param e
19    */
20   public abstract void changeElement(
21       Element e
22   );
23
24   /**
25    * Returns a Class list of all heuristics in the firebolt.commonpractices
        package
26    * @return A list of classes that implement this abstract CommonPractice class
27    */
28   @SuppressWarnings("unchecked")
29   public static LinkedList<Class<CommonPractice>> getHeuristics()
30     {
31     String heuristics[] = {"TestCommonPractice"};
32     LinkedList<Class<CommonPractice>> classes = new LinkedList<Class<
           CommonPractice>>();
33
34     try
35     {
36       for(int i = 0; heuristics.length > i; i++)
37       {
38         classes.add((Class<CommonPractice>)Class.forName("firebolt.commonpractices
               ." + heuristics[i]));
39       }
40     }
41     catch(ClassNotFoundException cnfe)
42     {
43       System.out.println("Error loading a Heuristic: " + cnfe.getMessage());
44     }
```

```
45
46        return classes;
47      }
48
49  }
```

C.3 Configuration.java

```
1   package firebolt;
2
3   import java.util.HashMap;
4
5   /**
6    * Configuration object
7    *
8    * @author Alex Jeensma
9    */
10  public class Configuration {
11    /**
12     * Key => value, all the configuration options
13     */
14    private HashMap<String,String> options = new HashMap<String,String>();
15
16    /**
17     * Adds a configuration option
18     *
19     * @param key key of the configuration
20     * @param value value of the configuration
21     */
22    public void add(String key, String value)
23    {
24      options.put(key, value);
25    }
26
27    /**
28     * Get a key if it exists, otherwise returns null
29     *
30     * @param key key to find
31     *
32     * @return value if found
33     */
34    public String get(String key)
35    {
36      return options.get(key);
37    }
38  }
```

C.4 Element.java

```
1  package firebolt;
2
3  import java.awt.Color;
4  import java.awt.image.BufferedImage;
```

```
5  import java.io.File;
6  import java.util.*;
7  import java.util.Map.Entry;
8
9  import javax.imageio.ImageIO;
10
11 import psd.model.Layer;
12 import psd.parser.layer.LayerType;
13
14 import firebolt.css.Stylesheet;
15 import firebolt.css.StylesheetSelector;
16 import firebolt.metadata.MetadataParser;
17 import firebolt.exceptions.*;
18
19 /**
20  * The Element class basically holds every HTML element
21  *
22  * @author Alex Jeensma
23  */
24 public class Element
25 {
26   /**
27    * Holds the tag name
28    */
29   private String tag;
30
31   /**
32    * Holds the so called void elements
33    */
34   public final String[] void_elements = {"area","base","br","col","command","embed
         ","hr","img","input","keygen","link","meta","param","source","track","wbr"};
35
36   /**
37    * Layer coupled with this Element
38    */
39   private Layer layer;
40
41   /**
42    * Holds the attributes of this element
43    */
44   private HashMap<String,String> attributes = new HashMap<String,String>();
45
46   /**
47    * Holds the CSS properties for this element
48    */
49   private StylesheetSelector css;
50
51   /**
52    * Parent (null if no parent)
53    */
54   private Element parent;
55
56   /**
57    * Children of this layer
```

```
58     */
59    private LinkedList<Element> children;
60
61    /**
62     * Siblings of the element, index == flow order
63     */
64    private LinkedList<Element> siblings;
65
66    /**
67     * Create a new element
68     *
69     * @param tagname
70     * @param id
71     * @param attributes
72     */
73    public Element(String tagname, String id, HashMap<String,String>attr, Layer l)
74    {
75      children = new LinkedList<Element>();
76      siblings = new LinkedList<Element>();
77      tag = tagname;
78      layer = l;
79      if(id.length() > 0) {
80        attributes.put("id", id);
81        css = new StylesheetSelector("#" + id);
82      }
83      else {
84        css = new StylesheetSelector(tagname);
85      }
86      new MetadataParser(this, attr);
87    }
88
89    /**
90     * Recur all the children + this node and build the style
91     */
92    public void recursiveBuildStyle()
93    {
94      buildStyle();
95
96      for(Element e : children)
97      {
98        e.recursiveBuildStyle();
99      }
100   }
101
102   /**
103    * Build the style of the element
104    *
105    * @TODO Do some complex calculations based on the layer
106    */
107   public void buildStyle()
108   {
109     try
110     {
111       // get the image from the layer
```

```
112         BufferedImage bi = layer.getImage();
113 
114         if(bi == null) {
115           throw new ParseException("Could't get the image from the layer, is it
                named correctly?");
116         }
117 
118         // image dimensions
119         int layerWidth = bi.getWidth();
120         int layerHeight = bi.getHeight();
121 
122         /**
123          *  BACKGROUND OF THE ELEMENT
124          *  - Solid color
125          *  - Gradient
126          *  - Image
127          */
128 
129         Set<Integer> colors = new HashSet<Integer>();
130 
131           for(int y = 0; y < layerHeight; y++)
132           {
133               for(int x = 0; x < layerWidth; x++)
134               {
135                   int pixel = bi.getRGB(x, y);
136                   colors.add(pixel);
137               }
138           }
139 
140           if(colors.size() == 1) {
141             // solid color
142           Color c = new Color(bi.getRGB(0, 0));
143           css.addProperty("background", "rgb("+c.getRed()+","+c.getGreen()+","+c.
                getBlue()+")");
144           }
145           else
146           {
147             // more than one color
148 
149             /** @TODO check for tileable.. */
150 
151             // write image to file and set reference
152             String imagePath = "images/bg_" + attributes.get("id")  + ".png";
153 
154             File file = new File(Parser.getOutput() + imagePath);
155             file.createNewFile();
156 
157             ImageIO.write(bi, "png", file);
158 
159             css.addProperty("background", "url('"+imagePath+"')");
160           }
161 
162         /**
163          * DIMENSIONS OF THE ELEMENT
```

```
164         * - Width
165         * - Height
166         */
167
168        if(!tag.equals("body")) {
169          css.addProperty("width", layerWidth + "px");
170          css.addProperty("height", layerHeight + "px");
171        }
172
173        /**
174         * BOX MODEL OF THE ELEMENT
175         * - Margin
176         * - Padding
177         * - Border
178         */
179        if(hasParent() && parent.getLayer().getType() == LayerType.NORMAL) {
180          Layer p = parent.getLayer();
181
182          // parent x position
183          int xp = p.getX();
184          // parent x position right
185          int xpr = p.getWidth() + xp;
186          // parent y position
187          int yp = p.getY();
188          // parent y position bottom
189          int ypb = p.getHeight() + yp;
190
191          // x position
192          int x = layer.getX();
193          // x right position
194          int xr = layer.getWidth() + x;
195          // y position
196          int y = layer.getY();
197          // y bottom position
198          int yb = layer.getHeight() + y;
199
200          // check for center align in this parent element
201          if((xr - xpr) == (xp - x)) {
202            css.addProperty("margin-left", "auto");
203            css.addProperty("margin-right", "auto");
204          }
205
206          // calculate next and previous in flow elements
207          Element nif = getNextInFlow();
208          Element pif = getPrevInFlow();
209
210          StylesheetSelector parentSelector = parent.getSelector();
211
212          if((y - yp) > 0 && pif == null) {
213            // @TODO: according to the box model, the height should decrease -
                 padding
214            parentSelector.addProperty("padding-top", (y - yp) + "px");
215            if(parentSelector.getProperty("height") != null) {
```

```
216             int newHeight = Stylesheet.pixelsToInt(parentSelector.getProperty("
                    height")) - (y - yp);
217             parentSelector.addProperty("height", newHeight + "px");
218           }
219         }
220
221         if(pif == null) {
222           // check for the left margin of this element (judging by it's parent)
223           int xDifference = x - xp;
224           if(xDifference > 0) {
225             if(css.getProperty("margin-left") != "auto")
226             {
227               css.addProperty("margin-left", xDifference + "px");
228             }
229           }
230         }
231
232         // check for margin top of this element
233         if(nif != null) {
234           Layer nifL = nif.getLayer();
235           if((nifL.getY() - yb) > 0) {
236             nif.getSelector().addProperty("margin-top", (nifL.getY() - yb) + "px")
                    ;
237           }
238         }
239
240         // calculate floats (http://www.w3schools.com/cssref/pr_class_float.asp)
241         // a float occurs when this element has a sibling are on the same "level"
242         if(nif != null || pif != null) {
243           if(nif != null)
244           {
245             int nif_x = nif.getLayer().getX();
246
247             // get the ranges for the Y axis of the next in flow element..
248             int nif_y = nif.getLayer().getY();
249
250             if(nif_y == y) {
251               // @TODO: more complex comparison for parallel Y elements..
252               css.addProperty("float", "left");
253               nif.getSelector().addProperty("float", "left");
254
255               if((nif_x - xr) > 0) {
256                 css.addProperty("margin-right", (nif_x - xr) + "px");
257               }
258             }
259           }
260           if(pif != null)
261           {
262
263           }
264         }
265
266       }
267
```

```
268       for(Class<CommonPractice> heur : CommonPractice.getHeuristics())
269       {
270         try
271         {
272           CommonPractice heuristic = heur.newInstance();
273           heuristic.changeElement(this);
274         }
275         catch (InstantiationException e)
276         {
277           System.out.println("Couldn't initiate Heuristic: " + e.getMessage());
278         }
279         catch (IllegalAccessException e)
280         {
281           System.out.println("Couldn't access Heuristic: " + e.getMessage());
282         }
283       }
284     }
285     catch(ParseException pe)
286     {
287       System.err.println(pe.getMessage());
288     }
289     catch(Exception e)
290     {
291       System.err.println("Exception in buildStyle(): " + e.getMessage());
292     }
293   }
294
295   /**
296    * Gets the next element in the flow
297    *
298    * @see http://explainth.at/en/css/flow.shtml
299    * @return element next in flow, otherwise null
300    */
301   public Element getNextInFlow()
302   {
303     int thisIndex = siblings.indexOf(this);
304
305     if(thisIndex == 0) {
306       // this is the first element..
307       return null;
308     }
309     else {
310       return siblings.get(thisIndex - 1);
311     }
312   }
313
314   /**
315    * Gets the prev element in the flow
316    *
317    * @see http://explainth.at/en/css/flow.shtml
318    * @return element prev in flow, otherwise null
319    */
320   public Element getPrevInFlow()
321   {
```

```
322     int thisIndex = siblings.indexOf(this);
323     int siblingsSize = siblings.size();
324 
325     if((thisIndex + 1) == siblingsSize || siblingsSize == 1) {
326       // this is the last element, so it has no next
327       return null;
328     }
329     else {
330       // there must be a next
331       return siblings.get(thisIndex + 1);
332     }
333   }
334 
335   /**
336    * Merges 2 elements together, if you have this structure
337    *
338    * F <body>
339    *   L Layer
340    *   L Layer2
341    *   L <body>
342    *
343    * Folder <body> and Layer <body> will get merged into one element
344    *
345    * So the layer <body>, will become the group <body>!
346    *
347    * @param child the child
348    */
349   public void merge(Element child)
350   {
351     // this = top folder (parent)
352     // child = same named element as top folder, new element
353     if(child.hasParent()) {
354       // get the parent, of the child, and remove it from the children
355       child.getParent().removeChild(child);
356     }
357 
358     // copy settings from the layer element to the group element
359 
360     // attributes
361     attributes.putAll(child.getAttributes());
362 
363     // layer information
364     layer = child.getLayer();
365 
366 
367   }
368 
369   /**
370    * Adds a child
371    *
372    * @param e the child
373    */
374   public void addChild(Element e)
375   {
```

```
376     children.add(e);
377     e.setParent(this);
378   }
379
380   /**
381    * Returns the tag + it's attributes
382    */
383   public String toString()
384   {
385     String elem = "<"+tag+buildAttributes()+">";
386     return elem;
387   }
388
389   /**
390    * Build the attributes string
391    *
392    * @return attributes as string
393    */
394   private String buildAttributes()
395   {
396     String attributesString = "";
397     Iterator<Entry<String, String>> entries = attributes.entrySet().iterator();
398     while (entries.hasNext()) {
399       Map.Entry<String,String> entry = (Map.Entry<String,String>)entries.next();
400         String attr = (String)entry.getKey();
401         String value = (String)entry.getValue();
402         attributesString += " " + attr + "=\"" + value + "\"";
403     }
404     return attributesString;
405   }
406
407   /**
408    * Build the append string (replace tag name, tag id, etc)
409    *
410    * @param append the append string
411    *
412    * @return parsed string
413    */
414   private String buildAppend(String append)
415   {
416     String id = "";
417
418     if(attributes.get("id") != null) {
419       id = "#" + attributes.get("id");
420     }
421     else if(attributes.get("class") != null) {
422       id = "." + attributes.get("class");
423     }
424
425     append = append.replace("tag_name", tag);
426     append = append.replace("tag_id", id);
427
428     if(id.length() == 0) {
429       // no length? no id comment string
```

```
430       append = "";
431     }
432
433     return append;
434   }
435
436   /**
437    * Print the element and it's children recursively
438    *
439    * @TODO Make a String repeat method
440    *
441    * @return parsed HTML
442    */
443   public String printRecursive(int level, String appendFirst, String appendLast)
444   {
445     if(tag.length() == 0) {
446       return "";
447     }
448
449     String HTML = "<"+tag;
450     String indentation = "";
451
452     HTML += buildAttributes();
453
454     HTML += ">"+buildAppend(appendFirst)+"\n";
455
456     level++;
457     // Element e : children
458     for(int y = children.size() - 1; y >= 0; y--) {
459       indentation = "";
460       for(int i = 0; level > i; i++) {
461         indentation += "\t";
462       }
463       HTML += indentation + children.get(y).printRecursive(level, appendFirst,
              appendLast);
464     }
465     level--;
466
467     indentation = "";
468     for(int i = 0; level > i; i++) {
469       indentation += "\t";
470     }
471
472     String end_tag = "";
473
474     if(!Arrays.asList(void_elements).contains(tag)) {
475       end_tag = tag;
476     }
477
478     HTML += indentation + "</"+end_tag+">"+buildAppend(appendLast)+"\n";
479
480     return HTML;
481   }
482
```

```
483   /**
484    * Adds a sibling to this element
485    *
486    * @param e the sibling
487    */
488   public void addSibling(Element e)
489   {
490     siblings.add(e);
491   }
492
493   /**
494    * Adds a attribute to the element
495    *
496    * @param attr attribute name
497    * @param value value of the attribute
498    */
499   public void addAttribute(String attr, String value)
500   {
501     attributes.put(attr, value);
502   }
503
504   /**
505    * Remove the Element c from the children collection
506    *
507    * @param c the child to remove
508    */
509   public void removeChild(Element c)
510   {
511     children.remove(c);
512   }
513
514   /**
515    *  GETTERS AND SETTERS
516    */
517
518   /**
519    * Get the parent
520    *
521    * @return parent element
522    */
523   public Element getParent()
524   {
525     return parent;
526   }
527
528   /**
529    * Set a parent
530    *
531    * @param e the parent
532    */
533   public void setParent(Element e)
534   {
535     parent = e;
536   }
```

```
537
538   /**
539    * Does this element have a parent?
540    *
541    * @return true on parent
542    */
543   public boolean hasParent()
544   {
545     return parent != null;
546   }
547
548   /**
549    * Get the layer
550    *
551    * @return layer
552    */
553   public Layer getLayer()
554   {
555     return layer;
556   }
557
558   /**
559    * Get the attributes for this element
560    *
561    * @return attributes
562    */
563   public HashMap<String,String> getAttributes()
564   {
565     return attributes;
566   }
567
568   /**
569    * Get the children of this Element
570    *
571    * @return children
572    */
573   public LinkedList<Element> getChildren()
574   {
575     return children;
576   }
577
578   /**
579    * Get the tag name
580    *
581    * @return tagstring
582    */
583   public String getTag()
584   {
585     return tag;
586   }
587
588   /**
589    * Get the stylesheet selector
590    *
```

```
591     * @return style for this element
592     */
593   public StylesheetSelector getSelector()
594   {
595     return css;
596   }
597 }
```

C.5 LayerMatcher.java

```
1  package firebolt;
2
3  import java.util.HashMap;
4  import java.util.StringTokenizer;
5  import java.util.regex.Matcher;
6  import java.util.regex.Pattern;
7
8  import psd.model.Layer;
9
10 /**
11  * This class disects a layername into it's different components
12  * @author Alex Jeensma
13  */
14 public class LayerMatcher
15 {
16   /**
17    * Optional tag name
18    */
19   private String tag = "";
20
21   /**
22    * Optional ID
23    */
24   private String id = "";
25
26   /**
27    * Optional attributes
28    */
29   private HashMap<String,String> attributes = new HashMap<String,String>();
30
31   /**
32    * Create a new layer and disect it into it's components
33    *
34    * @param layer the layer to dissect
35    */
36   public LayerMatcher(Layer layer)
37   {
38     // match 1 = tagname
39     // match 2 = id
40     // match 3 = attributes
41     String layerRegex = "^(<[a-z^>]*>)?([A-z\\-\\_]*)?(\\[.*\\])?$";
42     String layerName = layer.toString();
43
44     Pattern p = Pattern.compile(layerRegex);
```

```
45      Matcher m = p.matcher(layerName);
46
47      if(m.matches()) {
48        // get the tagname (if its there, otherwise default to div)
49        tag = m.group(1) != null ? m.group(1).substring(1, m.group(1).length() - 1)
              : "div";
50        // get the identification (if its there, otherwise leave it empty)
51        id = m.group(2) != null ? m.group(2) : "";
52        // check for attributes
53        if(m.group(3) != null) {
54          // has attributes
55          StringTokenizer st = new StringTokenizer(m.group(3).substring(1, m.group
                (3).length() - 1), ",");
56          StringTokenizer attr;
57          while(st.hasMoreTokens()) {
58            attr = new StringTokenizer(st.nextToken(), "=");
59            while(attr.hasMoreTokens()) {
60              String key = attr.nextToken();
61              String value = attr.nextToken();
62              attributes.put(key, value.substring(1, value.length() - 1));
63            }
64          }
65        }
66      }
67      else {
68        // verdorie nog aan toe
69        System.err.println("Couldn't parse layer: " + layerName);
70      }
71    }
72
73    /**
74     * Get the tagname
75     *
76     * @return the tag name
77     */
78    public String getTag()
79    {
80      return tag;
81    }
82
83    /**
84     * Get the ID
85     *
86     * @return the ID
87     */
88    public String getID()
89    {
90      return id;
91    }
92
93    /**
94     * Get the attributes map
95     *
96     * @return the attributes
```

```
97     */
98    public HashMap<String,String> getAttributes()
99    {
100     return attributes;
101   }
102 }
```

C.6 Parser.java

```
1  package firebolt;
2
3  import java.io.*;
4  import java.util.HashMap;
5  import java.util.LinkedList;
6  import java.util.StringTokenizer;
7  import java.util.regex.*;
8  import psd.model.*;
9  import psd.parser.layer.LayerType;
10
11 /**
12  * Represents the main object that reads the file, does the parsing and generates
        the HTML
13  *
14  * @author Alex Jeensma
15  */
16 public class Parser {
17
18   /**
19    * File object representing the Photoshop file
20    */
21   private File psd_file;
22
23   /**
24    * The folder to the output
25    */
26   private static String output;
27
28   /**
29    * Holds the document
30    */
31   private Document document;
32
33   /**
34    * Constructor, sets up the parser and calls necessary methods
35    *
36    * @param path path to the psd file that adheres to the specification
37    * @throws Exception
38    */
39   public Parser(String path, String outputPath) throws Exception
40   {
41     psd_file = new File(path);
42
43
44     // check for trailing slash
```

```
45      if(outputPath.charAt(outputPath.length() - 1) != '/') {
46        outputPath += "/";
47      }
48
49      output = outputPath;
50
51      if(!psd_file.exists()) {
52        throw new Exception("Read error: File '" + path + "' doesn't exist!");
53      }
54
55      document = new Document();
56
57      // parse the PS file
58      Psd parsedPSD = new Psd(psd_file);
59
60      // element top <body>
61      Element body = null;
62
63      // top element == <body>
64      body = recursiveParse(parsedPSD.getLayer(0), null);
65      document.setBody(body);
66
67      // reset body
68      body.getSelector().addProperty("margin", "0px").addProperty("padding", "0px");
69
70      // second element == configuration (optional)
71      if(parsedPSD.getLayersCount() > 1 && parsedPSD.getLayer(1).toString().equals("
            configuration")) {
72        Configuration config = readConfig(parsedPSD.getLayer(1));
73        document.setConfiguration(config);
74      }
75      else {
76        document.setConfiguration(new Configuration());
77      }
78
79      body.recursiveBuildStyle();
80
81      BufferedWriter html = new BufferedWriter(new FileWriter(output + "index.html")
            );
82
83      html.write(document.print());
84
85      html.close();
86    }
87
88    /**
89     * Recursive parser, builds a ElementTree
90     *
91     * @param l
92     * @return
93     * @throws Exception
94     */
95    private Element recursiveParse(Layer l, Element currentParent) throws Exception
96    {
```

```
97      // parent of current layer
98      Element e = parseLayer(l);
99      LinkedList<Element> children = new LinkedList<Element>();
100
101     document.addCSSLine(e.getSelector());
102     for(int x = 0; x < l.getLayersCount(); x++)
103     {
104       // traverse children
105       Layer cL = l.getLayer(x);
106
107       // ignore layer groups (nesting prob)
108       if(!cL.toString().equals("</Layer group>"))
109       {
110         Element c = recursiveParse(cL, e);
111
112         e.addChild(c);
113
114         LayerMatcher lm1 = new LayerMatcher(l);
115         LayerMatcher lm2 = new LayerMatcher(cL);
116
117         String dummy1ID = lm1.getID();
118         String dummy2ID = lm2.getID();
119
120         if(l.getType() == LayerType.FOLDER && cL.getType() == LayerType.NORMAL &&
              dummy1ID.equals(dummy2ID)) {
121           e.merge(c);
122         }
123         else {
124           children.add(c);
125         }
126       }
127     }
128
129     // iterate the siblings
130     for(Element parentChild : children)
131     {
132       for(Element childChild : children)
133       {
134         parentChild.addSibling(childChild);
135       }
136     }
137
138     return e;
139   }
140
141   /**
142    * Read the configuration group
143    *
144    * @param layer
145    * @return
146    * @throws Exception
147    */
148   private Configuration readConfig(Layer group) throws Exception
149   {
```

```
150     Configuration c = new Configuration();
151
152     for(int x = 0; group.getLayersCount() > x; x++)
153     {
154       String cfgStr = group.getLayer(x).toString();
155       String[] parts = cfgStr.split("=");
156
157       c.add(parts[0], parts[1].substring(1, parts[1].length() - 1));
158     }
159
160     return c;
161   }
162
163   /**
164    * Parses a layer into a element
165    *
166    * @param layerName
167    * @throws Exception
168    */
169   private Element parseLayer(Layer layer) throws Exception
170   {
171     // parse
172     LayerMatcher lm = new LayerMatcher(layer);
173     // parsing done, create the Element
174     return new Element(lm.getTag(), lm.getID(), lm.getAttributes(), layer);
175   }
176
177   /**
178    * Gets the output folder
179    *
180    * @return the output folder including a trailing slash
181    */
182   public static String getOutput()
183   {
184     return output;
185   }
186
187 }
```

C.7 TestCommonPractice.java

```
1  package firebolt.commonpractices;
2
3  import firebolt.*;
4
5  /**
6   * This Common Practice does nothing
7   * @author Alex Jeensma
8   */
9  public class TestCommonPractice extends CommonPractice {
10   /**
11    * As said before, does absolutely nothing..
12    */
13   @Override
```

```
14   public void changeElement(Element e) { }
15 }
```

C.8 Stylesheet.java

```
1  package firebolt.css;
2
3  import java.util.*;
4
5  /**
6   * Represents the Stylesheet of the Document. Every element basically has it's own
7   * StyleSheetSelector, when the Photoshop file is parsed, all the
        StyleSheetSelectors
8   * are accumulated and build up to a valid CSS file
9   *
10  * @author Alex Jeensma
11  */
12 public class Stylesheet {
13   /**
14    * List of all the selectors (in order)
15    */
16   private LinkedList<StylesheetSelector> selectors = new LinkedList<
        StylesheetSelector>();
17
18   /**
19    * Adds a new selector to the end of the selectors list
20    *
21    * @param s the full selector
22    * @return the created StylesheetSelector
23    */
24   public void addSelector(StylesheetSelector s)
25   {
26     selectors.add(s);
27   }
28
29   /**
30    * Print the Stylesheet
31    *
32    * @return printed string
33    */
34   public String printRecursive()
35   {
36     String CSSString = "";
37
38     for(StylesheetSelector s : selectors)
39     {
40       CSSString += s.print();
41     }
42
43     return CSSString;
44   }
45
46   /**
47    * STATIC HELPERS
```

```
48     */
49
50    /**
51     * Changes a pixels based measurement to it's integer value
52     *
53     * @param px the dimension in px in a String
54     *
55     * @return the size in integer
56     */
57    public static int pixelsToInt(String px)
58    {
59      return Integer.parseInt(px.substring(0, px.length( ) - 2));
60    }
61 }
```

C.9 StylesheetSelector.java

```
1  package firebolt.css;
2
3  import java.util.*;
4  import java.util.Map.Entry;
5
6  /**
7   * @author Alex Jeensma
8   */
9  public class StylesheetSelector {
10
11    /**
12     * Holds the FULL selector
13     */
14    private String selector;
15
16    /**
17     * Holds the CSS properties for this selector
18     */
19    private HashMap<String,String> properties;
20
21    /**
22     * Creates a new selector
23     *
24     * @param s the full selector
25     */
26    public StylesheetSelector(String s)
27    {
28      selector = s;
29      properties = new HashMap<String,String>();
30    }
31
32    /**
33     * Print the stylesheet selector and it's properties
34     *
35     * @return
36     */
37    public String print()
```

```
38   {
39     // no selector or no properties? no need for selector
40     if(selector.length() == 0 || properties.size() == 0) { return ""; }
41
42     String CSS = "\t" + selector + " {\n";
43
44     Iterator<Entry<String, String>> entries = properties.entrySet().iterator();
45
46     while (entries.hasNext()) {
47       Map.Entry<String,String> entry = (Map.Entry<String,String>)entries.next();
48         String key = (String)entry.getKey();
49         String value = (String)entry.getValue();
50         CSS += "\t\t"+key+": "+value+";\n";
51     }
52
53     return CSS + "\n\t}\n\n";
54   }
55
56   /**
57    * Get a certain property
58    *
59    * @param property property to get
60    * @return property or null
61    */
62   public String getProperty(String property)
63   {
64     return properties.get(property);
65   }
66
67   /**
68    * Add a CSS property
69    *
70    * @param property property of the CSS line
71    * @param value value of the CSS line
72    *
73    * @return this stylesheetselector for chaining
74    */
75   public StylesheetSelector addProperty(String property, String value)
76   {
77     properties.put(property, value);
78     return this;
79   }
80 }
```

C.10 ParseException.java

```
1 package firebolt.exceptions;
2
3 /**
4  * When something goes wrong parsing the document
5  *
6  * @author Alex Jeensma
7  */
8
```

```
@SuppressWarnings("serial")
public class ParseException extends Exception
{
  /**
   * Calls the super constructor
   *
   * @param string parse error
   */
  public ParseException(String string) {
    super("Firebolt parse error: " + string);
  }
}
```

C.11 Attribute.java

```
package firebolt.metadata;

import firebolt.*;

/**
 * The magic attribute class
 * @author Alex Jeensma
 */
public class Attribute extends Metadata {
  /**
   * Parse the attribute
   */
  public void parse() {
    element.addAttribute(key,value);
  }
}
```

C.12 CSS.java

```
package firebolt.metadata;

import firebolt.css.*;

/**
 * CSS class (just adds a selector directly)
 *
 * @author Alex Jeensma
 */
public class CSS extends Metadata
{
  /**
   * Add the style to the end of the StylesheetSelector
   */
  public void parse()
  {
    StylesheetSelector s = element.getSelector();
    s.addProperty(key, value);
  }
```

```
}
```

C.13 Metadata.java

```
package firebolt.metadata;

import firebolt.Element;

/**
 * Holds the basic Metadata class
 *
 * @author Alex Jeensma
 *
 */
public abstract class Metadata {
  /**
   * Holds the element for this Metadata rule
   */
  protected Element element;

  /**
   * Holds the key without the Metadata prepended
   */
  protected String key;

  /**
   * Holds the value (without the quotes)
   */
  protected String value;

  /**
   * Creates a new metadata line
   *
   * @param e the element for this line
   * @param k key string
   * @param v value string
   */
  public void setMetadata(Element e, String k, String v)
  {
    element = e;
    key = k;
    value = v;
  }

  /**
   * Must be implemented by the child class
   */
  public abstract void parse();
}
```

C.14 MetadataParser.java

```java
package firebolt.metadata;

import firebolt.Element;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;
import java.util.StringTokenizer;
import java.util.Map.Entry;

/**
 * @author Alex Jeensma
 */
public class MetadataParser {
  /**
   * Constant hashmap that holds the metadata objects
   * Maps a short string like attr to a object
   */
    public final static HashMap<String,Metadata> metaDataObjects = new HashMap<
        String,Metadata>();
    static {
      metaDataObjects.put("attr", new Attribute());
      metaDataObjects.put("css", new CSS());
  }

  /**
   * Parse the metadata given with an element
   *
   * @param el
   * @param attributes
   */
  public MetadataParser(Element el, HashMap<String,String> attributes)
  {
    Iterator<Entry<String, String>> entries = attributes.entrySet().iterator();

    try
    {
      while (entries.hasNext()) {
        Map.Entry<String,String> entry = (Map.Entry<String,String>)entries.next();
          String key = (String)entry.getKey();
          String value = (String)entry.getValue();

          StringTokenizer parsedKey = new StringTokenizer(key, ":");

          while(parsedKey.hasMoreTokens()) {
            String namespace = parsedKey.nextToken();
            String namespace_value = parsedKey.nextToken();

            Metadata m = metaDataObjects.get(namespace);
            if(m == null) {
              throw new Exception("Namespace " + namespace + " not found..");
            }
            m.setMetadata(el, namespace_value, value);
            m.parse();
          }
```

```
54
55          }
56      }
57      catch(Exception e) {
58          System.out.println(e.getMessage());
59      }
60    }
61
62 }
```

Bibliography

[1] The Photoshop File Format http://www.adobe.com/devnet-apps/photoshop/fileformatashtml/PhotoshopFileFormats.htm#50577409_pgfId-1030196

[2] Imageready http://en.wikipedia.org/wiki/Adobe_ImageReady

[3] Fireworks vs Photoshop Compression http://webdesignerwall.com/general/fireworks-vs-photoshop-compression

[4] "Why Tables Are Bad (For Layout*) Compared to Semantic HTML + CSS" http://phrogz.net/css/WhyTablesAreBadForLayout.html

[5] Formats that ImageMagick supports http://www.imagemagick.org/script/formats.php

[6] jQuery JavaScript library http://jquery.com/

[7] Void tags in HTML http://www.whatwg.org/specs/web-apps/current-work/multipage/syntax.html#void-elements

[8] Data attributes in HTML5 http://www.w3.org/html/wg/html5/#custom

[9] Doctypes in HTML http://en.wikipedia.org/wiki/Document_Type_Declaration

[10] Element flow in HTML http://webdesign.about.com/od/cssglossary/g/bldefnormalflow.htm

[11] Sliding Doors in CSS http://www.alistapart.com/articles/slidingdoors/

[12] Conditional comments Internet Explorer http://www.quirksmode.org/css/condcom.html