INSTANT

Short | Fast | Focused

HTML5 Fonts and Typography

Make your websites stand out by using custom fonts with splendid effects

K. Jaouher

[PACKT] PUBLISHING

Table of Contents

Instant HTML5 Fonts and Typography

Instant HTML5 Fonts and Typography

First published: August 2013

Production Reference: 1230813

Published by Packt Publishing Ltd.

Livery Place

35 Livery Street

Birmingham B3 2PB, UK.

ISBN 978-1-84969-786-6

www.packtpub.com

Credits

Author

K. Jaouher

Reviewer

Christopher L Martin

Acquisition Editor

Martin Bell

Commissioning Editor

Shaon Basu

Technical Editor

Krutika Parab

Project Coordinator

Esha Thakker

Proofreader

Lindsey Thomas

Production Coordinator

Nitesh Thakur

Cover Work

Nitesh Thakur

Cover Image

Valentina Dsilva

About the Author

K. Jaouher is a web developer from Tunisia and has a degree in Software Engineer in 2007. Ever since, his objective was to improve his skills in multiple domains especially in web and industrial development. When he is not working on client projects he is often creating some new articles in his blog. He is also a mentor in different training centers in his country.

He, his wife, and his little baby girl make their home in Tunis city. He likes video games and music.

You can follow Jaouher on his blog at https://sites.google.com/site/webdevetipstuto or on his about me page at http://about.me/thabigboss/

I wish to thank my wife, my love Amal for backing me up and being patient whenever I was not there. Thanks to my sweet little girl Salma, my angel, and my good luck charm.

I dedicate this work to my father Kamel and my mother Samira. I could not be the person I am without your continuous backup. Walid, Ghassen, and Raed, I love you too. Special thanks to my in-laws for their encouragements and continuous support.

Thanks also to the folks at Packt Publishing, the technical reviewers, and those who provided help and inspiration for giving me the opportunity to achieve this dream.

About the Reviewer

Christopher L Martin believes in a more thoughtful approach to design which promotes human interaction and engagement; whether it's on the web or in print, it should be memorable, but not intrusive. It should develop a meaningful connection with the audience no matter how small the project is.

Christopher has been designing and developing websites and web applications for over five years. In that time he's helped bring countless projects to fruition for both large and small companies and he can't wait to see what tools the web will have to offer designers and developers as it grows and evolves into the future.

Preface

Typography is a composition of the Greek words: τύπος (typos) which means form and γραφή (graphe) which means writing. It is the art and technique of arranging type in order to make language visible. This art is very ancient and was used for multiple purposes such as literature, advertising, newspapers, and mathematics. Surprisingly, even the web is 95 percent typographic but it took some years for web developers and web designers to adopt the idea. Now, with the invention of HTML5, CSS, and JavaScript, new tags and properties; creating artistic images became fun and easy. It is also important to take care of older versions by ensuring cross-browser compatibility via a lot of techniques that will be discussed in this book. You can already create your own font and use it on your website and even import some premade fonts from a web font host.

What this book covers

Use of custom fonts with @font-face (Simple) will guide you on how to use the most fundamental method to incorporate locally stored fonts, how to manage the cross-browser compatibility by multiple font formats, and finally the method to find an issue if an error occurs while loading the font via web-safe fonts.

Creating different font files (Intermediate) will explain how to generate our own font and other cross-browser font files. In this recipe, we will adopt the simplest and most common method to create the necessary files. We will even discover how to create text inside modern browsers' container canvas.

Using Web fonts (Simple) will cover web hosted fonts. We will take an example of Google fonts and explain the different methods of using them. We will also learn how to use the open JavaScript library, WebFont loader, for multiple web font hosts.

Enhancing fonts with outstanding effects (Intermediate) will cover how to use some special CSS properties for creating some visual effects and how to import some premade effects hosted on the web font server.

Using Cufón – fonts for the people (Intermediate) will guide you in using a replacement technique via some special libraries, in particular, the JavaScript library Cufón; as not all our clients have a powerful rendering engine on their PCs. This method is efficient especially for larger font sizes due to smoothing leak in older versions of Windows.

Combing Fonts and other HTML5/CSS3 features (Advanced) will teach you how to combine fonts to create a nice rendering. Such results may be an image or a flash video product which are lighter in weight and give a modern look to your website.

What you need for this book

Having basic JavaScript, HTML5, and CSS3 is an asset to better understanding and further advanced development. You may install any commercial editor (Dreamweaver, phpDesigner…) or a free one (Notepad++, Eclipse, Vim…).

Who this book is for

This book is for every web developer who is looking to enhance and personalize his products with custom fonts.

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 can include other contexts through the use of the `include` directive."

A block of code is set as follows:

```
[default]
exten => s,1,Dial(Zap/1|30)
exten => s,2,Voicemail(u100)
exten => s,102,Voicemail(b100)
exten => i,1,Voicemail(s0)
```

When we wish to draw your attention to a particular part of a code block, the relevant lines or items are set in bold:

```
[default]
exten => s,1,Dial(Zap/1|30)
exten => s,2,Voicemail(u100)
exten => s,102,Voicemail(b100)
exten => i,1,Voicemail(s0)
```

Any command-line input or output is written as follows:

```
# cp /usr/src/asterisk-addons/configs/cdr_mysql.conf.sample
     /etc/asterisk/cdr_mysql.conf
```

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: "clicking the **Next** button moves you to the next screen".

Note

Warnings or important notes appear in a box like this.

Tip

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.

Chapter 1. Instant HTML5 Fonts and Typography

Welcome to *Instant HTML5 Fonts and Typography*. In this book, we will cover creating custom fonts and generating cross-browser font file formats, importing fonts via `@font-face`, `@import`, the `link` tag, and JavaScript from both local and distant servers, applying fonts in both a static and dynamic way on normal text or canvas tag, and applying different effects over text using CSS properties or by using predefined effects from distant font servers.

We will also cover specific JavaScript commands to generate fonts to avoid old browser problems such as rendering engines (for example, Cufón) and rendering a static or animated image via fonts, scripts, and other HTML5 and CSS features.

Using custom fonts with @font-face (Simple)

The most fundamental task in manipulating fonts is the use of the `font` command. CSS gives us different methods to use this command so that we can display it in the major browsers. Fonts vary from one language to another. Yet in the same one, for example, Latin letters, multiple variants are possible.

Major browsers offer us some common fonts such as Helvetica and Verdana. However, in order to use custom fonts, we will use the `@font-face` command. The `@font-face` command is not only amazing, but it is also simple to use, easy to implement, and most of all completely cross-browser compatible, and therefore supported even by older versions of Internet Explorer. All of this is done by just a few lines of CSS.

Getting ready

Please refer to the project `InitFont` to get the full source code in addition to the custom fonts that we will be using.

How to do it...

The `InitFont` project takes us through the following steps:

1. First we will initialize the style:

```
<!DOCTYPE html>
<html>
  <head>
    <title>My first @font-face demo</title>
    <style type="text/css">
```

2. Then we will define the font:

```
@font-face {
  font-family: 'jokal';
```

3. Then we will make provisions for loading fonts for multiple browsers:

```
  src: url('Font/jokal.eot?#iefix') format('eot');
  src: local(jokal),
       url('Font/jokal.woff') format('woff'),
       url('Font/jokal.ttf') format('truetype'),
       url('Font/jokal.svg#webfontL9Me1nzs')
format('svg');
}
@font-face {
  font-family: 'pincoyablackblack';
  src: url('Font/pincoyablack-webfont.eot?#iefix');
  src: local(pincoyablackblack),
       url('Font/pincoyablack-webfont.woff')
format('woff'),
       url('Font/pincoyablack-webfont.ttf')
format('truetype'),
       url('Font/pincoyablack-
webfont.svg#pincoyablackblack') format('svg');
}
```

4. Then we will load a font for mathematical purposes:

```
@font-face {
  font-family: 'STIXGeneral';
  src: url('Font/stixgeneral-webfont.eot?#iefix');
  src: local('STIXGeneral'),
       url('Font/stixgeneral-webfont.woff')
format('woff'),
       url('Font/stixgeneral-webfont.ttf')
format('truetype'),
       url('Font/stixgeneral-webfont.svg#webfontZXtFAA5Q')
format('svg');
  font-weight: normal;
  font-style: normal;
}
```

5. Next we will check alternative bold fonts:

```
@font-face {
  font-family: 'STIXGeneral';
  src: url('Font/stixgeneralbol-webfont.eot?#iefix');
  src: local('STIXGeneral'),
       url('Font/stixgeneralbol-webfont.woff')
format('woff'),
       url('Font/stixgeneralbol-webfont.ttf')
format('truetype'),
       url('Font/stixgeneralbol-
webfont.svg#webfontwFpnxWyx') format('svg');
  font-weight: bold;
  font-style: normal;
}
```

6. We will then cover font usage:

```
h1 {
  font-family: 'pincoyablackblack', Helvetica, Arial,sans-
serif;
  font-size: 45px;
}
p {
  font-family: Helvetica, 'jokal', Arial, sans-serif;
  font-size: 18px;
  line-height: 27px;
}
#math {
  font-family: 'STIXGeneral', sans-serif;
}
    </style>
  </head>
  <body>
    <div id="container">
    <h1>Header</h1>
    <p>common text format.</p>
```

7. Finally, we will check for usage of Unicode characters:

```
      <div id="math"> This is a Sample Mathematical font :
      <br />
      &#x424 = &#x222D &#x2016 &#x472 &#x2016 + &#x2211
&#x394 * &1D7D8
      <br/>
      <b> &#x424 = &#x222D &#x2016 &#x472 &#x2016 +
&#x2211 &#x394 * &1D7D8</b>
      </div>
    </div>
  </body>
</html>
```

Tip

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.

How it works...

This recipe takes us through the following aspects:

- **Font definition**: We will be creating the font name that will be used later. You may use an easy to remember name. It is better to keep this name different from other common fonts such as Helvetica and Arial:

  ```
  @font-face {
    font-family: 'jokal';
  }
  ```

- **Loading fonts for multiple browsers**: To cover the majority of browsers we use a specific approach. We can note that we have used two `src` commands.

  The first one is for IE and the other one is for the rest.

  IE will use an EOT file (Embedded Open Type) in the first declaration and will ignore the rest because it is considered an invalid syntax. However, it downloads the whole set of fonts, even those that are not meant for IE use. To fix that, simply use a question mark (`?`) after the `eot` file declaration. It is meant to fool the browsers (versions before IE9); so it will consider the rest of the syntax as a query and it will not load it:

  ```
  src: url('Font/jokal.eot?#iefix');
  ```

  The second `src` declaration will be adapted by the rest of browsers as follows:

  The first one is the `local()` function. This one will prevent loading the font in the browser cache, if it is already installed on the user's computer:

  ```
  src: local(jokal),
  ```

  The whole function associated with the second `src` command will be separated by a comma (`,`). If a font isn't found, other sources will be tried until one is found.

  For modern browsers, we will use WOFF file (Web Open Font Format). WOFF was developed in 2009 and uses the zlib compression. WOFF is compatible with

Firefox 3.6+, Chrome 6.0+, IE 9.0+, Opera 11.10+, and Safari 5.1+:

```
 url('Font/jokal.woff') format('woff'),
```

The next format is the TTF file (True Type Font). This font will be used by Firefox 3.5+, Chrome 4.0+, Opera 10+, and Safari 3.1+:

```
url('Font/jokal.ttf') format('truetype'),
```

Finally, we will use the SVG format (Scalable Vector Graphics), which is a data format based on XML. SVG is usually considered as a vector graphics standard. However, it can also be used in font usage. This format is used within Firefox3.5+, Chrome 4.0+, IE9+, Opera 9+, and Safari 3.2+. We notice the use of the `#pincoyablackblack` in the `url` property of the import, as it is the ID created when generating the SVG file:

```
url('Font/pincoyablack-webfont.svg#pincoyablackblack')
format('svg');
```

- **Loading a font for mathematical purpose**: We will be using the Scientific and Technical Information Exchange (STIX which is available at http://www.stixfonts.org/) font in order to create some lines for electronic representation of scientific documents. We can verify that we have created the same STIXGeneral font twice using `@font-face` with different font loading sources:

```
font-family: 'STIXGeneral';
```

We can distinguish that they are specific to different properties of the same font. The first one is for the normal weight. Here also we used the cross-browser compatibility method by using different formats (EOT, WOFF, TTF, and SVG):

```
src: url('Font/stixgeneral-webfont.eot?#iefix');
src: local('STIXGeneral'),
     url('Font/stixgeneral-webfont.woff') format('woff'),
     url('Font/stixgeneral-webfont.ttf')
format('truetype'),
     url('Font/stixgeneral-webfont.svg#nbfont')
format('svg');
```

- **Alternative bold font**: Now we will insert another font with the same name but in a bold format. Same as the first one, we have established the compatibility with the major browsers. A `font-weight` property was added to enhance the bold effect. By doing this, we will use that font automatically whenever the `bold` tag is implemented within the HTML code:

```
src: url('Font/stixgeneralbol-webfont.eot?#iefix');
src: local('STIXGeneral'),
     url('Font/stixgeneralbol-webfont.woff')
format('woff'),
     url('Font/stixgeneralbol-webfont.ttf')
format('truetype'), url('Font/stixgeneralbol-
webfont.svg#webfontwFpnxWyx') format('svg');
font-weight: bold;
font-style: normal;
```

- **Fonts usage**: To use the font declared, we need to reference it like any common font. The name referenced is the same as the one already defined in the `@font-face` command. It is recommended that we use an alternative web-safe font, so users with really old browsers don't find themselves consulting a web page with a default browser font:

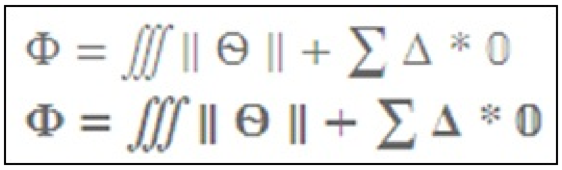
```
h1 {
  font-family: 'pincoyablackblack', Helvetica, Arial,sans-
serif;
  font-size: 45px;
}
p {
  font-family: Futura, 'jokal', Arial, sans-serif;
  font-size: 18px;
  line-height: 27px;
}
#math {
  font-family: 'STIXGeneral', sans-serif;
}
```

The Pincoyablack Black font will be downloaded for pages with the `h1` elements. The Jokal font will be downloaded for the `p` elements, even if the Futura font is available locally. Usual font effects may be applied to our custom font (size, weight, and so on).

- **Usage of the Unicode characters**: For the use of the scientific characters, we will adopt the Unicode characters to identify the characters. For example, consider the following code:

```
&#x424 = &#x222D &#x2016 &#x472 &#x2016 + &#x2211; &#x394
* &#x1D7D8.
```

The preceding code will generate the following in both normal and bold fonts, as shown in the following screenshot:



$\Phi = \iiint \| \Theta \| + \sum \Delta * 0$

$\boldsymbol{\Phi = \iiint \| \Theta \| + \sum \Delta * 0}$

Note

You can get the complete list of font contents and character listing from the following link:

http://www.stixfonts.org/allGlyphs.html

There's more...

Now we will look at the different properties of the `@font-face` rule:

```
@font-face {
  [font-family: <family-name>;]?
  [src: [ <url> [format(<string>#)]? | <font-face-name> ]#;]?
  [unicode-range: <urange>#;]?
  [font-variant: <font-variant>;]?
  [font-feature-settings: normal|<feature-tag-value>#;]?
  [font-stretch: <font-stretch>;]?
  [font-weight: <weight>];
  [font-style: <style>];
}
```

Some of the attributes are listed and defined in the following table:

Attribute	Definition
`<family-name>`	It specifies a font name that will be used as font-face value for font properties.
`<url>`	It provides URL for the remote font file location, or the name of a font on the user's computer in the local form (Font Name).
`<font-variant>`	It defines whether or not a small caps variant will be used. Different values are available: `normal`, `small-caps`, or `inherit`.
`<font-`	This is still an experimental function. It grants control over some

feature-settings>	advanced typographic functions.
<font-stretch>	It defines how the font should be stretched. Here are some of the possible values: normal, condensed, semi-condensed, expanded, extra-expanded, ultra-expanded, and so on.
<font-style>	It defines how the font should be styled. It can be: normal, italic, or bold.
<font-weight>	It defines the boldness of the font. It can be: normal, bold, or [value].
<unicode-range>	It identifies the range of the Unicode characters that will be supported. It can be: U+000-49F, U+2000-27FF, U+2900-2BFF, U+1D400-1D7FF, and so on.

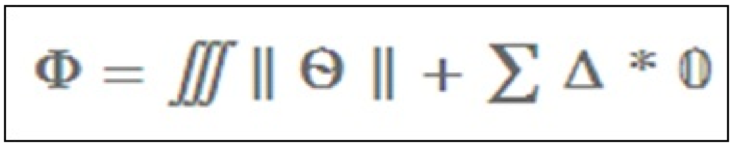
However, there are some other font specifications that have been published by the W3C and are still not fully supported by the modern browsers. These specifications can be divided into two major categories:

- **High-level access**: In this category, we can find different CSS properties such as font-kerning, font-variant, font-variant-ligatures, font-variant-position, font-variant-caps, and font-variant-numeric. It is not yet supported consistently in major modern browsers. However, this is most likely going to change in the near future.
- **Low-level access**: It is used within the CSS property, font-feature-settings. It is commonly used by type designers. It is associated to four letter tags (for example, smcp is used to identify small caps). In Firefox, the used syntax is -moz-font-feature-settings.

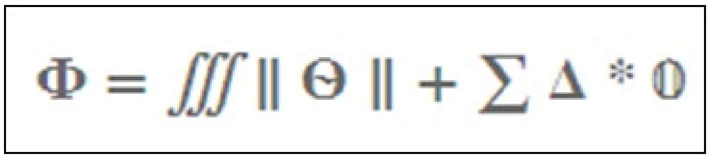
Font-weight = bold versus Bold font format

Sometimes, the use of font-weight = bold is different from loading a specific bold variant for the font. We can watch the difference in the following rendering:

- Without loading a specific bold variant of the font, as shown in the following screenshot:



- Using a specific font, as shown in the following screenshot:



As you can see, the difference is clear especially in the integration part. The first screenshot demonstrates that the character is transformed into bold which made it more condensed. The second screenshot demonstrates that it adopted another character which took care of that problem and gave us a character that had a logic space between the triple integration symbols.

Applying font properties via JavaScript

In the `InitFont` project, we used JavaScript to modify dynamically the font properties such as `font-family` and `font-size`. For example, the following function was used to dynamically change the font-family:

```
function SetFont(element,target)
{
  var elt = document.getElementById(target);
  elt.style.fontFamily=element;
}
```

The arguments of the function are: `target` which is the tag ID we want to apply the new `font-family` on and `element` is the font-family name. This function is triggered by an `onclick` property of the drop-down menu.

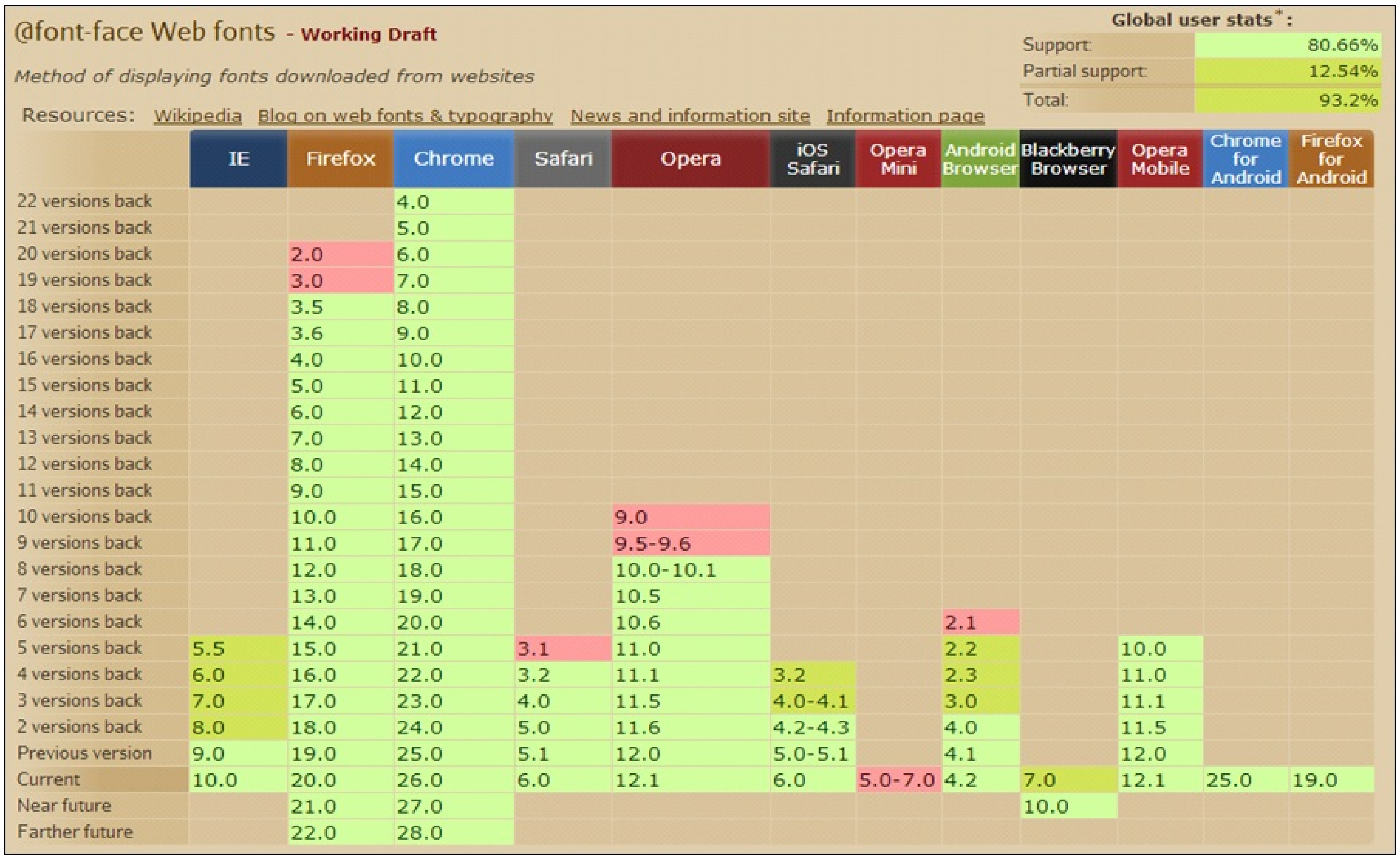
Font formats and browsers support

The following table provides definitions of all font formats which we can use on a website:

String	Font format	Common extensions
`woff`	Web Open Font Format	`.woff`

`truetype`	True Type Font	`.ttf`
`opentype`	Open Type	`.ttf`, `.otf`
`embedded-opentype`	Embedded Open Type	`.eot`
`svg`	Scalable Vector Graphics	`.svg`, `.svgz`

In the following screenshot, we have a display of the browser's compatibility with the `@font-face` rule. We can also find the degree of support in the different versions. An up-to-date version of this table can be found in the following link: http://caniuse.com/fontface.



@font-face Web fonts - **Working Draft**

Method of displaying fonts downloaded from websites

Resources: Wikipedia Blog on web fonts & typography News and information site Information page

Global user stats*:

Support:	80.66%
Partial support:	12.54%
Total:	93.2%

	IE	Firefox	Chrome	Safari	Opera	iOS Safari	Opera Mini	Android Browser	Blackberry Browser	Opera Mobile	Chrome for Android	Firefox for Android
22 versions back			4.0									
21 versions back			5.0									
20 versions back		2.0	6.0									
19 versions back		3.0	7.0									
18 versions back		3.5	8.0									
17 versions back		3.6	9.0									
16 versions back		4.0	10.0									
15 versions back		5.0	11.0									
14 versions back		6.0	12.0									
13 versions back		7.0	13.0									
12 versions back		8.0	14.0									
11 versions back		9.0	15.0									
10 versions back		10.0	16.0		9.0							
9 versions back		11.0	17.0		9.5-9.6							
8 versions back		12.0	18.0		10.0-10.1							
7 versions back		13.0	19.0		10.5							
6 versions back		14.0	20.0		10.6			2.1				
5 versions back	5.5	15.0	21.0	3.1	11.0			2.2		10.0		
4 versions back	6.0	16.0	22.0	3.2	11.1	3.2		2.3		11.0		
3 versions back	7.0	17.0	23.0	4.0	11.5	4.0-4.1		3.0		11.1		
2 versions back	8.0	18.0	24.0	5.0	11.6	4.2-4.3		4.0		11.5		
Previous version	9.0	19.0	25.0	5.1	12.0	5.0-5.1		4.1		12.0		
Current	10.0	20.0	26.0	6.0	12.1	6.0	5.0-7.0	4.2	7.0	12.1	25.0	19.0
Near future		21.0	27.0						10.0			
Farther future		22.0	28.0									

Global loading time versus font file size

The average file size of a normal font is about 40 KB. This means an average of 160

KB per font. STIX fonts are much heavier. They vary from 150 KB to 1024 KB with an average of 450 KB. In our project, we have loaded at worse 4 MB of font files, which means, we have added to the page an approximated download time as shown in the following table:

Download rate	**Download time**
128 Kbps	00:04:10
256 Kbps	00:02:05
1.024 Mbps	00:00:31

So we have to pay a close attention to such use of custom fonts, because if the page takes long time to load the website in our visitor's browser, then he will just ignore it and all the work we have done to enhance the template to make it stunning will just turn to become a handicap. So to keep a good loading time, it is recommended to use at most three fonts and keep the average total files size lower than 800 KB.

Creating different font files (Intermediate)

In the last recipe, we used different fonts and we manipulated the cross-browser compatibility. However, how to create or get these fonts and how to generate the different formats for use in different browsers (Embedded Open Type, Open Type, True Type Font, Web Open Font Format, and SVG font) is explained in this recipe.

Getting ready

To get the original file of the font created during this recipe in addition to the generated font formats and the full source code of the project `FontCreation`; please refer to the `receipe2` project folder.

How to do it...

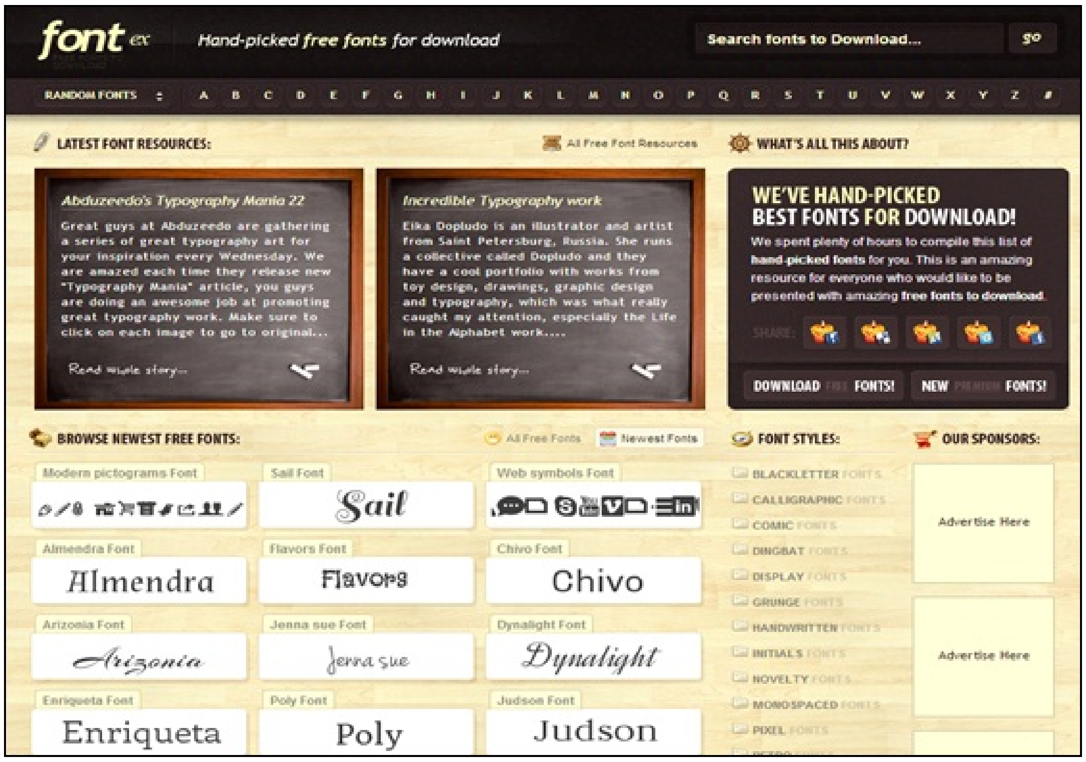
The following steps are preformed for creating different font files:

1. Firstly, we will get an original TTF font file. There are two different ways to get fonts:

   The first method is by downloading one from specialized websites. Both free and commercial solutions can be found with a wide variety of beautiful fonts.

   > **Note**
   >
   > The following are a few sites for downloading free fonts: Google fonts, Font squirrel, Dafont, ffonts, Jokal, fontzone, STIX, Fontex, and so on.Here are a few sites for downloading commercial fonts: Typekit, Font Deck, Font Spring, and so on.

   We will consider the example of Fontex, as shown in the following screenshot. There are a variety of free fonts. You can visit the website at http://www.fontex.org/.



fontex Hand-picked free fonts for download

Search fonts to Download... go

RANDOM FONTS A B C D E F G H I J K L M N O P Q R S T U V W X Y Z #

LATEST FONT RESOURCES: All Free Font Resources

Abduzeedo's Typography Mania 22

Great guys at Abduzeedo are gathering a series of great typography art for your inspiration every Wednesday. We are amazed each time they release new "Typography Mania" article, you guys are doing an awesome job at promoting great typography work. Make sure to click on each image to go to original...

Read whole story...

Incredible Typography work

Elka Dopludo is an illustrator and artist from Saint Petersburg, Russia. She runs a collective called Dopludo and they have a cool portfolio with works from toy design, drawings, graphic design and typography, which was what really caught my attention, especially the Life in the Alphabet work....

Read whole story...

WHAT'S ALL THIS ABOUT?

WE'VE HAND-PICKED BEST FONTS FOR DOWNLOAD!

We spent plenty of hours to compile this list of hand-picked fonts for you. This is an amazing resource for everyone who would like to be presented with amazing free fonts to download.

SHARE: DOWNLOAD FREE FONTS! NEW PREMIUM FONTS!

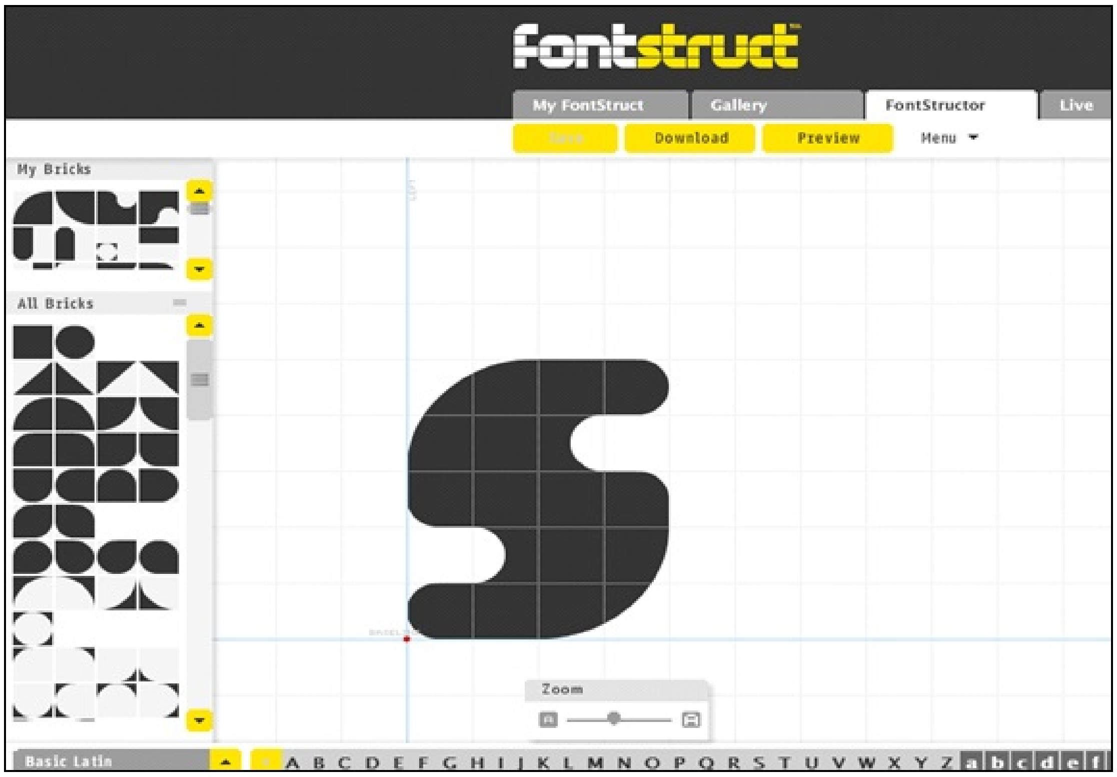
BROWSE NEWEST FREE FONTS: All Free Fonts Newest Fonts

Modern pictograms Font · Sail Font · Web symbols Font · Almendra Font · Flavors Font · Chivo Font · Arizonia Font · Jenna sue Font · Dynalight Font · Enriqueta Font · Poly Font · Judson Font

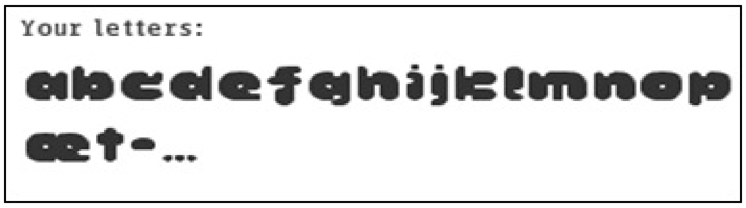
FONT STYLES: BLACKLETTER FONTS · CALLIGRAPHIC FONTS · COMIC FONTS · DINGBAT FONTS · DISPLAY FONTS · GRUNGE FONTS · HANDWRITTEN FONTS · INITIALS FONTS · NOVELTY FONTS · MONOSPACED FONTS · PIXEL FONTS

OUR SPONSORS: Advertise Here · Advertise Here

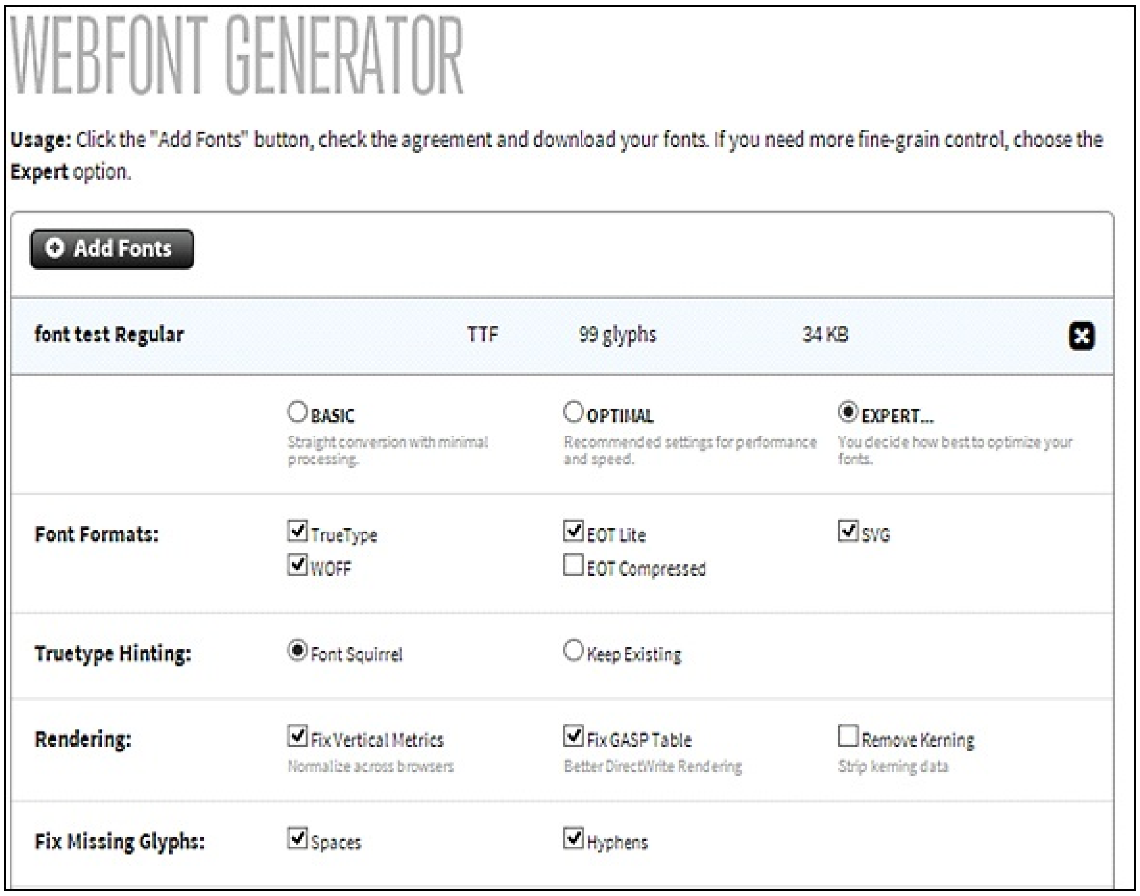
The second method is by creating your own font and then generating a TIFF file format. There are a lot of font generators on the Web. We can find online generators or follow the professionals by scanning handwritten typography and finally import it to Adobe Illustrator to change it into vector based letters or symbols. For newbies, I recommend trying Fontstruct (http://fontstruct.com). It is a WYSIWYG flash editor that will help you create your first font file, as shown in the following screenshot:



As you can see, we were trying to create the **S** letter using a grid and some different forms. After completing the font creation, we can now preview it rather than download the TTF file. The file is in the `receipe2` project folder. The following screenshot is an example of a font we have created on the run:



2. Now we have to generate the rest of file formats in order to ensure maximum compatibility with common browsers. We highly recommend the use of Font squirrel web font generator (http://www.fontsquirrel.com/tools/webfont-generator). This online tool helps to create fonts for `@font-face` by generating different font formats. All we need to do is to upload the original file (optionally adding same font variants `bold`, `italic`, or `bold-italic`), select the output formats, add some optimizations, and finally download the package. It is shown in the following screenshot:



WEBFONT GENERATOR

Usage: Click the "Add Fonts" button, check the agreement and download your fonts. If you need more fine-grain control, choose the **Expert** option.

Add Fonts

font test Regular	TTF	99 glyphs	34 KB

	BASIC Straight conversion with minimal processing.	OPTIMAL Recommended settings for performance and speed.	[✓ EXPERT...] You decide how best to optimize your fonts.
Font Formats:	[✓ TrueType] [✓ WOFF]	[✓ EOT Lite] EOT Compressed	[✓ SVG]
Truetype Hinting:	[✓ Font Squirrel]	Keep Existing	
Rendering:	[✓ Fix Vertical Metrics] Normalize across browsers	[✓ Fix GASP Table] Better DirectWrite Rendering	Remove Kerning Strip kerning data
Fix Missing Glyphs:	[✓ Spaces]	[✓ Hyphens]	

3. The following code explains the how to use this font:

```
<!DOCTYPE html>
<html>
```

```
  <head>
    <title>My first @font-face demo</title>
    <style type="text/css">
@font-face {
    font-family: 'font_testregular';
    src: url('font_test-webfont.eot');
    src: url('font_test-webfont.eot?#iefix')
format('embedded-opentype'),
         url('font_test-webfont.woff') format('woff'),
         url('font_test-webfont.ttf') format('truetype'),
         url('font_test-
webfont.svg#font_testregular')format('svg');
    font-weight: normal;
    font-style: normal;
}
```

Normal font usage:

```
h1 , p{
  font-family: 'font_testregular', Helvetica, Arial,sans-
serif;
}
h1 {
  font-size: 45px;
}
p:first-letter {
  font-size: 100px;
  text-decoration: wave;
}
p {
  font-size: 18px;
  line-height: 27px;}
</style>
```

Font usage in canvas:

```
<script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.4.2/jqu
ery.min.js" ></script>
<script language="javascript" type="text/javascript">
  var  x = 30, y = 60;
  function generate(){
  var canvas = $('canvas')[0],
  tx = canvas.getContext('2d');
  var t = 'font_testregular'; var c = 'red';
  var v =' sample text via canvas';
  ctx.font = '52px "'+t+'"';
  ctx.fillStyle = c;
  ctx.fillText(v, x, y);}
</script>
  </head>
  <body onload="generate();">
```

```
        <h1>Header sample</h1>
        <p>Sample text with lettrine effect</p>
        <canvas height="800px" width="500px">
    Your browser does not support the CANVAS element.
    Try the latest Firefox, Google Chrome, Safari or Opera.
        </canvas>
      </body>
    </html>
```

How it works...

This recipe takes us through getting an original TTF file:

- **Font download**: When downloading a font (either free or commercial) we have to pay close attention to terms of use. Sometimes, you are not allowed to use these fonts on the web and you are only allowed to use them locally.
- **Font creation**: During this process, we have to pay attention to some directives.

  We have to create **Glyphs** for all the needed alphabets (upper case and lower case), numbers, and symbols to avoid font incompatibility.

  We have to take care of the spacing between glyphs and eventually, variations, and ligatures. A special creation process is reserved for right- to left-written languages.

- **Font formats generation**: Font squirrel is a very good online tool to generate the most common formats to handle the cross-browser compatibility. It is recommended that we optimize the font ourselves via expert mode. We have the possibility of fixing some issues during the font creation such as missing glyphs, X-height matching, and Glyph spacing.
- **Font usage**: We will go through the following font usage:

  Normal font usage:

  We used the same method as already adopted via `font-family`; web-safe fonts are also applied:

  ```
  h1 , p{
  font-family: 'font_testregular', Helvetica, Arial, sans-
  serif;
  }
  ```

  Font usage in canvas:

  The `canvas` is a HTML5 tag that renders dynamically, bitmap images via scripts creating 2D shapes. In order to generate this image based on fonts, we will create the `canvas` tag at first. An alternative text will be displayed if canvas is not supported by the browser.

```
<canvas height="800px" width="500px">
Your browser does not support the CANVAS element.
Try the latest Firefox, Google Chrome, Safari or Opera.
</canvas>
```

We will now use the jQuery library in order to generate the canvas output. An `onload` function will be initiated to create the content of this tag:

```
<script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.4.2/jqu
ery.min.js" >
</script>
```

In the following function, we create a variable `ctx` which is a canvas occurrence of 2D context via `canvas.getContext('2d')`. We also define `font-family` using `t` as a variable, `font-size`, text to display using `v` as a variable, and color using `c` as a variable. These properties will be used as follows:

```
<script language="javascript" type="text/javascript">
  var  x = 30, y = 60;
  function generate(){
    var canvas = $('canvas')[0],
    ctx = canvas.getContext('2d');
    var t = 'font_testregular';
    var c = 'red' ;
    var v =' sample text via canvas';
```

This is for font-size and family. Here the font-size is `52px` and the font-family is `font_testregular`:

```
  ctx.font = '52px "'+t+'"';
```

This is for color by `fillstyle`:

```
  ctx.fillStyle = c;
```

Here we establish both text to display and axis coordinates where `x` is the horizontal position and `y` is vertical one.

```
  ctx.fillText(v, x, y);
```

Using Web fonts(Simple)

Using a font hosted in the web server is not the only solution. Sometimes, it is better to use fonts hosted in distant servers for many reasons such as support services and special loading scripts.

A lot of solutions are widely available on the web such as Typekit, Google fonts, Ascender, Fonts.com web fonts, and Fontdeck. In this task, we will be using Google fonts and its special JavaScript open source library, WebFont loader.

Getting ready

Please refer to the project `WebFonts` to get the full source code.

How to do it...

We will get through four steps:

1. Let us configure the `link` tag:

```
  <link rel="stylesheet" id="linker" type="text/css"
href="http://fonts.googleapis.com/css?
family=Mr+De+Haviland">
```

2. Then we will set up the `WebFont` loader:

```
  <script type="text/javascript">
  WebFontConfig = {
    google: {
      families: [ 'Tangerine' ]
    }
  };
  (function() {
    var wf = document.createElement('script');
    wf.src = ('https:' == document.location.protocol ?
'https' : 'http') +
'://ajax.googleapis.com/ajax/libs/webfont/1/webfont.js';
    wf.type = 'text/javascript';
    wf.async = 'true';
    var s = document.getElementsByTagName('script')[0];
    s.parentNode.insertBefore(wf, s);
  })();
  </script>
  <style type="text/css">
  .wf-loading p#firstp {
    font-family: serif
  }
  .wf-inactive p#firstp {
    font-family: serif
  }
  .wf-active p#firstp {
    font-family: 'Tangerine', serif
  }
```

3. Next we will write the `import` command:

```
  @import url(http://fonts.googleapis.com/css?
family=Bigelow+Rules);
```

4. Then we will cover font usage:

```
  h1 {
    font-size: 45px;
    font-family: "Bigelow Rules";
  }
  p {
    font-family: "Mr De Haviland";
    font-size: 40px;
    text-align: justify;
    color: blue;
    padding: 0 5px;}
  </style>
  </head>
  <body>
  <div id="container">
  <h1>This H1 tag's font was used via @import command
</h1>
  <p>This font was imported via a Stylesheet link</p>
  <p id="firstp">This font was created via WebFont loader
and managed by wf a script generated from webfonts.js.<br
/>
  loading time will be managed by CSS properties :
<i>.wf-loading , .wf-inactive and .wf-active</i> </p>
  </div>
  </body>
</html>
```

How it works...

In this recipe and for educational purpose, we used following ways to embed the font in the source code (the `link` tag, the `WebFont` loader, and the `import` command).

- **The link tag**: A simple `link` tag to a style sheet is used referring to the address already created:

```
<link rel="stylesheet" type="text/css"
href="http://fonts.googleapis.com/css?
family=Mr+De+Haviland">
```

- **The WebFont loader**: It is a JavaScript library developed by Google and Typekit. It grants advanced control options over the font loading process and exceptions. It lets you use multiple web font providers.

  In the following script, we can identify the font we used, Tangerine, and the link to predefined address of Google APIs with the world google:

  ```
  WebFontConfig = {
    google: { families: [ 'Inconsolata:bold' ] }
  };
  ```

  We now will create `wf` which is an instance of an asynchronous JavaScript element. This instance is issued from Ajax Google API:

  ```
  var wf = document.createElement('script');
  wf.src = ('https:' == document.location.protocol ? 'https'
  : 'http') +
  '://ajax.googleapis.com/ajax/libs/webfont/1/webfont.js';
  wf.type = 'text/javascript';
  wf.async = 'true';
  var s = document.getElementsByTagName('script')[0];
  s.parentNode.insertBefore(wf, s);
  })();
  ```

  We can have control over fonts during and after loading by using specific class names. In this particular case, only the `p` tag with the ID `firstp` will be processed during and after font loading.

  During loading, we use the class `.wf-loading`. We can use a safe font (for example, Serif) and not the browser's default page until loading is complete as follows:

  ```
  .wf-loading p#firstp {
    font-family: serif;
  }
  ```

  After loading is complete, we will usually use the font that we were importing earlier. We can also add a safe font for older browsers:

  ```
  .wf-active p#firstp {
    font-family: 'Tangerine', serif;
  }
  ```

- **Loading failure**: In case we failed to load the font, we can specify a safe font to avoid falling in default browser's font:

```
.wf-inactive p#firstp {
  font-family: serif;
}
```

- **The import command**: It is the easiest way to link to the fonts:

```
@import url(http://fonts.googleapis.com/css?
family=Bigelow+Rules);
```

- **Font usage**: We will use the fonts as we did already via font-family property:

```
h1 {
  font-family: "Bigelow Rules";
}
p {

  font-family: "Mr De Haviland";
}
```

There's more...

The WebFont loader has the ability to embed fonts from mutiple WebFont providers. It has some predefined providers in the script such as Google, Typekit, Ascender, Fonts.com web fonts, and Fontdeck. For example, the following is the specific source code for Typekit and Ascender:

```
WebFontConfig ={
  typekit: {
    id: 'TypekitId'
  }
};
WebFontConfig ={
  ascender: {
    key: 'AscenderKey',
    families: ['AscenderSans:bold,bolditalic,italic,regular']
  }
};
```

For the font providers that are not listed above, a custom module can handle the loading of the specific style sheet:

```
WebFontConfig = {
  custom: {
    families: ['OneFont', 'AnotherFont'],
    urls:
['http://myotherwebfontprovider.com/stylesheet1.css','http://y
etanotherwebfontprovider.com/stylesheet2.css' ]
  }
};
```

For more details and options of the WebFont loader script, you can visit the following link:

https://developers.google.com/fonts/docs/webfont_loader

To download this API you may access the following URL:

https://github.com/typekit/webfontloader

How to generate the link to the font?

The URL used in every method to import the font in every method (the `link` tag, the `WebFont` loader, and the `import` command) is composed of the Google fonts API base `url` (http://fonts.googleapis.com/css) and the `family` parameter including one or more font names, `?family=Tangerine`. Multiple fonts are separated with a pipe character (`|`) as follows:

```
?family=Tangerine|Inconsolata|Droid+Sans
```

Optionally, we can add subsets or also specify a style for each font:

```
Cantarell:italic|Droid+Serif:bold&subset=latin
```

Browser-dependent output

The Google fonts API serves a generated style sheet specific to the client, via the browser's request. The response is relative to the browser. For example, the output for Firefox will be:

```
@font-face {
  font-family: 'Inconsolata';
  src: local('Inconsolata'),
       url('http://themes.googleusercontent.com/fonts/font?
kit=J_eeEGgHN8Gk3Eud0dz8jw') format('truetype');
}
```

This method lowers the loading time because the generated style sheet is relative to client's browser. No multiformat font files are needed because Google API will generate it, automatically.

Enhancing fonts with outstanding effects (Intermediate)

We can boost the font with some beautiful effects instead of using images. Two methods can be used to use custom fonts, **web hosted** font or **locally hosted** font. We will detail the use of both methods in the following examples.

Getting ready

To get the full code please refer to `PrebuiltFontEffect` and `FontEffect` projects.

How to do it...

This recipe takes us through two parts: using predefined effects from Google fonts and creating effects via CSS properties.

We will first cover predefined effects:

1. We will first import the libraries of fonts and effects from Google fonts:

    ```
      <link rel="stylesheet" type="text/css"
    href="http://fonts.googleapis.com/css?
    family=Philosopher|Tangerine&effect=anaglyph|brick-
    sign|canvas-
    print|crackle|decaying|destruction|emboss|fragile|ice|neon
    |static|3d-float|wallpaper|shadow-multiple|fire-
    animation|3d">
      <style type="text/css">
      h1 {
        font-size: 45px;
        font-family: Philosopher;
      }
      p {
        font-family: "Tangerine";
        font-size: 30px;
        line-height: 50px;
        text-align: justify;
        color: blue;}
      </style>
    ```

2. To make it dynamic, we use the following script to alter the font-family:

    ```
    <script type="text/javascript" id="script">
    function SetEffect() {
      var elt = document.getElementById("toeffect");
      var t = document.getElementById("selctor").value;
    ```

```
      elt.className = t;
    }
    </script>
    </head>
    <body>
  Choose a Font effect to apply to the text on the right
  side:<br />
    <select size="1" id="selctor" onchange="SetEffect();">
    <option value="font-effect-anaglyph">
    Anaglyph
    </option>
    <option value="font-effect-brick-sign">
    Brick Sign
    </option>
    <option value="font-effect-canvas-print">
    Canvas Print
    </option>
    <option value="font-effect-crackle">
    Crackle
    </option>
    </select>
```

3. This is the method defined by Google to use the font effect:

```
    <h1 class="font-effect-brick-sign" id="toeffect"
  contenteditable="true">
    Sample of pre-built font effects from Google fonts
    </h1>
    <p class="font-effect-shadow-multiple">
    These effects are downloaded from web font host:
  Googlefonts
    </p>
    </body>
  </html>
```

Secondly, we can create effects using CSS3 as follows:

```
body {
  font-family: "Lucida Grande", Lucida, Verdana, sans-serif;
  text-transform: uppercase;
}
```

1. This is how we create some simple shadow effects:

```
.shadow {
  text-shadow: 1px 2px 5px #fff;
}
```

2. This is how to create some complex shadow effects in order 3D, Fire, Neon, and Flare:

```
.effect-3d {
  color: #ddd;
  text-shadow: 0 1px 0 #ccc, 0 2px 0 #bbb,
  0 3px 0 #aaa, 0 4px 0 #999,
  0 5px 0 #888, 0 6px 0 #777,
  0 7px 0 #666, 0 8px 7px rgba(0, 0, 0, 0.5),
  0 9px 10px rgba(0, 0, 0, 0.1);
  font-size: 40px;
}
h2.fire {
  color: #004080;
  text-shadow: 0 0 20px #fefcc9, 10px -10px 30px #feec85,
  -20px -20px 40px #ffae34, 20px -40px 50px #ec760c,
  -20px -60px 60px #cd4606, 0 -80px 70px #973716,
  10px -90px 80px #451b0e;
  font-size: 50px;
  background: #CCCCCC;
}
.neon {
  font-size: 45px;
  text-shadow: 0 0 5px #0051CA, 0 0 15px #0051CA,
  0 0 25px #0051CA, 0 0 50px #0051CA,
  0 0 80px #0051CA;
  color: #A5F1FF;
  background-color: black;
  padding: 15px 0;}
h2 em {
  color: #FFF;
  text-shadow: 0 0 150px #FFF, 0 0 60px #FFF, 0 0 10px
#FFF;
  font-size: 250px;
  text-transform: uppercase;
}
.wrap {
  background: #E22;
  background: -moz-radial-gradient(center 40%, circle
cover, #F41 0%, #400 100%);
  background: -webkit-gradient(radial, center 40%, 0,
center 40%, 800, from(#F41), to(#400) );}
// We will now apply the gradient effect
.gradient h1 {
  color: # #ACB013;
  -webkit-mask-box-image: -webkit-gradient(linear,right
top, right bottom,
  color-stop(10%,rgba(0,0,0,1)),
  color-stop(100%,rgba(0,0,0,0)));
  font-size: 50px;
  padding: 10px 50px;
}
```

3. And now we will create a real image effect over text:

```
.likezebra {
  position:relative;
  color:#8080FF ;
  background-color: black;
}
```

```
.likezebra span{
  position:absolute;
  top:0;left:0;
  height: 50px;
  width:100%;
  background:url(images/zebra.png);
}
  </style>
  </head>
```

How it works...

We will distinguish the enhancement of fonts to create the following effects:

Predefined effects

- **Importing libraries of fonts and effects**: We will import the fonts and effects libraries from Google fonts. The method is the same as in the previous recipe via the `link` tag for the font import. To include the effects we add the parameter `effect` in the URL and its value. Effects are delimited by a pipe character (`|`):

```
<link rel="stylesheet" type="text/css"
href="http://fonts.googleapis.com/css?
family=Philosopher|Tangerine&effect=anaglyph|brick-
sign|canvas-
print|crackle|decaying|destruction|emboss|fragile|ice|neon
|static|3d-float|wallpaper|shadow-multiple|fire-
animation|3d">
```

- **Using the effect**: To use the effect in a specific tag, we have to create a predefined class in relation to the selected effect in the `link` tag. Usually the class name is the effect name prefixed with `font-effect`:

```
<h1 class="font-effect-brick-sign" id="toeffect"
contenteditable="true">
Sample of pre-built font effects from Google fonts
</h1>
```

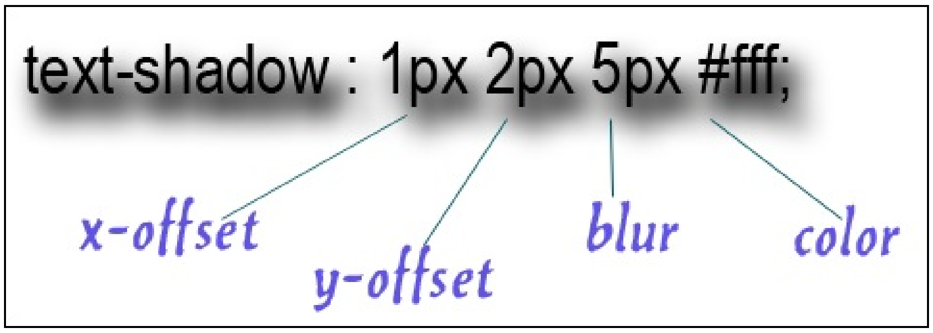
- **Dynamically change the effect**: We can change the effect dynamically by changing the class name via JavaScript. The input is given by values from a drop-down menu:

```
function SetEffect() {
  var elt = document.getElementById("toeffect");
  var t = document.getElementById("selctor").value;
  elt.className = t;
}
```

The `h1` tag will be the target of the change by editing the class name. The name is issued from the selected option of the drop-down menu.

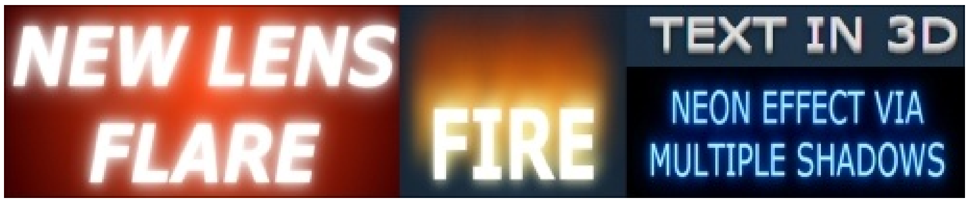
Create effects using CSS3

- **Simple shadow effect**: CSS3 has a property that adds a shadow to each letter in its simplest form. We used the simplest form in the following screenshot:



The first parameter is the **x-offset** (horizontal), the second one is the **y-offset**, then the **blur** (the spread) and finally the **color** of the shadow.

- **Complex shadow effects using 3D, Fire, Neon, and Flare**: We don't have to settle with one shadow, we can build multiple ones to create some really beautiful effects. Different colors, offset, and blur distance can generate some effect illusions such as fire, neon, and 3D as shown in the following screenshot:



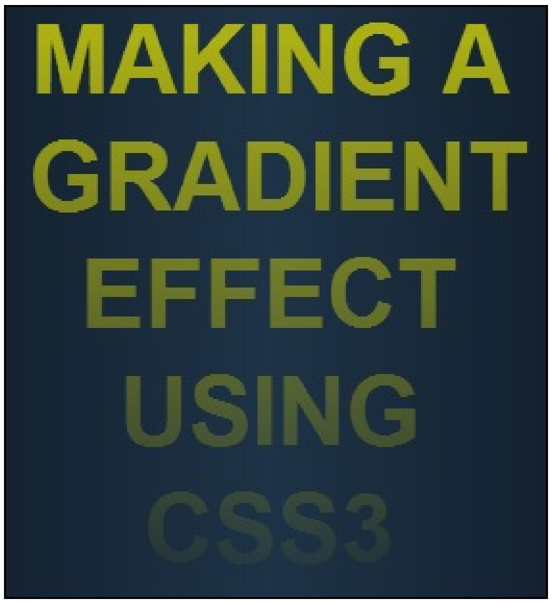
In this example of 3D effect, the tip is to create shadows every 1px with a degraded color from #999 to #333. Finally, in the last two layers, we will reduce the opacity:

```
    text-shadow: 0 1px 0 #ccc, 0 2px 0 #bbb,
                 0 3px 0 #aaa, 0 4px 0 #999,
                 0 5px 0 #888, 0 6px 0 #777,
                 0 7px 0 #666, 0 8px 7px rgba(0, 0, 0, 0.5),
                 0 9px 10px rgba(0, 0, 0, 0.1);
    font-size: 40px;
  }
```

In Fire, Flare, and Neon for the effects we have created, the result is mainly obtained by increasing the blur parameter via multiple shadows. As an example, the following is the code to create the Neon effect:

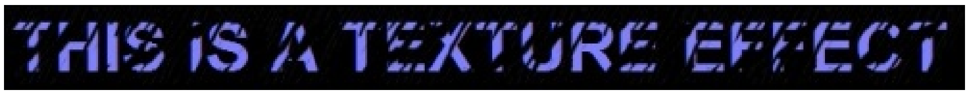
```
text-shadow: 0 0 5px #0051CA, 0 0 15px #0051CA,
             0 0 25px #0051CA, 0 0 50px #0051CA,
             0 0 80px #0051CA;
   color: #A5F1FF;
```

- **Gradient effect**: The keys to create the gradient effect are mainly `-webkit-mask-box-image` (which sets the mask image for an element's border box) and `–webkit-gradient` (which generates a gradient, in particular, a linear gradient). The following screenshot shows a gradient effect:



```
  -webkit-mask-box-image: -webkit-gradient(linear,right
top, right bottom, color-stop(10%,rgba(0,0,0,1)),color-
stop(100%,rgba(0,0,0,0)));
```

- **Image effect over text**: To create this effect we need to use a trick; we will use an empty span over the `h1` tag and place it as an **absolute** position inside the other **relative** one, as shown in the following screenshot:



You can note that the text cannot be selected and that is due to the image on top of the `h1` tag. To remedy to this problem you may use the following CSS rule `pointer-event:none`; in the `span` tag:

```
.likezebra span {
  position:absolute;
  top:0;
  left:0;
  height: 50px;
  width:100%;
  background:url(images/zebra.png);
}
```

Using Cufón – fonts for the people (Intermediate)

On older versions of Windows, the use of fonts in the `@font-face` property may look hideous. This result is due to Windows' **font rendering engine** that does not smooth the edge of each character of the used font. This "disastrous" result is more visible in larger font sizes.

Cufón—fonts for the people (commonly pronounced as koo-fon) is a replacement technique. Cufón may provide an alternative solution where older Windows versions failed to render nice looking fonts. It is based on JavaScript and vector graphics and uses an OTF or TTF font to generate the required result.

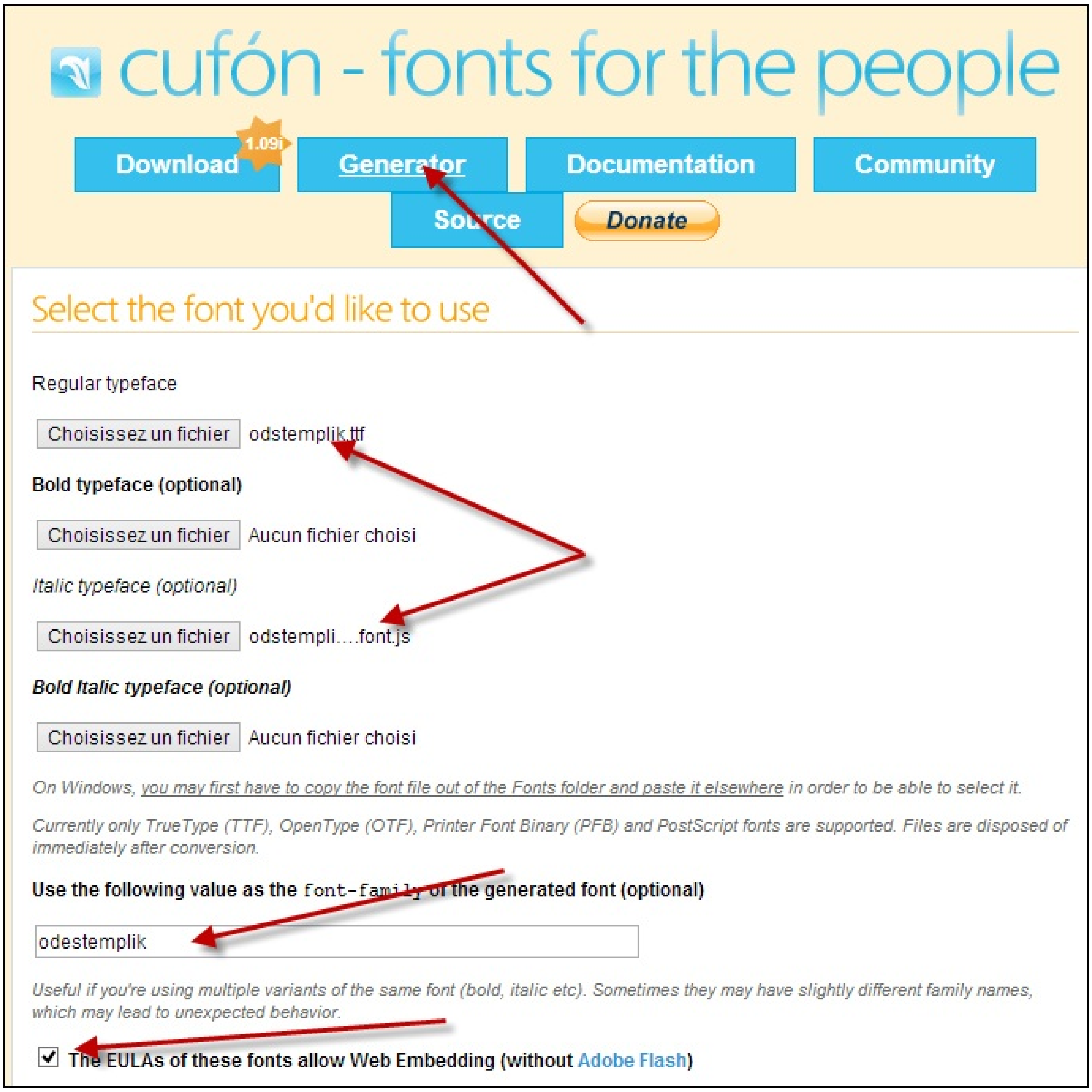
Getting ready

To start this task, we need at first to dispose of a couple of fonts in OTF or TTF format and Cufón's source. These files are in the project `CufonUse`. For the latest versions of Cufón please visit the download page, http://cufon.shoqolate.com/js/cufon-yui.js.

How to do it...

Let's list the different steps to create fonts based on Cufón:

1. Generating the JavaScript version of the font via Cufón.
2. After downloading or creating the font, it is time to generate the vector graphic. Visit the font generator page of Cufón at http://cufon.shoqolate.com/generate/. We may upload our original font file in **Regular typeface** and any other options such as **Italic typeface**. Finally, we may select the **The EULAs of these...** checkbox, as shown in the following screenshot:



Select the font you'd like to use

Regular typeface

Choisissez un fichier odstemplik.ttf

Bold typeface (optional)

Choisissez un fichier Aucun fichier choisi

Italic typeface (optional)

Choisissez un fichier odstempli….font.js

Bold Italic typeface (optional)

Choisissez un fichier Aucun fichier choisi

On Windows, you may first have to copy the font file out of the Fonts folder and paste it elsewhere in order to be able to select it.

Currently only TrueType (TTF), OpenType (OTF), Printer Font Binary (PFB) and PostScript fonts are supported. Files are disposed of immediately after conversion.

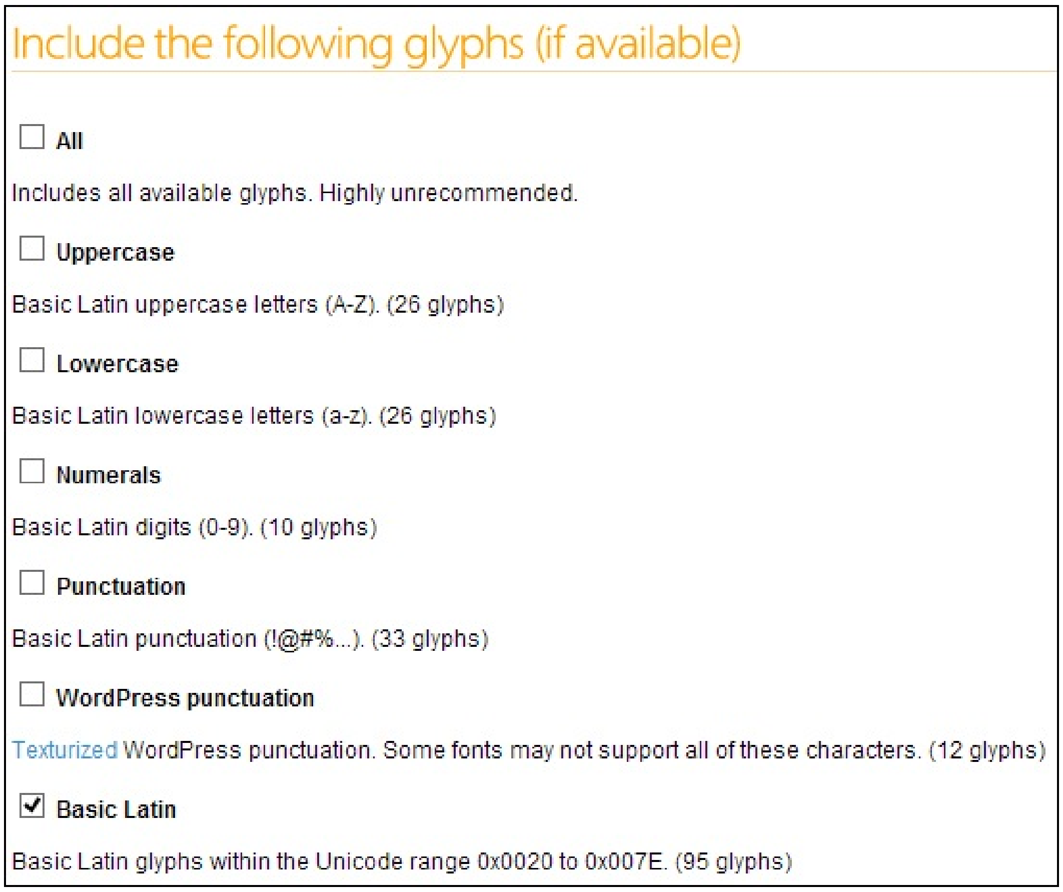
Use the following value as the `font-family` of the generated font (optional)

odestemplik

Useful if you're using multiple variants of the same font (bold, italic etc). Sometimes they may have slightly different family names, which may lead to unexpected behavior.

☑ **The EULAs of these fonts allow Web Embedding (without Adobe Flash)**

3. To ensure maximum compatibility, we may select the options on the **Including the following glyphs (if available)** page. This is shown in the following screenshot:


Include the following glyphs (if available)

☐ **All**

Includes all available glyphs. Highly unrecommended.

☐ **Uppercase**

Basic Latin uppercase letters (A-Z). (26 glyphs)

☐ **Lowercase**

Basic Latin lowercase letters (a-z). (26 glyphs)

☐ **Numerals**

Basic Latin digits (0-9). (10 glyphs)

☐ **Punctuation**

Basic Latin punctuation (!@#%...). (33 glyphs)

☐ **WordPress punctuation**

Texturized WordPress punctuation. Some fonts may not support all of these characters. (12 glyphs)

☑ **Basic Latin**

Basic Latin glyphs within the Unicode range 0x0020 to 0x007E. (95 glyphs)

4. Now, we may accept the terms of use and download the file, `jokal_400.font.js`. We may also need to do this with the second font file `pincoy_900.font.js`.
5. It is time to create the `CufonUse` project:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Using Cufon Simple</title>
```

6. Then load the scripts as follows.

   These are Cufón and generated scripts of the original fonts that need to be loaded:

```
<script src="cufon-yui.js" type="text/javascript">
</script>
<script src="jokal_400.font.js" type="text/javascript">
</script>
<script src="pincoy_900.font.js" type="text/javascript">
</script>
```

7. After loading these scripts, it's now time to execute the scripts as follows:

```
<script type="text/javascript">
  Cufon.now();
```

```
</script>
<script type="text/javascript">
  Cufon.replace('h1', { fontFamily: 'Pincoy' });
  Cufon.replace('h2', { fontFamily: 'jokal' });
</script>
<style type="text/css">
  h1 { font-size: 35px;}
  h2 { font-size: 20px;}
</style>
</head>
<body>
  <div id="head">
    <h1>Header 1</h1>
    <h2>Header 2</h2>
  </div>
</body>
</html>
```

How it works...

- **Generating the JavaScript version of the font via Cufón**: Cufón is using a proprietary JavaScript that generates a vector form transformed into a VML (Vector Markup Language). We are now disposing of both generated files (`jokal_400.font.js` and `pincoy_900.font.js`) in the JSON format. The script files are composed of lot of numbers that define curves and shapes. The result will be displayed in the `<canvas>` element of HTML5.

  The generated file has two major advantages:

  - **Size**: Once the processing is complete, the file size becomes 20 to 40 percent lighter.
  - **No plugin**: This is not a Flash format, so we will not need a plugin.

- **Loading the scripts**: We will now refer the three JavaScript files related to Cufón script and the generated font files. All these files are placed under the `js` folder:

  ```
  <script src="js/cufon-yui.js" type="text/javascript">
  </script>
  <script type="text/javascript">
  Cufon.now();
  </script>
  <script src="js/jokal_400.font.js" type="text/javascript">
  </script>
  <script src="js/pincoy_900.font.js"
  type="text/javascript">
  </script>
  ```

- **Executing the script**: We will now replace all fonts in `h1` with `Pincoy` and in `h2` with jokal. To solve problems caused by IE (as usual) concerning a delay until font is displayed, we need to execute the Cufon.now() method:

```
<script type="text/javascript">
Cufon.now();
</script>
<script type="text/javascript">
  Cufon.replace('h1', { fontFamily: 'Pincoy' });
  Cufon.replace('h2', { fontFamily: 'jokal' });
</script>
```

There's more...

To be able to select only specific h1 tags and not all of them as in the example, two methods can be used, that is, select either via **JavaScript** or via **jQuery**.

- **JavaScript**: To select a specific h1 tag only inside head div we use a simple identification method:

```
Cufon.replace(document.getElementById('header').getElement
sByTagName('h1') , { fontFamily: 'Pincoy' });
```

- **jQuery**: As for jQuery we need to import its library first. Then we need to identify all h1 inside head div via #headerh1:

```
<script
type="text/javascript"src="http://ajax.googleapis.com/ajax
/libs/jquery/1.3.2/jquery.min.js">
</script>
Cufon.replace('#headerh1', { fontFamily: 'Pincoy' });
```

About sIFR

Cufón may not be the only solution to generate a vector graphic alternative or the only remedy for older Windows versions to generate fonts, but it is still the most compact and simple to use method. The second in use is **sIFR** (**Scalable Inman Flash Replacement**) which generates fonts in Flash format and is considered difficult to implement. The following code illustrates a sample of sIFR usage:

```
<script src="js/sifr.js" type="text/javascript">
</script>
<h1>This is an example for sIFR</h1>
<span class="sifr">www.sifrgenerator.com</span></p>
<script type="text/javascript">
//<![CDATA[
if(typeof sIFR == "function"){
  sIFR.replaceElement(named({sSelector:"h1",
sFlashSrc:"swf/tradegothic.swf"}));
sIFR.replaceElement(named({sSelector:".sifr",
sFlashSrc:"swf/tradegothic.swf",sColor:"#ff0000",sBgColor:"#cc
ccff"}));
}
//]]>
</script>
```

Combing fonts and other HTML5/CSS3 features (Advanced)

By combining HTML5, CSS, and JavaScript features we can render an image or a flash result. It has the advantage of being easily alterable because we don't need any editing software. In addition, it is usually lighter in weight which means a faster loading web page. In fact, usually an image's weight is normally about 100 KB. However, using a combination of CSS3 and HTML5 is usually less than 10 KB and the result is almost the same.

Getting ready

To get the full code, images, and fonts, please refer to the `ImageLikeRendering` project. More outcomes are in the project folder.

How to do it...

Let's list the different outcomes we created.

To begin with, we will see the Neon effect on the night.

1. First, we will embed the fonts that seem like neon tubes using the following code:

```
@font-face {
  font-family: 'micromoog_remixregular';
  src: url('font/micromoog_remix-webfont.eot');
  src: url('font/micromoog_remix-webfont.eot?#iefix')
format('embedded-opentype'),
       url('font/micromoog_remix-webfont.woff')
format('woff'),
       url('font/micromoog_remix-webfont.ttf')
format('truetype'),

url('font/micromoog_remixwebfont.svg#micromoog_remixregula
r')format('svg');
  font-weight: normal;
  font-style: normal;
}
```

2. The Neon effect is created as follows:

```
#neon_cont {
  background: url(img/night.jpg) no-repeat top left;
  cursor: pointer;
}
```

3. We will create a neon circle in a `div` that will be overflown by a black `div` rectangle:

```
#neoncircle {
  position: relative;
  height: 280px;
  width: 280px;
  border: #D0F8FF ridge 10px;
  -webkit-border-radius: 190px / 190px;
  -moz-border-radius: 190px / 190px;
  border-radius: 190px / 190px;
  opacity: 0.6;
  box-shadow: 0 0 5px #A5F1FF, 0 0 10px #A5F1FF, 0 0 20px
#A5F1FF, 0 0 30px #A5F1FF, 0 0 40px #A5F1FF;
}
#contentblack {
  position: absolute;
  height: 210px;
  width: 370px;
  border: none;
  background-color: black;
  -webkit-transform: rotate(-15deg);
  -moz-transform: rotate(-15deg);
  -o-transform: rotate(-15deg);
  writing-mode: lr-tb;
}
```

4. In the following code we will create the holder of the text by applying a rotation of -15 degrees:

```
#neoneffect {
  position: absolute;
  top: 85px;
  margin-left: 80px;
  -webkit-transform: rotate(-15deg);
  -moz-transform: rotate(-15deg);
  -o-transform: rotate(-15deg);
  writing-mode: lr-tb;
  height: 280px;
  width: 280px;
  text-align: center;
}
```

5. The following code is the part of the effect of neon over the text followed by the on hover action:

```
.neoneff {
  font-family: 'micromoog_remixregular';
  color: #D0F8FF;
  text-shadow: 0 0 5px #A5F1FF, 0 0 10px #A5F1FF, 0 0 20px
#A5F1FF, 0 0 30px #A5F1FF, 0 0 40px #A5F1FF;
  font-size: 60px;
  transition: text-shadow 2s cubic-bezier(0.42,0,1,1);
}
```

```
.neoneff:hover {
  text-shadow: 0 0 10px #A5F1FF, 0 0 20px #A5F1FF,0 0 40px
#A5F1FF, 0 0 70px #A5F1FF, 0 0 100px #A5F1FF;
  color: #A5F1FF;
  transition: text-shadow 1s cubic-bezier(0.42,0,1,1);
}
```

Now we look into 3D wallpaper:

1. The following code is the container of the background image in addition to general parameters, followed by the paragraph parameters which will sustain some spatial modifications:

```
#wrapper {
  position: relative;
  text-align: center;
  font-weight: bold;
  font-family: "Lucida Grande", Lucida, Verdana,sans-
serif;
  width: 930px;
  height: 402px;
  background: url(img/trees.jpg) no-repeat center center;
}
#wrapper p {
  text-transform: uppercase;
  letter-spacing: 0.03em;
  text-shadow: rgba(0,0,0,0.1) 0 20px 80px;
  transform: rotate(5.5deg) rotateX(50deg) skewX(-4deg);
  -ms-transform: rotate(5.5deg) rotateX(50deg)
skewX(-4deg);
  webkit-transform: rotate(5.5deg) rotateX(50deg)
skewX(-4deg);
}
```

2. Now, we can identify the distance to reach the paragraph via number of the child:

```
#wrapper p:nth-child(1) {
  font-size: 35px;
  text-shadow: 0 0 10px #fff, 0 4px 3px #ddd, 0 9px 3px
#ccc,  0 12px 1px #eee;
}
#wrapper p:nth-child(2) {
  font-size: 44px;
  text-shadow: 0 0 10px #fff, 0 4px 3px #ddd, 0 9px 3px
#ccc,  0 12px 1px #eee;
}
```

Next we will look at a Game promo:

1. The following code is the container of the background image containing some general parameters:

```
#wra {
  text-align: center;
  font-weight: bold;
  font-size: 70px;
  background: url(img/war.jpg) no-repeat center center;
}
#wra span {
  text-shadow:   5px 0 3px grey;
  letter-spacing: 10px;
}
```

2. Next, we will create a horizontal line and add some shadow effects:

```
#wra hr {
  width: 300px;
  height: 2px;
  background-color: black;
  color: black;
  box-shadow: 2px 0 7px gray;
}
```

3. The following code is meant to shape the stars over the text:

```
#stars span {
  display: inline-block;
  color: gold;
  vertical-align: middle;}
#stars2 {
  margin: -20px 0; color: gold;
}
#stars span:nth-child(1), #stars span:nth-child(5) {
  font-size: 55px;
}
#stars span:nth-child(2), #stars span:nth-child(4) {
  font-size: 30px;
}
#stars span:nth-child(3) {
  font-size: 15px;
}
#title {
  font-family: sans-serif;
}
```

4. Then add a transition on `hover` for every character by increasing its size:

```
#title span {
  vertical-align: middle;
  line-height: 1.5em;
  transition: font-size 2s cubic-bezier(0, 1, 0, 1);
}
#title span:hover {
  font-size: 90px;
  line-height: 1em;
  transition: font-size .2s cubic-bezier(0, 0.75, 0, 1);
}
#slogan span{
  font-size: 20px;text-shadow: 1px 1px 1px white;
}
.tiny {
  font-size: 10px;
}
```

We now come across elevated writing from shadow with drag-and-drop animation.

5. The following code is the part to initialize the position of the holder of the main and elevated `div` in the z axis:

```
#content {
  -webkit-perspective: 1200;
  width: 100%;
  overflow: hidden;
}
#posters {
  position: relative;
  font-weight: 900;
  font-size: 62px;
  text-transform: uppercase;
  -webkit-transform-style: preserve-3d;
  -webkit-transform: translateZ(-200px)
}
.csstransforms3d #posters.reset {
  -webkit-transition: -webkit-transform .5s;
}
.csstransforms3d #posters:hover {
  cursor: move;
}
#posters > div {
  position: absolute;
  left: -500px;
  top: 0;
}
```

6. Then we will create the indented and rotation effects of each span of the main `div`:

```
#posters span {
  display: block;
  line-height: 0.9em;
  position: relative;
  white-space: nowrap;
  top: 250px;
  left: -200px;
}
#posters span:nth-of-type(1) { left: 8.8em; }
#posters span:nth-of-type(2) { left: 7.9em; }
#posters span:nth-of-type(3) { left: 7em; }
```

```
#posters span:nth-of-type(4) { left: 6.1em; }
#posters span:nth-of-type(5) { left: 5.2em; }
#posters span:nth-of-type(6) { left: 6.2em; }
#posters span:nth-of-type(7) { left: 7.1em; }
#posters span:nth-of-type(8) { left: 8.0em; }
#posters span:nth-of-type(9) { left: 8.9em; }
#posters span:nth-of-type(10) { left: 9.8em; }
#posters span:nth-of-type(11) { left: 10.8em;}
#posters .rotor-z {
  -webkit-transform: rotate(-45deg);
  -moz-transform: rotate(-45deg);
  -o-transform: rotate(-45deg);
  transform: rotate(-45deg);
}
#behold {
  color: #FFF;
  -webkit-transform: translate3d( 0, 0, 200px);
}
```

7. The shadow effect is added as follows:

```
#shadow {
  color: hsla(0,0,0,0);
  text-shadow: 0 5px 10px hsla(0,0,0,.2);
  -webkit-transform: translate3d( 0, 0, -200px);
}
.csstransforms3d #shadow{
text-shadow: 0 0px 30px hsla(0,0%,0%,.15);
}
```

8. To create the effect, we will use two scripts:

```
<script src="behold-dropshadows.js"></script>
<script src="modernizr.js"></script>
```

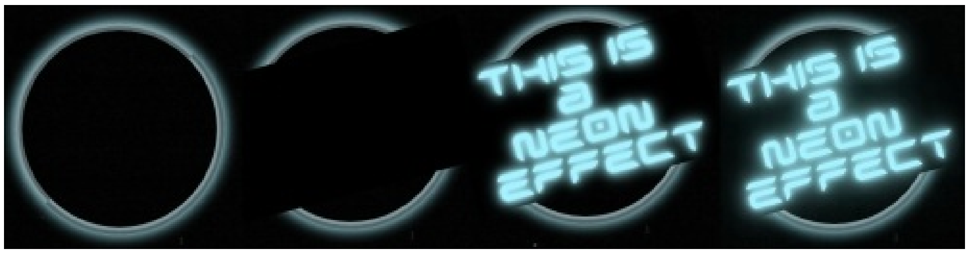
How it works...

Let's now explain how we managed to get these beautiful effects:

- **Neon effect on the night**: To generate this result we need to download (or create) a font that looks like wires or calculator digits. We used the `micromoog_remixregular` font downloaded from fontstruct.com and generated the cross-browser font formats via fontsquirrel.com. Later, we will create the Neon effect via `text-shadow`. On hover we will make it look as if the neon will become brighter via the `transition` property of the shadow. We will be using `cubic-bezier` to make this happen:

```
.neoneff {
  text-shadow: 0 0 5px #A5F1FF, 0 0 10px #A5F1FF, 0 0 20px
#A5F1FF, 0 0 30px #A5F1FF, 0 0 40px #A5F1FF;
  transition: text-shadow 2s cubic-bezier(0.42,0,1,1);
}
.neoneff:hover {
  text-shadow: 0 0 10px #A5F1FF, 0 0 20px #A5F1FF, 0 0
40px #A5F1FF, 0 0 70px #A5F1FF, 0 0 100px #A5F1FF;
  transition: text-shadow 1s cubic-bezier(0.42,0,1,1);
}
```

We can notice two arcs over and under the writing. The key to generating this is to create a `div` whose borders are so rounded that it will look like a circle. We have then applied a `box-shadow` property (the shadow is similar to the `text-shadow` property). We will now create a black rectangular `div` that will hover the circle in the middle. By doing so, we will simulate two neon arcs instead of a circle. A rotation of -15 degrees is applied on the black `div` and the text to make it look artistic via `rotate(-15deg)`, as shown in the following screenshot:

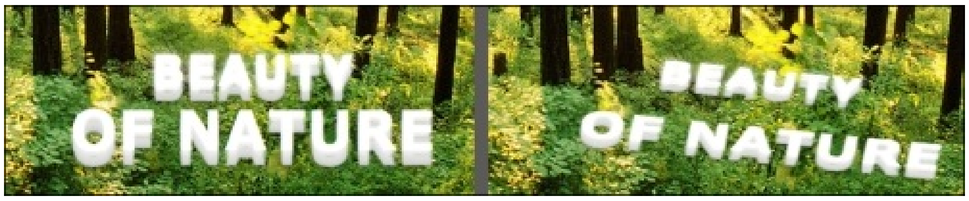


```
#neoncircle {
  border: #D0F8FF ridge 10px;
  -webkit-border-radius: 190px / 190px;
  -moz-border-radius: 190px / 190px;
  border-radius: 190px / 190px;
  opacity: 0.6;
  box-shadow: 0 0 5px #A5F1FF, 0 0 10px #A5F1FF, 0 0 20px
#A5F1FF, 0 0 30px #A5F1FF, 0 0 40px #A5F1FF;
}
#contentblack {
  border: none;
  background-color: black;
  -webkit-transform: rotate(-15deg);
  -moz-transform: rotate(-15deg);
  -o-transform: rotate(-15deg);
  writing-mode: lr-tb;
}
```

- **3D wallpaper**: We want to create a perspective 3D. First, we will identify each paragraph of the `div` by the properties `p:nth-child(1)` and `p:nth-child(2)`. In every occurrence of the `p` tag, we will create a simple 3D effect via `text-shadow`. Further elements will look smaller in perspective view, so we will make the second the `p` tag larger using `font-size` as shown in following code:

```
#wrapper p:nth-child(1) {
  font-size: 35px;
  text-shadow: 0 0 10px #fff, 0 4px 3px #ddd, 0 9px 3px
#ccc,  0 12px 1px #eee;
}
#wrapper p:nth-child(2) {
  font-size: 44px;
  text-shadow: 0 0 10px #fff, 0 4px 3px #ddd, 0 9px 3px
#ccc,  0 12px 1px #eee;
}
```

Finally, we will apply multiple transformations via rotation (`rotate` and `rotateX`) and torsion (`skewX`). And the result is a perspective view of the 3D text which looks like it is part of the space, as shown in the following screenshot. It is recommended to use a `sans–serif` font for such transformations.

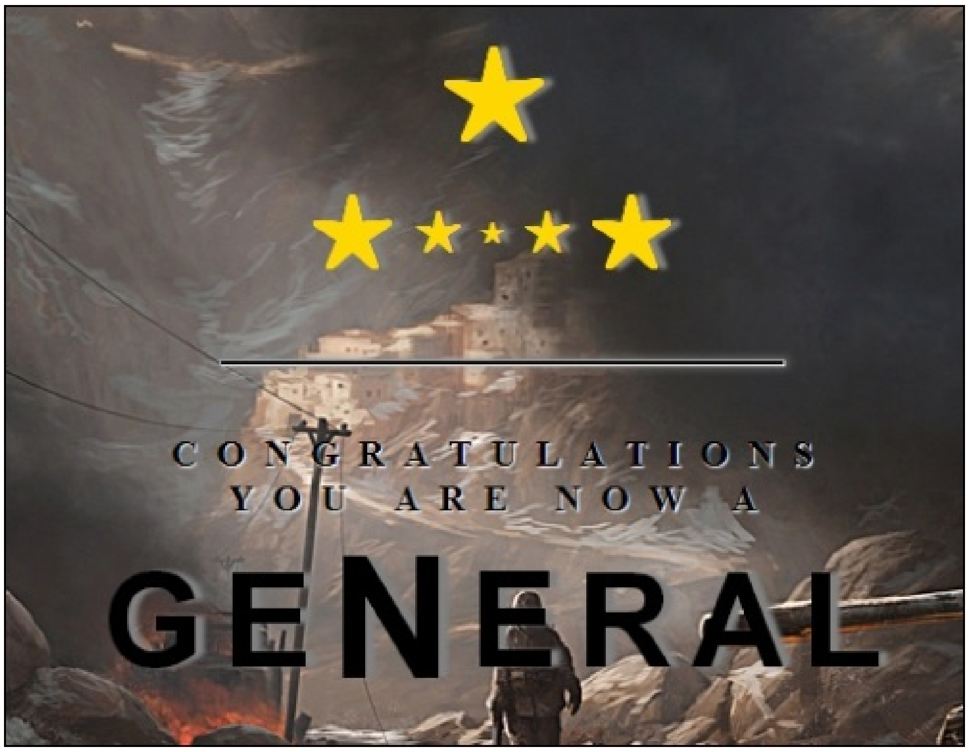


```
#wrapper {
  background: url(img/trees.jpg) no-repeat center center;
}
#wrapper p {
  font-family: Lucida, Verdana, sans-serif;
  text-transform: uppercase;
  letter-spacing: 0.03em;
  text-shadow: rgba(0,0,0,0.1) 0 20px 80px;
  transform: rotate(5.5deg) rotateX(50deg) skewX(-4deg);
  -ms-transform: rotate(5.5deg) rotateX(50deg)
skewX(-4deg-webkit-transform: rotate(5.5deg)
rotateX(50deg) skewX(-4deg);
}
```

- **Game promo**: We will now create a wallpaper that will simulate a congratulations page for a game, as shown in the following screenshot. We will use simple the `text-shadow` properties for texts and stars. We will identify later the stars by `span:nth-child()`. From outside to inside, the size of stars will decrease:

```
#slogan span{
  font-size: 20px;text-shadow: 1px 1px 1px white;
}
.tiny {
  font-size: 10px;
}
#stars span {
  color: gold;
}
#stars2 {
  margin: -20px 0;
  color: gold;
}
```

```
#stars span:nth-child(1), #stars span:nth-child(5) {
  font-size: 55px;
}
#stars span:nth-child(2), #stars span:nth-child(4) {
  font-size: 30px;
}
#stars span:nth-child(3) {
  font-size: 15px;
}
#wra span {
  text-shadow: 5px 0 3px grey;
  letter-spacing: 10px;
}
#wra hr {
  height: 2px;
  background-color: black;
  box-shadow: 2px 0 7px gray;
}
```

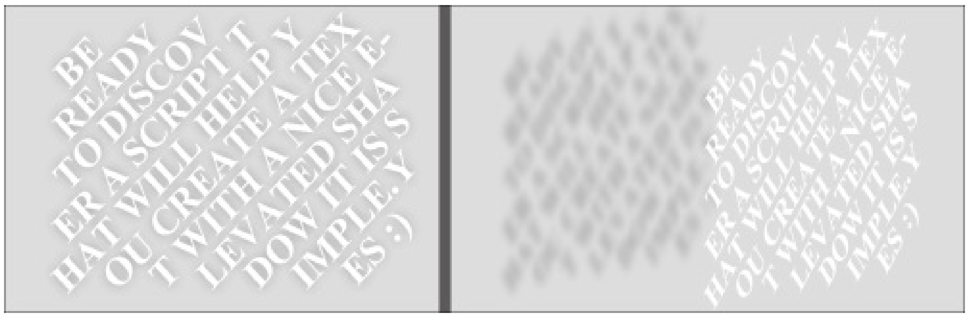


In the preceding screenshot, the **GENERAL** word is shown as a letter in every span. On hover, the letter will increase its size via the `transition` command and the `cubic-bezier` transformation:

```
#title {
  font-family: sans-serif;
}
#title span {
  vertical-align: middle;
  line-height: 1.5em;
  transition: font-size 2s cubic-bezier(0, 1, 0, 1);
}
```

```
#title span:hover {
  font-size: 90px;
  line-height: 1em;
  transition: font-size .2s cubic-bezier(0, 0.75, 0, 1);
}
```

- **Elevated writing from shadow with drag-and-drop animation**: The outcome of this part is an elevated text with a shadow dropped from about 200px. The text can be dragged-and-dropped along with its shadow as if it was a cube, as shown in the following screenshot:



To do so, we need first to download both `behold-dropshadows.js` from http://v2.desandro.com/articles/behold-dropshadows and `modernizr.js` from modernizr.com and embed them in the `script` tag:

```
<script src="behold-dropshadows.js"></script>
<script src="modernizr.js"></script>
```

We will now create an overall `div` (`poster`) and we will make it support multiple boxes in a shared 3D space via the `preserve-3d` option. **Modernizer** generates a class name, `csstransforms3d` on every `div`, if the browser handles 3D transformation. Only on `div` with class name `csstransforms3d` the effect and the JavaScript action will be applied. A `-webkit-transform` property via a `–webkit-transition` property will be then adopted on the `reset` status for these containers identified by their new class name. Both the main `div` and its shadow will receive a `transtate3d` property to create the height effect:

```
#posters {
  -webkit-transform-style: preserve-3d;
  -webkit-transform: translateZ(-200px)
}
.csstransforms3d #posters.reset {
  -webkit-transition: -webkit-transform .5s;
}
#behold {
  color: #FFF;
  -webkit-transform: translate3d( 0, 0, 200px);
}
#shadow {
  color: hsla(0,0%,0%,0);
  text-shadow: 0 5px 10px hsla(0,0%,0%,.2);
  -webkit-transform: translate3d( 0, 0, -200px);
}
.csstransforms3d #shadow {
  text-shadow: 0 0px 30px hsla(0,0%,0%,.15);
}
```

Now we will create an indent on the spans of posters. These spans are detected via the `span:nth-of-type()` property. And finally, we will apply a rotation of 45 degrees via `rotate` method's `transform` property:

```
  #posters span:nth-of-type(1)  { left: 8.8em; }
  #posters .rotor-z {
    -webkit-transform: rotate(-45deg);
    -moz-transform: rotate(-45deg);
     -o-transform: rotate(-45deg);
    transform: rotate(-45deg);
  }
```

The JavaScript will start its behavior via the `docReady` function. It will generate a `shadow` div cloned from the main `div`. When making a drag action over poster div, we will trigger the function `handleCursorStart` that will move the object via `handleCursorMove`. When we drop the element, the `handleCursorEnd` function will bring the object to its original position.