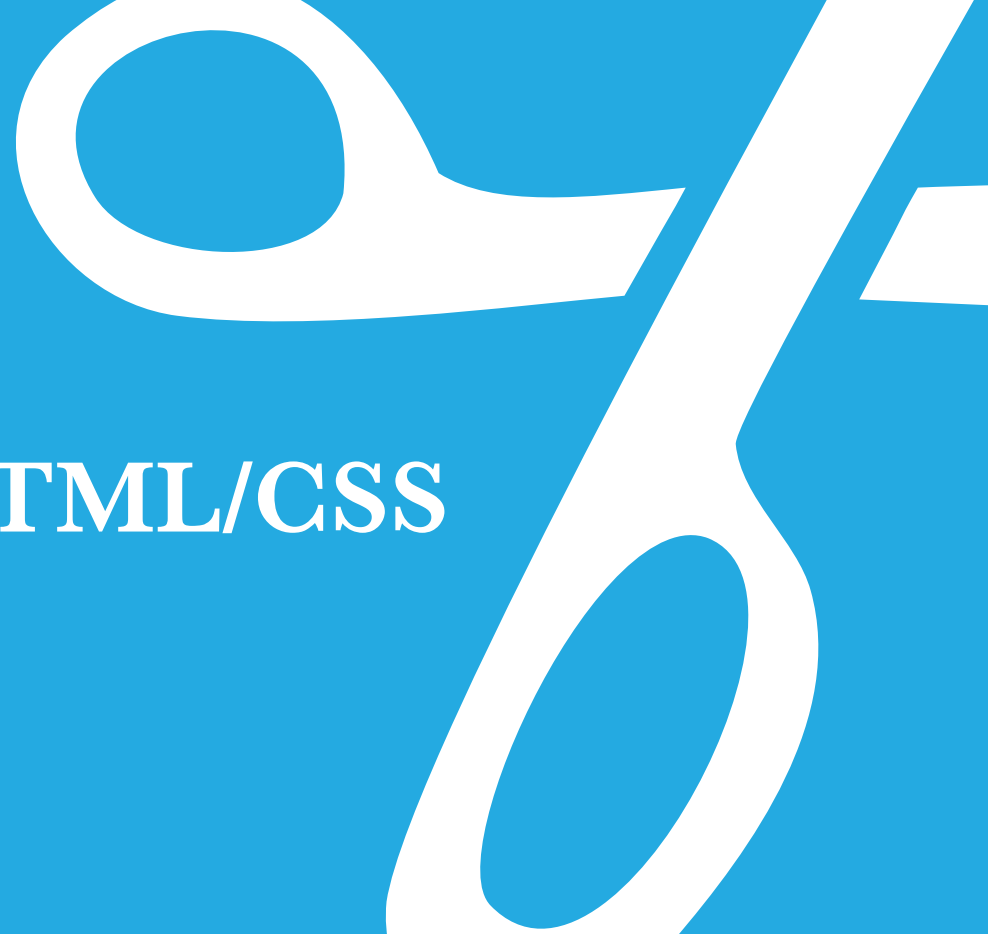


HOW-TO:

Convert PSD to HTML/CSS

The definitive guide for converting
Adobe® Photoshop® designs into HTML/CSS.

Artmov∞

R.002

CONTENTS

This guide is for you.

It's for you if you want to learn how to convert a design, made with Adobe Photoshop, into a HTML/CSS document.

It's a step-by-step guide that explains (mostly) everything you need to know to get started.

You should already be familiar with basic concepts of HTML & CSS (e.g. *display*, *float*, *etc.)* and so on.

Some basic Photoshop knowledge is required too.

It comes packed with a ready Photoshop design, called *Monoplate* - exclusively made for this guide.

The converted files of *Monoplate* are available.

There's also a blank folder of HTML & CSS files that you can start with and implement *Monoplate*, side by side, along with this guide.

The blank folder already has the framework that is being used in this guide called *F2fw*.

F2fw is a free and open-source framework built for converting designs: *http://artmov.com/f2fw*

Just to recap, your package contains these:

- the Photoshop design, *Monoplate.psd*
- the converted HTML/CSS files of *Monoplate*
- a blank folder of HTML/CSS for learning along with this guide, side by side.

What do you need before you start:

Text editor
You can use any text editor you're familiar with.

TextMate for Mac OS X or *Notepad*++ for Windows.

Adobe Photoshop CS5.5
You need Photoshop CS 5.5 to open *Monoplate.psd* design file.

Web browser

Google Chrome is a good and fast browser that you can use for this guide, if you don't have it yet.

Screenshots from these pages are from *Google Chrome*.

Start.

Analyze

Analyzing the design before getting to work

Before you start converting a design it's important to make an analysis on how that design, made in Adobe Photoshop, was done.

Imagine that the converting process is like a lego game that has been already built by a kid. You need to break that fnished piece into smaller ones.

The basic concept of a converting process is to take any given design, break it down into smaller pieces and see which of those can be reproduced with CSS properties or which can't, and so on.

You start by taking a look at your design and ask questions, such as:

- which parts can be made with CSS properties?
- which parts can't and needs cropping?

Open your Photoshop with the *Monoplate.psd* file and do just that for a few minutes.

It's going to be a good start if you ask these type of questions (*what can be reproduced with CSS properties or not*) before you write any HTML or CSS code.

This step gives you a higher speed when you start to write the code since you already have a check list to start with. You write faster, better and your project is going to be easier to manage.

Take a few moments, do a quick analysis over the design and decide, from the very beginning, which and what.

> Take your time. Have a look at your design. Try to see and understand how it was made and what you can break into smaller pieces to start with.

Also, don't neglect the small details. In the end, they are the ones to make a huge difference.

In more familiar terms, check out and identify parts such as: header, footer, menu etc. These are the most common elements found on a web page.

See how they are made, what they are made of (e.g. *borders, gradient backgrounds, etc.*) and gather just enough information to have the bigger picture.

> Take a look at your design before you start to work; analyze it, gather your information, and then start.

At this stage try to focus on the big parts, rather on the small details. Don't forget those, but leave them for later.

In *Monoplate* case have a look at how the header is done: a left-to-right black-gradient bar, no?

How is the big picture going to look?

When you are at this step, ask:

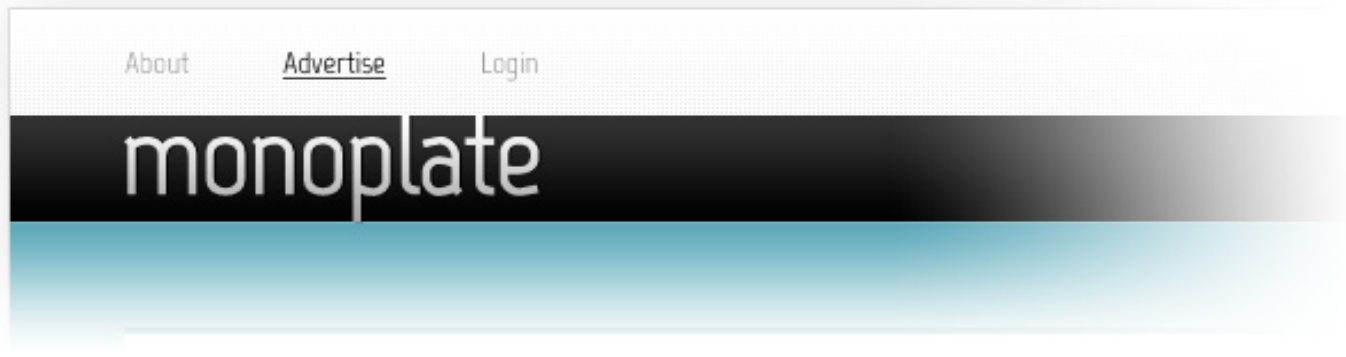
- which elements can be reproduced with CSS?
- which can't and I need to crop them?

You can also find a mix between portions that can be done with CSS properties and portions that needs cropping (e.g. *custom buttons*).

Start by doing this analysis on *Monoplate*.

1 You should already have *Monoplate.psd* open. Zoom in at the top part of it, the *top bar* part.

This is where you start from.



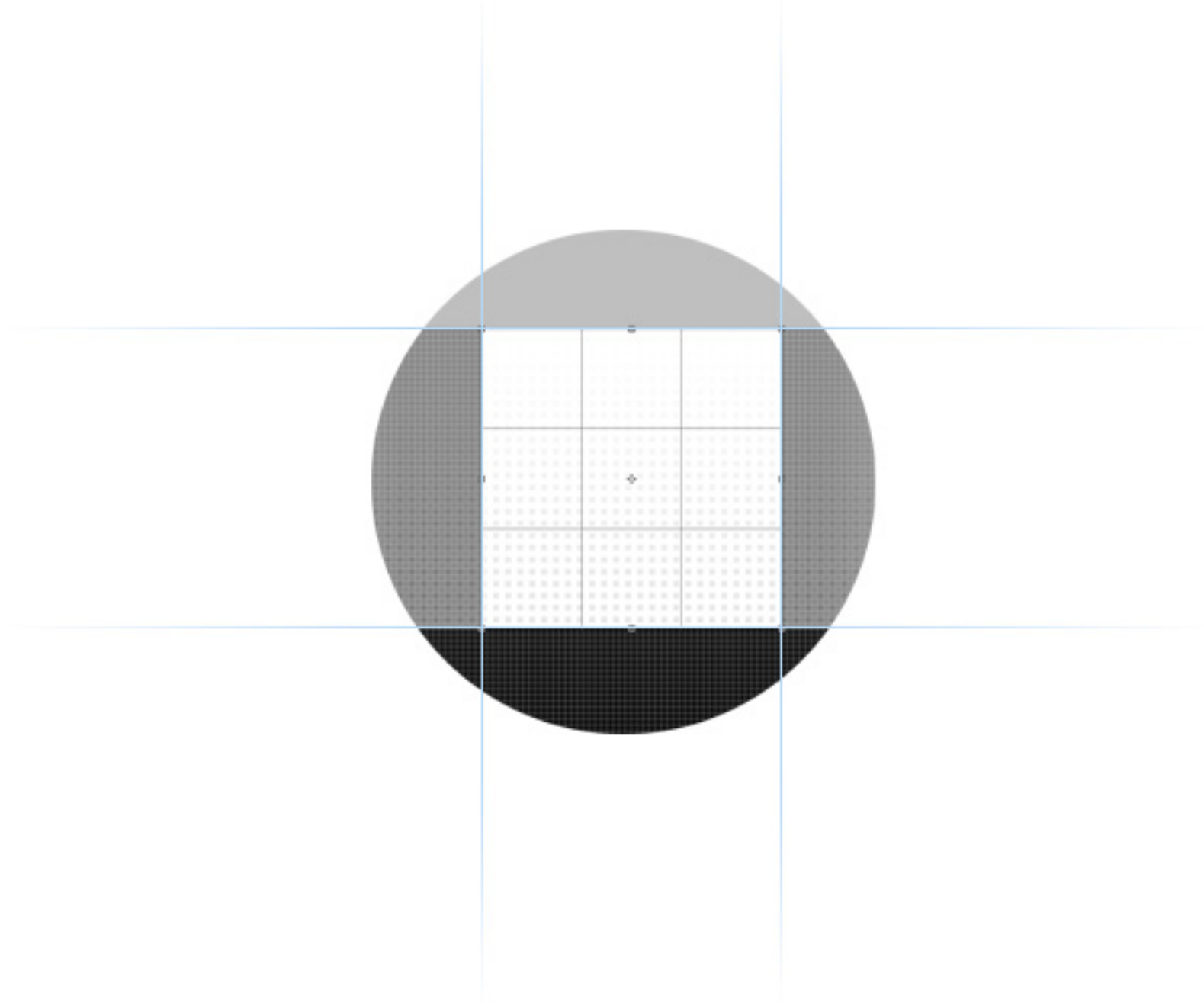
Look at the top part (*#top*) and you'll notice that the background has dotted gradient lines. Crop this into a new image: *top.png*.

You can choose any selection to crop from, but try to get its minimum functional size (e.g. *without any graphical interruptions when its being repeated*).

A 50x50 pixel selection is good for this background, but a 2x40 pixel one is much better. It includes the 2px width as the minimum size required to have the image properly shown when being repeated.

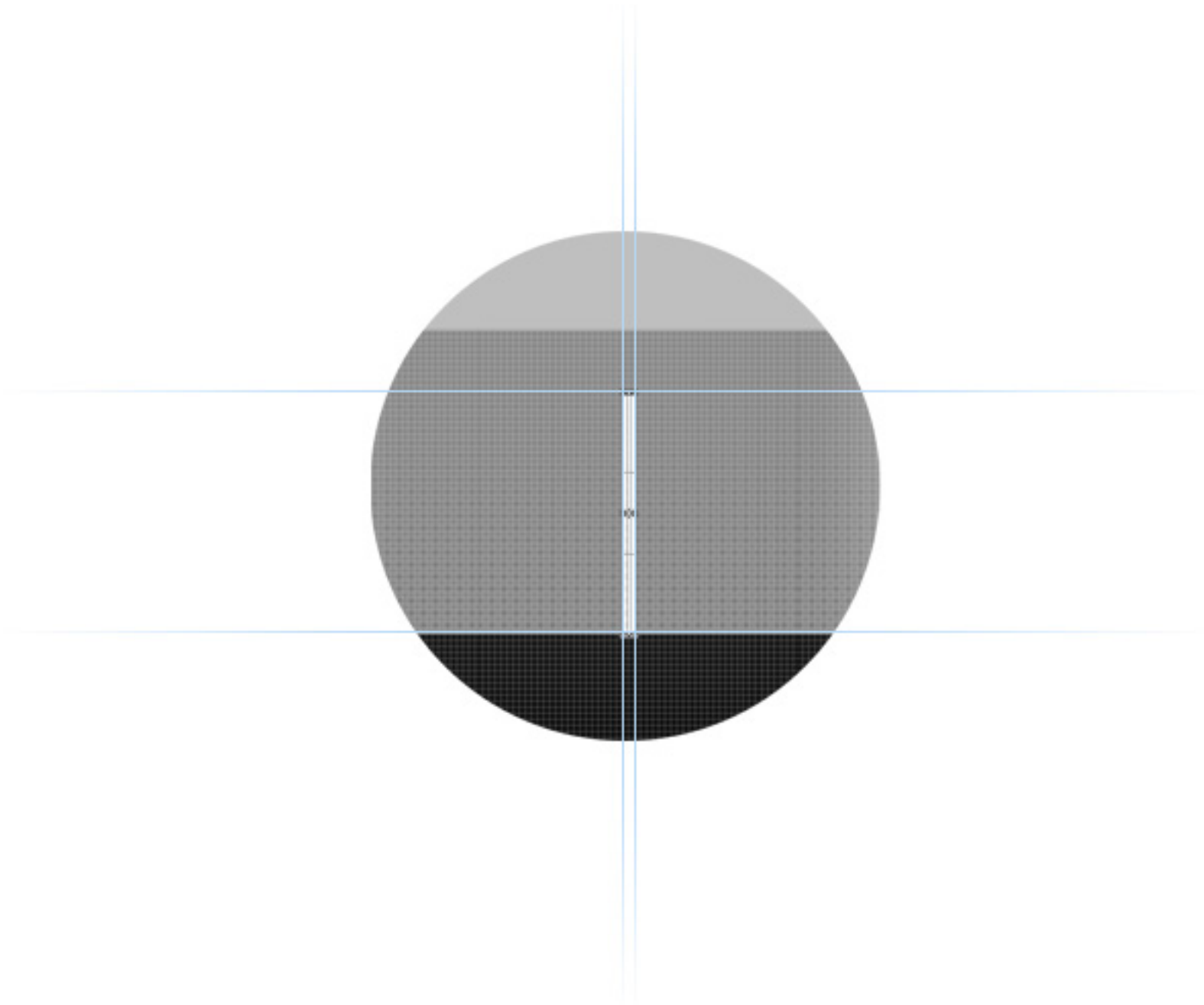
> Crop any images to the most minimum functional size, both vertical and horizontal, in order to reduce its size as much as possible.

Have a look at the next comparison.



This is a random cropped section of the top bar of *Monoplate* at a 50x50 pixels selection.

It looks good enough, but its size can be improved even further if you had a 2x40 pixels selection.



Much better. This is a 2x40 pixels crop, a portion of the background trimmed to its minimum.

It's functional and *top.png* has a smaller size now.

Minimizing a crop selection of any given image ultimately provides you with much lower file sizes, resulting in a more lightweight web page.

Depending on multiple factors, it can be a save.

In *top.png* case, a 50x50 crop selection resulted into a size of **231 bytes**, while a 2x40 one gives you **142 bytes**. It was a save of 89 bytes.

2 Back to Photoshop: save the newly image as *top.png;* this image is in the */includes/images* folder.

You don't write any HTML or CSS code for now, you are just doing an analysis over the design and crop some parts of it.

Save it and move on the next part of *Monoplate*.

You can find all the images from *Monoplate* in the */includes/images* folder; the stylesheets are in the */includes/css folder*, and the thumbnails from the content are in the */includes/pictures*.

3 The header: it has a logo, a menu and a bar from left to right with a black gradient. The logo and the menu can be discussed later.

What's the most effective solution for implementing the header background? What are your options?

- crop the header's black gradient background;
- reproduce the background with CSS3 gradients.

By far the best decision for these types of questions strictly depends on your project requirements.

Each project is different and each has different needs on compatibility, speed and so on.

A few questions that may help you:

- is speed more important than multi-browser compatibility ?
- is the header height going to change ?
- and so on..

Other questions that might come in handy:

?

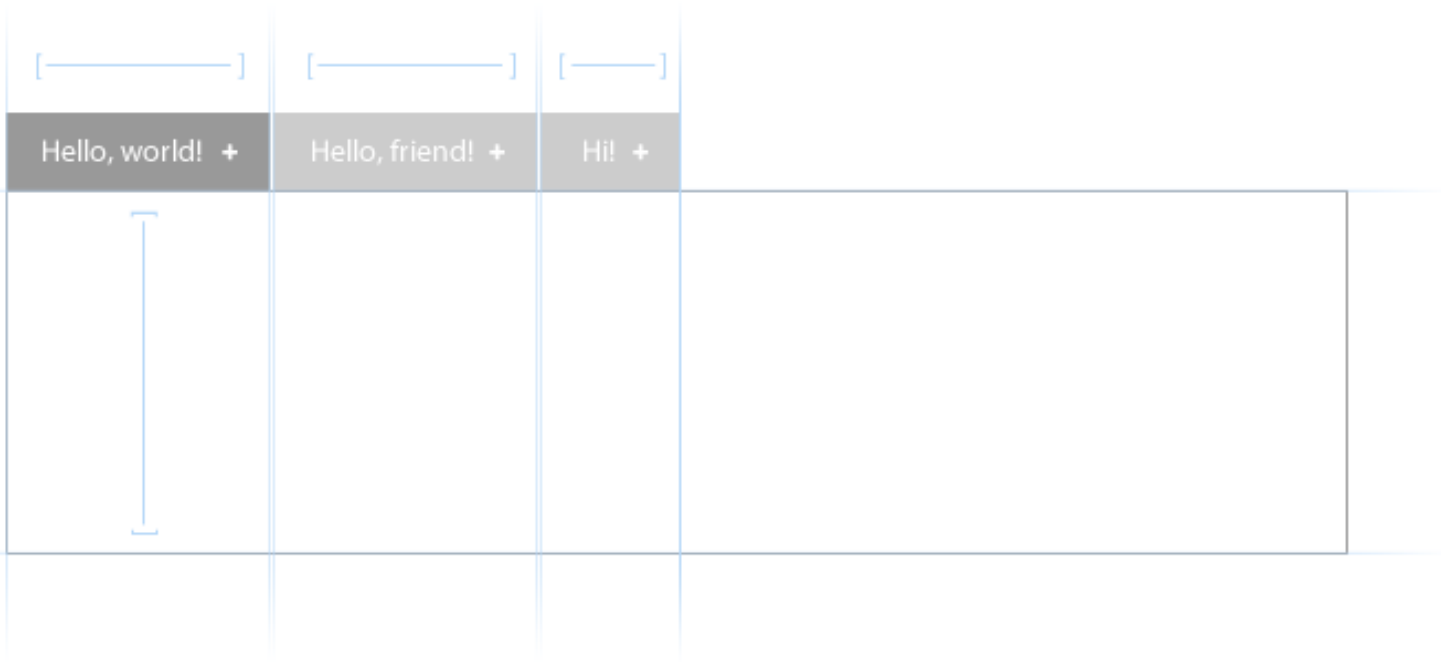
Is the element going to be with dynamic content ?
Is the element size going to change - how ?
What about the height, width or both ?

When you find an element with no upfront requests about it, try to assume what it is and what is it for.

Here’s a common example for that.

The title boxes of a tab component will most likely need to have a dynamic width to allow the title to expand as needed, while the content box may require a dynamic height.

A mock-up:



Back to the *Monoplate*: next is a quick overview of the two options you have for the *#header*.

The first option is to crop the background directly, just like you did with *top.png*.

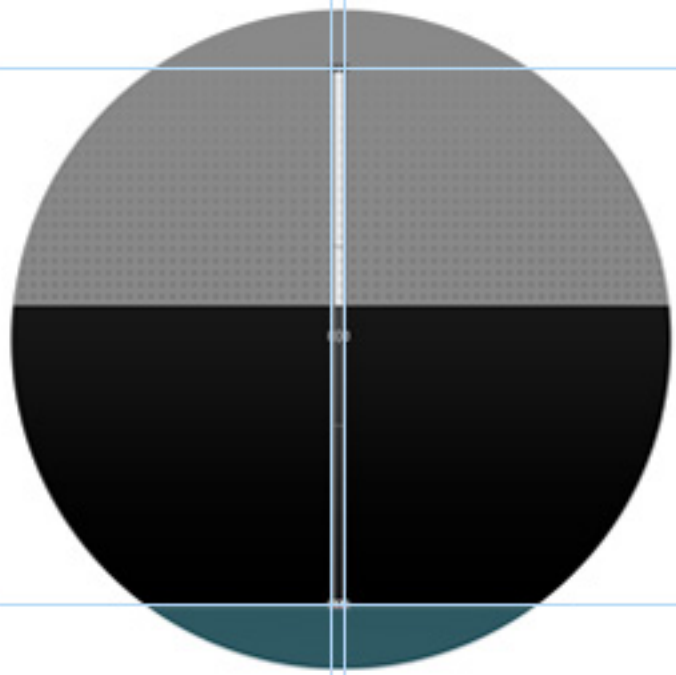
If the requirement is to have the highest level of multi-browser compatibility, cropping is the solution (e.g. *gradient is shown on all supported browsers*).

Let's have a glimpse at this example where the height of the header remains unchanged (*fixed*).

The previous crop you did was the top background, trimmed to its minimum functional size.

You can apply the same approach here and save a *header.png* file, but you'll basically have two files then: *top.png* and *header.png*.

If the top bar and the header stays fixed in height, have those images combined to save on the page size and reduce the number of resources to download.



Like the *top.png* image, this new image is cropped to its minimum size needed.

It has the 2 pixels width for the top part and the height needed to include the header background.

The second option is to reproduce the gradient with CSS3 functionality.

Speed is much more important for *Monoplate* than browser compatibility, so make the gradient with CSS3 gradient (*more in Chapter 3; no coding for now*).

This reduces the page size by excluding a selection inside *top.png*, it keeps a low size. Anytime you can reproduce elements using CSS, it saves you both development time and page size.

4 Move on to the next part: the content itself, named *#page* in the *Monoplate* converted HTML.

The *#page* gradient is going to be reproduced with CSS3, but you can also choose to append it to the *top.png* image.

Monoplate is going to use the CSS3's variant to make the web faster.

More about this in Chapter 3.

> When you analyze any given design, try to indentify parts of a element (e.g. *buttons, menu, etc.*) that can be easily reproduced using CSS properties.
>
> - can it be fully reproduced using CSS?
>   (e.g. *buttons with shadows, gradients and so on*)
>
> - if not, what parts of it can be reproduced?
>   (e.g. *buttons with custom background, but 1px border*)

Identity as much as possible what you can do with CSS properties.

This saves you time.

5 You identified the main parts of the *Monoplate* and decided what to crop (*top*) and what you can do with CSS (*header*, *page*).

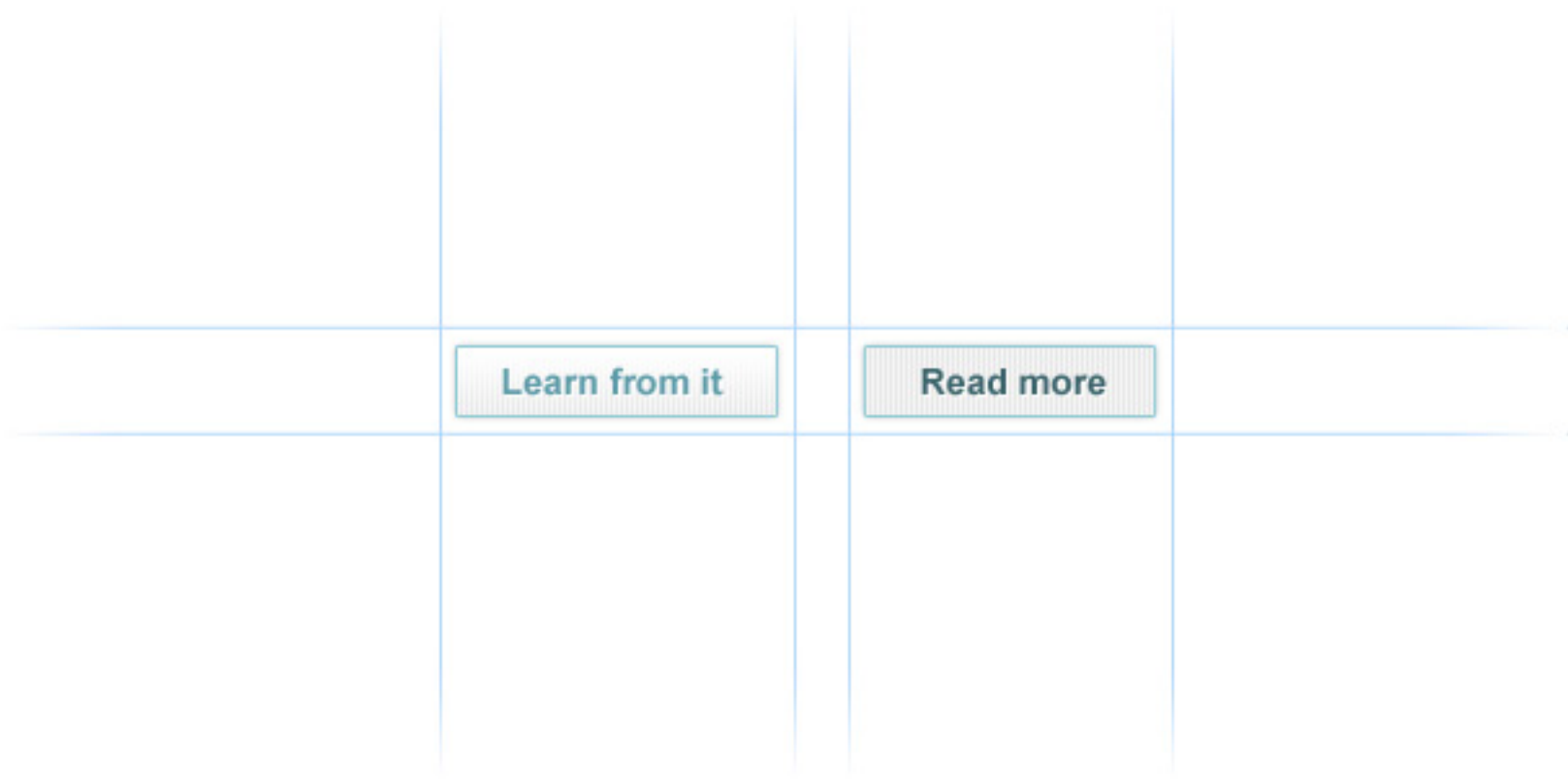
Another part of *Monoplate* are the custom buttons.

You can either analyze them later before you write the HTML/CSS code for that part or do it now.

Actually, have a look at them now.

Go with an in-depth check on the custom buttons and see which parts of a state (e.g. *default*, *:hover*) can be made with CSS and which needs to be cropped.

Divide a button into smaller pieces to see what makes it a button.



*These **<u>can be done</u>** with CSS:*

The button text shadow

The button shadow

The button border

*These **<u>can't be done</u>** with CSS:*

The lines gradient background

This comparison should help you to find a balanced method of converting these buttons, using a mix of CSS properties and cropped images.

Once you identified that (as above), go to deactivate those effects from the Photoshop Layers Panel and leave only what needs to be cropped.

So, the steps applied to these buttons would be:

1. Remove the text

Remove the "*Learn from it*" and "*Read more*" text to have the two button states ready (*default and hover*).

Move these buttons into a new PSD document or remove some of the noise around it (e.g. *pictures, text etc.*) so you can stay focused on the buttons.

Moving the buttons into a new document is simple to do: select the buttons and hold click while your drag them into the new document.

2. Remove the borders

The buttons have a solid border that can be done with CSS, so you can remove it.

3. Remove the shadows

The shadows can be entirely recreated with CSS3, so remove them as well.

Almost done. What are you left with now?

What you have left now with is from the column of "*what can't be done with CSS*": the background.

4. Stack the backgrounds

Stack the backgrounds one after another to allow a dynamic change of the text inside each button.

"*Learn from it*" to easily be "*Learn from Monoplate*" without having the background interrupted.

If you had them positioned otherwise the width of the *default* state would have been limited up until it reached the *:hover* background.

Because the width of the button is unknown and can change anytime, go with a dynamic width.

Just as with *top.png* trim this background image to its most minimum size you can.

The background can be cropped up to a maximum width of 2 pixels, the minimum size you need to have it repeated it properly.

5. Save the file

Save the new cropped image as *btn.png*. You can find it in the */includes/images* folder.

There has been about five steps that you did for the *Monoplate* analysis. From making a quick overview of its structure to deciding which & what to cropping 2 images also.

The steps you did so far:

- you did an overview of *Monoplate* design to grasp the big picture;

- you identified and decided which parts of *Monoplate* can be reproduced with CSS;
- you also identified elements that can be recreated with a mix of CSS properties and cropped images;
- you have trimmed the necessary images.

Congratulations!

This chapter showed you what to ask before you convert a new design, see which parts needs to be trimmed or which can be easily done with CSS.

You can crop all the necessary images straight from this step (e.g. *social icons, logo etc.*), but its purpose is to give you the start idea and what you should ask when you've just opened a .psd file to grasp the big picture of it.

Now it's time to write some code.

Write

Getting to work

Before writing the HTML and CSS, just recap the structure of *Monoplate* and divide it into your future containers (e.g. *IDs and classes*). There is no need to write them in *index.html* now.

It would (probably) look something like this:

#top
    links
    icons
#header
    logo
    menu
#page
    hello
        article
        slider
    albums
    blog
#footer

The naming was done fairly simple: when you did the analysis of this design, there was a top bar part of *Monoplate*, so that takes the *#top* container; the header has the *#header* and so on.

Usually the containers are named after what they represent or contain (e.g. *div.icons for the top bar's right icons*), but you can choose and develop your own personal system for naming.

Or you can find existing systems.

Getting used to a naming system improves your converting skills. You will work faster each time.

It gives you a more accelerated rate of coding any HTML/CSS and it also improves the maintainance of your projects.

It's time to write some HTML and CSS.

You'll be switching regularly between the text editor, Photoshop and your browser of choice.

So with *Monoplate.psd* open, load the blank folder from your package to start writing some HTML and CSS.

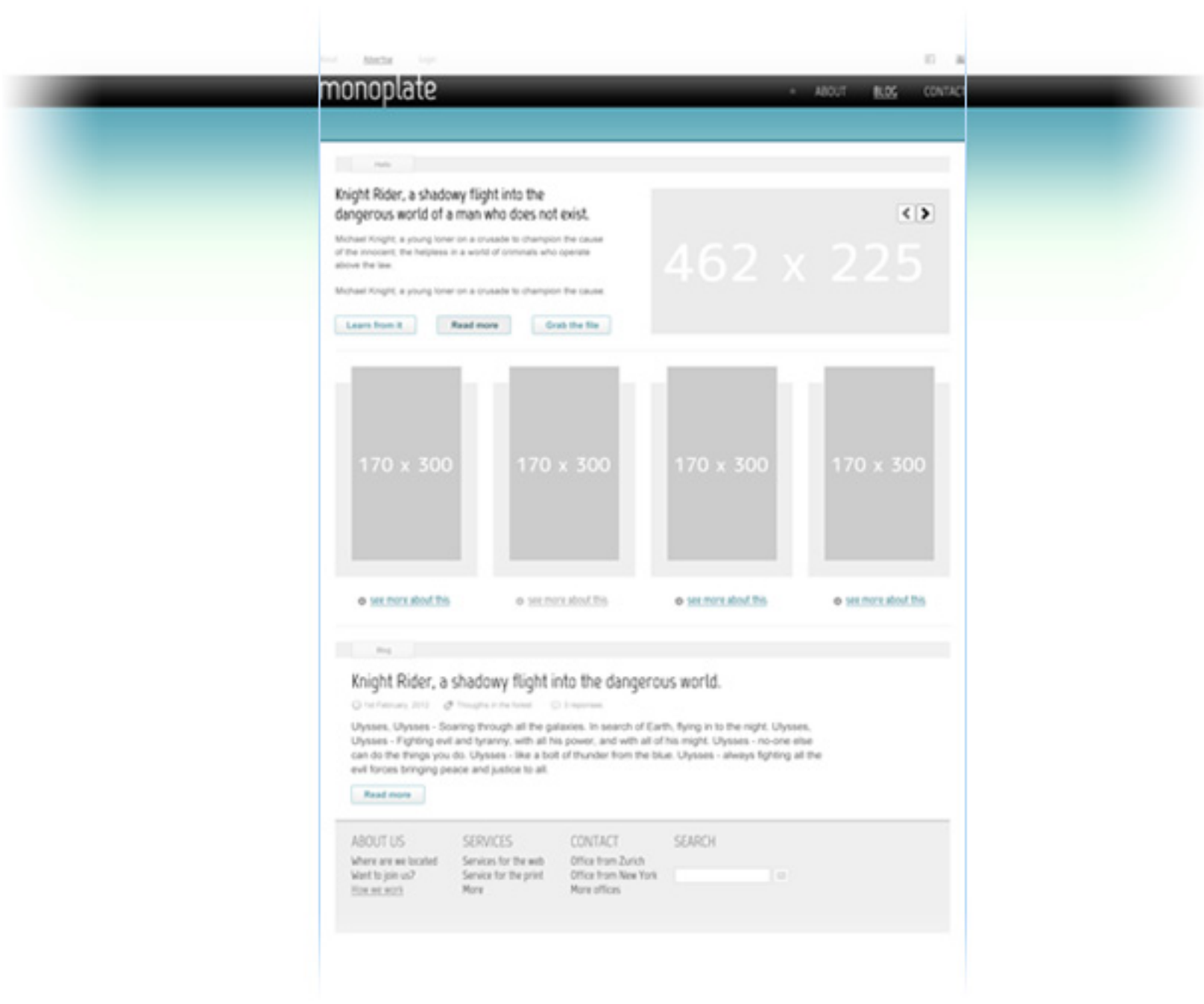
You already have the converted files of *Monoplate* in the same package, which you can use whenever you need to double check that you're on the right track.

The *index.html* and *style.css* files are what you need for now. The *style-ie8.css / style-ie7.css* - specific stylesheets for Internet Explorer - are less important now.

You start converting in one browser and move after it's done to the other web browsers for testing and debugging.

Start with your content container to define its width in your CSS.

You can measure the width with the *Marquee Tool*.



Monoplate content container is 1000 pixels wide, so limit the container to that size, 1000px.

1 Define the content container in *style.css*

```
div.container {
     margin:0 auto; /* makes the content centered */
     width:1000px;
}
```

The *#top*, *#header*, *#page* backgrounds are at 100% width, outside the 1000px limit, meaning that those backgrounds will stretch from left to right in the browser's window, regardless of inner content you have (*see the previous image*).

Go for a *div* with class to call it several times within the document. An ID *div* can be called only once in the HTML page.

Switching to the *index.html* file, the converting process starts with the top bar of *Monoplate*, *#top*.

2 Code the *#top* selector

You've recently cropped *top.png* as the background image for *#top*, so it's time to show that.

Start with the HTML first:

```
<div id="top">
    <div class="container">

    </div><!-- end .container -->
</div><!-- end #top -->
```

The *div.container* is part of *#top* to center the content, while its parent, *#top*, is to display the background.

That's because the *#top* is going to be 100% in width, which is what you need to display the *top.png* background for the top bar, from far left to far right.

Apply the *top.png* image to the *#top* selector.

The "*repeat-x*" translates in the background being repeated from left to right, while "*bottom left*" gives the starting position as the bottom left.

```
#top {
     background:url(../images/top.png) repeat-x bottom left;
}
```

Check the results to see if *top.png* is repeated. You may need to add some dummy content inside the HTML for *#top* background to show.

Use either blank *<p>* or simply *
* tags.

Save the *index.html* and *style.css* files and refresh your web browser. How does it look?

Move on the next page and see what are the steps to complete the *#top* selector.

Switch back to *Monoplate* design and you'll see that the paragraphs inside the top bar are positioned on the left and respectively, on the right.

So make use of the *F2fw* reserved classes for floating elements: *fl* and *fr*.

```
.fl {
    float:left;
}
```

fl (*floatleft*)
Floats elements on the left, regardless of type.

```
.fr {
    float:right;
}
```

fr (*floatright*)
Floats elements on the right, regardless of type.

The *fl* and *fr* classess can be applied to any element.

For the reserved parts that are from *F2fw*, this guide outlines their properties to make it easier for you to understand it and see how it works.

As said earlier, apply the *fl* and *fr* classes to each *p* inside the *#top* container and add the content.

```
<div id="top">
     <div class="container">

          <p class="fl">
               <a href="#" title="#">About</a>
               <a href="#" title="#">Advertise</a>
               <a href="#" title="#">Login</a>
          </p>
          <p class="fr">
               <a href="#" title="#">Facebook</a>
               <a href="#" title="#">Twitter</a>
          </p>
          <div class="cl"></div>

     </div><!-- end .container -->
</div><!-- end #top -->
```

The *cl* class is from *F2fw* and it is used to clear floats. It simply has the "*clear:both*" property.

Here is a quick recap of these classes:

- the *fl* for left floating;
- the *fr* for right floating;
- the *cl* for clearing the floats.

```
.fl {
    float:left;
}
```

fl (*floatleft*)
Floats elements on the left, regardless of type.

```
.fr {
    float:right;
}
```

fr (*floatright*)
Floats elements on the right, regardless of type.

```
.cl {
    clear:both;
}
```

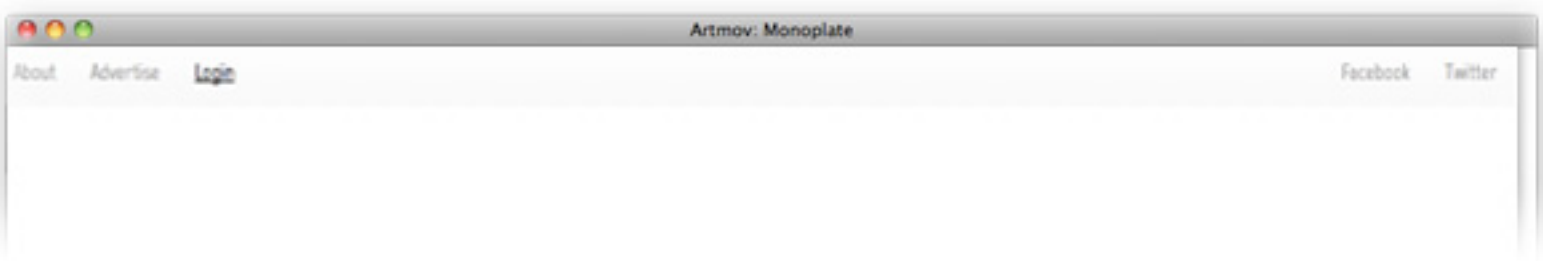
cl (*clearing*)
Clears any above elements, both on the left and on the right.

Switch back to *style.css* and begin the styling of the paragraphs with fonts, colors and margins.

Selecting the text (e.g. *link*) inside Photoshop will show you the font used and the color.

```
#top p {
     font:normal 14px/18px “Marvel”, Arial, sans-serif;
     padding:12px 0; /* top and bottom space */
     margin:0; /* disables any margin */
}
#top p a {
     text-decoration:none; /* disables the underline */
     margin:0 20px 0 0; /* adds the space between links */
     color:#a0a0a0;
}
#top p a:hover {
     text-decoration:underline; /* enables the underline */
     color:#000;
}
```

Save and refresh. This is how it should look:



If yes, you're on the right track now.

Your next step now is to add the icons, but let's see how *F2fw* can help you with that.

First, update your *index.html* file with the reserved class called "*ico*".

```
<div id="top">
     <div class="container">

          <p class="fl">
               <a href="#" title="#">About</a>
               <a href="#" title="#">Advertise</a>
               <a href="#" title="#">Login</a>
          </p>
          <p class="fr">
               <a href="#" title="#" class="ico ico-facebook">Facebook</a>
               <a href="#" title="#" class="ico ico-twitter">Twitter</a>
          </p>
          <div class="cl"></div>

     </div><!-- end .container -->
</div><!-- end #top -->
```

Also include a "*ico-facebook*" and "i*co-twitter*" class to differentiate the icons.

The "*ico-facebook*" and "*ico-twitter*" classes aren't part of *F2fw*, but you do need them to target the icons and make use of that.

Follow up on the next page for an example, while below you can see which properties a "*ico*" class brings from *F2fw*.

```
a.ico {
    background:url(../images/ico.png)
               no-repeat 0 0;
    color:transparent !important;
    text-indent:-9999px;
    cursor:pointer;
    overflow:hidden;
    display:block;
    line-height:0;
    font-size:0;
    border:0;
}
```

ico (*icon*)

Loads a *ico.png* from the */includes/images* folder.

Disables content within element with class applied.

The "*ico*" makes it easier to include icons without rewriting the above properties each time.

F2fw also has the "*btt*" and the "*arr*" classes with the same properties (*calls their own images: btt.png, and arr.png*). These are used usually for custom buttons (*btt*) and arrows (*arr*).

3 Back to Photoshop: social icons

Switch back to *Monoplate*, go and grab the social icons and export them into the a *ico.png* file to make use of the "*ico*" class.

The icons have a 40% level of opacity for the *default* state, while for the *:hover* state it's at 100%.

You can duplicate the icons, so the first copy to be at 40% opacity, while the second to be at the 100%.

But you can also save them at 100% opacity and do the 40% level from CSS directly, with the *opacity* property, thus excluding the duplicate.

How the social icons looks inside a new document - to be saved as a *ico.png* image file.

Before saving the *ico.png* image, get each icon size and background position.

These social icons from *Monoplate* have a 16px width, with the same 16px for the height.

Switch to *style.css* and write them down there. The positions are:

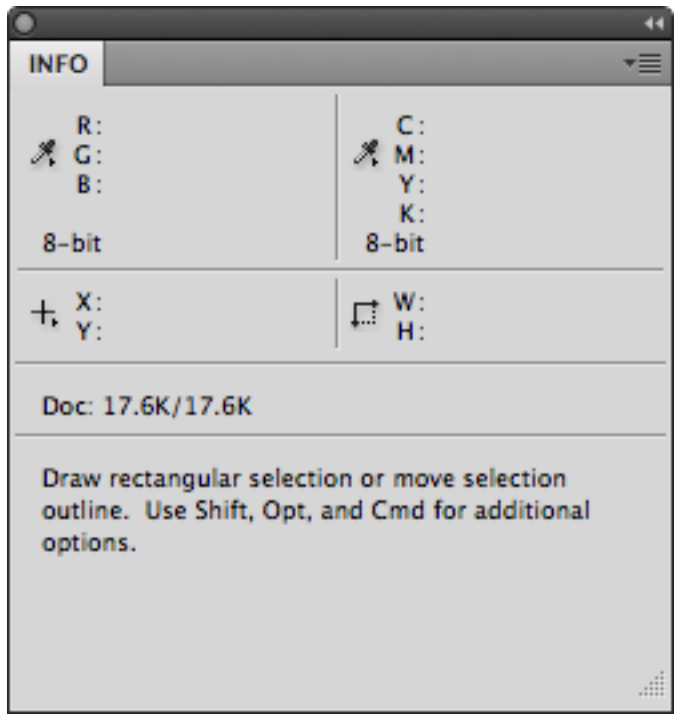
- Facebook icon position is *0 0*;
- Twitter icon position is *-16px 0*.

If simply selecting them doesn't work (e.g. *using the Move Tool*) use the *Marquee Tool* to get the width, height and their position.

The output information is available in the Info panel of Photoshop.

Example of using the Rectangular Marquee Tool to select a portion of an image:

The below is a screenshot of the Info panel in the Photoshop. This panel gives you the X, Y positions of a selection, as well as the width and height.



If you haven't already, switch to *style.css* and write the size and position. The HTML already has the "*ico-facebook*" and "*ico-twitter*" classes added.

Both icons already share the "*ico*" class, you don't need to load the *ico.png* twice or add any properties that disables the content.

It's already there:

```
a.ico-facebook,
a.ico-twitter {
     height:16px;   /* height of social icons */
     width:16px;    /* width of social icons */
}
a.ico-facebook {
     background-position:0 0;     /* position of Facebook in ico.png */
}
a.ico-twitter {
     background-position:-16px 0; /* position of Twitter in ico.png */
}
```

The icons already have the same width and height, so they're grouped together: *a.ico-facebook*, *a.ico-twitter* and then called individually to include position of background.

Once *style.css* is saved, the *index.html* file will show the icons stacked one after another because there is no *float* property added.

This happens because each "*ico*" class is going to be rendered as a block element since that class comes with a *display:block;* property included.

So the next step is to add the *float* property for the Facebook icon only and the space between icons.

What other options can you have here? Maybe the "*display:inline-block;*" property?

Your new updated HTML would be:

```
<div id="top">
     <div class="container">

          <p class="fl">
               <a href="#" title="#">About</a>
               <a href="#" title="#">Advertise</a>
               <a href="#" title="#">Login</a>
          </p>
          <p class="fr">
               <a href="#" title="#" class="ico ico-facebook fl">Facebook</a>
               <a href="#" title="#" class="ico ico-twitter nmr">Twitter</a>
          </p>
          <div class="cl"></div>

     </div><!-- end .container -->
</div><!-- end #top -->
```

You already know what *fl* does, but you could also apply the *float* property to the *a.ico-facebook* directly from CSS without adding a *fl* class in the HTML.

The *nmr* (e.g. *no margin right*) class is new. It's there to disable the margin right of the Twitter icon.

The next page details more about this class.

When you styled the paragraphs earlier, you also spaced the anchors between them by adding a right margin.

To cancel that space after the Twitter icon (e.g. *its right margin*), use the "*nmr*" class from *F2fw*.

```
.nmr {
    margin-right:0 !important;
}
```

nmr (*nomarginright*)
Cancels right margin.

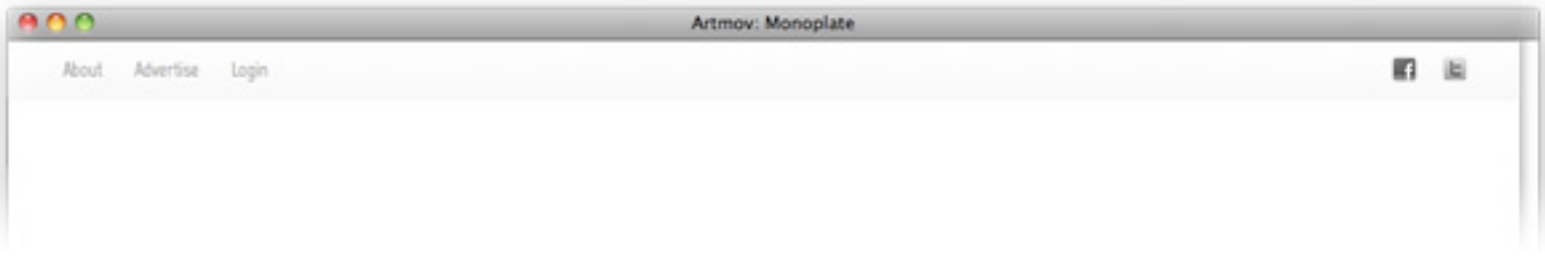
Other similar classes that *F2fw* gives you are:

- **nml** (*nomarginleft*), **nmt** (*nomargintop*), **nmb** (*nomarginbottom*)
- **npl** (*nopaddingleft*), **npt** (*nopaddingtop*), **npr** (*nopaddingright*) and **npb** (*nopaddingbottom*)
- and so on..

Good.

It's time to save your files, the *style.css* and *index.html* and check the results in your browser.

Here's how it should look:



This is the *#top* converted in HTML/CSS.

4 Code the *#header*

The same idea applies to *#header*, just as it was with *#top*: the *div.container* inside your *#header* centers the content and the *#header* creates the background.

The base HTML structure for *#header:*

```
<div id="header">
     <div class="container">

          <div id="logo">

          </div><!-- end #logo -->
          <div id="menu">

          </div><!-- end #menu -->
          <div class="cl"></div>

     </div><!-- end .container -->
</div><!-- end #header -->
```

The structure is pretty simple so far, isn't it? Your move now is to add the logo and the menu.

The *#logo* and *#menu* will be positioned on the left and, respectively on the right (*also automatically by F2fw; more about this on the next page*).

The *div.cl* comes after the *#menu* and clears the floats of those containers with the *clear:both;* property.

Below are the properties that *F2fw* gives to a *#logo* and how it can help you to add a logo easily.

```
#logo {
     float:left;
}
#logo h3 {
     line-height:54px;
     font-size:25px;
     margin:0;
}
#logo h3.incl a {
     background:url(../images/logo.png)
               no-repeat 0 0;
     text-indent:-9999px;
     display:block;
     line-height:0;
}
```

#logo
Positions #logo on the left (*float:left;*).

Adds a basic styling for an inner *h3*, if any.

Includes the logo using a "*incl*" class if applied to the *h3* tag.

It automatically loads any *logo.png* from the */images* folder. You just need to set the size of the logo.

In more simple words, with the *#logo h3.incl a*, *F2fw* automatically loads the *logo.png* and you just set the sizes. It's very similiar to the "*ico*" class.

If your logo isn't in a PNG format, you can easily overwrite that rule and replace it with another one.

Something like this:

```
#logo h3.incl a {
     background-image:url(../images/logo.jpg);
}
```

It's in the PNG format by default since quite a few web sites require a transparent logo.

Like the example above, *F2fw* contains more of these goodies.

It was built as a framework for converting designs to allow developers to write less, but faster.

How about the *#menu* container? What does *F2fw* adds to a *#menu*?

See below what *F2fw* adds to it.

```
#menu {
     float:right;
}
#menu ul {
     padding:0;
     margin:0;
}
#menu ul li {
     list-style-position:outside;
     list-style-type:none;
     margin:0 10px 0 0;
     display:block;
     float:left;
}
#menu ul li:last-of-type {
     margin-right:0;
}
```

#menu
Positions *#menu* on the right with *float:right;*

Makes inner *ul* display as a horizontal menu, with the *li* being floated on the left.

Before you include the logo and the menu in the HTML, recreate the gradient background for the *#header*.

Monoplate needs speed rather than multi-browser compatibility, so the *#header* container background is going to be added using the CSS3 gradient, not another cropped image.

It's fast and easy.

It's best to use an online tool that will output for you the entire properties for adding a CSS gradient.

There are quite a few properties for targeting a wide range of web browsers, such as Webkit-based browsers (Safari, Chrome), Firefox, Opera or Internet Explorer.

The CSS gradient generator from ColorZilla is an excellent choice: *www.colorzilla.com/gradient-editor/*

Whenever you implement CSS3, make your web pages more accessible by including a fallback for those browsers that don't support it.

This usually means a solid color for the background background if the *gradient* property isn't supported.

The mighty code would be:

```
#header {
      background:rgb(51,51,51);
      background:-moz-linear-gradient(top, rgb(51,51,51) 0%, rgb(0,0,0) 100%);
      background:-webkit-gradient(linear, left top, left bottom, color-stop(0%,
                  rgb(51,51,51)), color-stop(100%, rgb(0,0,0)));
      background:-webkit-linear-gradient(top, rgb(51,51,51) 0%, rgb(0,0,0) 100%);
      background:-o-linear-gradient(top, rgb(51,51,51) 0%, rgb(0,0,0) 100%);
      background:-ms-linear-gradient(top, rgb(51,51,51) 0%, rgb(0,0,0) 100%);
      background:linear-gradient(top, rgb(51,51,51) 0%, rgb(0,0,0) 100%);
}
```

ColorZilla outputs a IE8-7 property, *filter*, but it's removed because it's buggy and not recommended.

Use a solid background for IE7-8 instead.

Chapter 3 shows the IE hack to thave the gradient background, but *Monoplate* doesn't use it.

5 Code the *#logo*

Get the logo from *Monoplate* and create a new PSD document with it. Afterwards save it as a *logo.png* image inside the */images* folder.

The HTML to include it is easy:

```
<div id="header">
     <div class="container">

          <div id="logo">
               <h3 class="incl"><a href="#" title="#">Monoplate</a></h3>
          </div><!-- end #logo -->
          <div id="menu">

          </div><!-- end #menu -->
          <div class="cl"></div>

     </div><!-- end .container -->
</div><!-- end #header -->
```

Simple, no?

A quick recap on the logo:

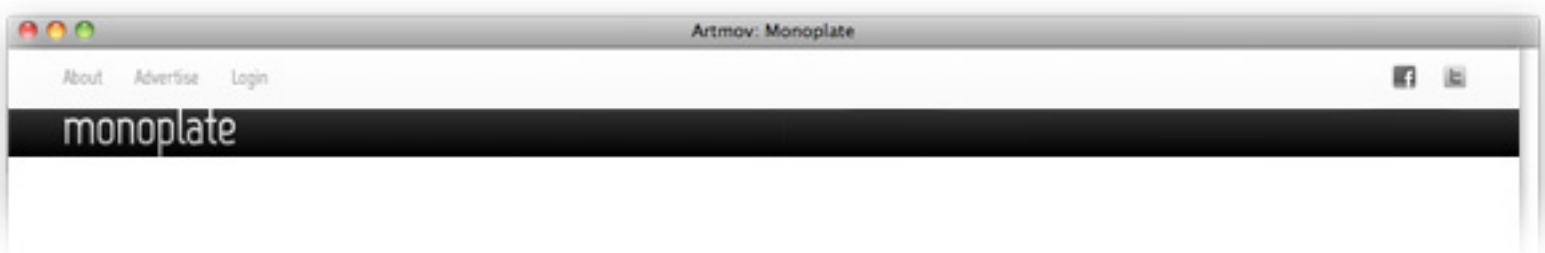
Any *h3 a* element with the "*incl*" class applied to the *h3* inside *#logo* is reserved for *F2fw* to include the *logo.png* image file from the */images* folder.

Afterwards, in your *style.css* you only set the size.

The CSS for the logo is simple as the HTML was:

```
#logo h3.incl a {
        height:51px;
        width:182px;
}
```

Save & refresh your browser to see the results:



Keep up and move on to the menu and add the links as an unordered list. (e.g. *ul*)

6 Code the *#menu*

Inside the *#menu* add an active state for one of the links (e.g. *Blog*) to style it accordingly to *Monoplate*.

The HTML code would be as this one:

```
<div id="header">
     <div class="container">
          <div id="logo">
               <h3 class="incl"><a href="#" title="#">Monoplate</a></h3>
          </div><!-- end #logo -->
          <div id="menu">
               <ul class="cf">
                    <li><a href="#" title="#">About</a></li>
                    <li class="active"><a href="#" title="#">Blog</a></li>
                    <li><a href="#" title="#">Contact</a></li>
               </ul>
          </div><!-- end #menu -->
          <div class="cl"></div>
     </div><!-- end .container -->
</div><!-- end #header -->
```

Quite simple. The "*cf*" (clearfix) class is also used for clearing floats. It's the well-known "*clearfix*" hack for auto clearing.

Read about the *clearfix* here:
www.positioniseverything.net/easyclearing.html

It's added to clear the floats of *li* items in the menu, without adding a "*cl*" class at the end of the *ul*.

Here is how a "*cl*" alternative would look:

```
<div id="header">
     <div class="container">
          <div id="logo">
               <h3 class="incl"><a href="#" title="#">Monoplate</a></h3>
          </div><!-- end #logo -->
          <div id="menu">
               <ul>
                    <li><a href="#" title="#">About</a></li>
                    <li class="active"><a href="#" title="#">Blog</a></li>
                    <li><a href="#" title="#">Contact</a></li>
                    <div class="cl"></div>
               </ul>
          </div><!-- end #menu -->
          <div class="cl"></div>
     </div><!-- end .container -->
</div><!-- end #header -->
```

Either way, they would both work.

Sometimes a "*cl*" is better, sometimes it's "*cf*". It depends on what you need to get done for the task.

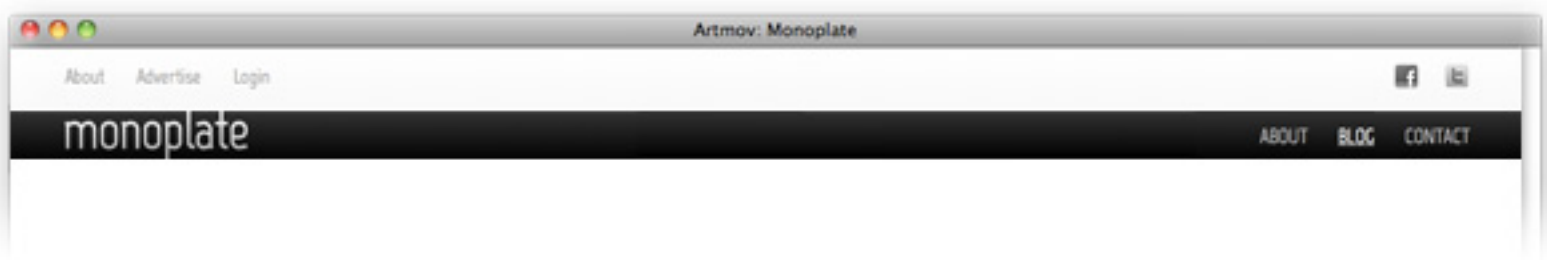
Move on to the menu styling needed for the *#header*.

Here's how the CSS should look like:

```
#menu {
      line-height:0; /* remove unwanted space caused by a children with clearfix */
}
#menu ul {
      padding:13px 0; /* create space between the top and bottom edges of links */
}
#menu ul li {
      font:normal 20px/25px "Marvel", Arial, sans-serif; /* set the custom font */
      text-transform:uppercase; /* uppercase all text from CSS */
      margin:0 30px 0 0; /* create space between the right and left edges of links */
}
#menu ul li a {
      text-decoration:none; /* disable the underline style */
      color:#b9b9b9; /* change the color of each anchor */
}
#menu ul li.active a,
#menu ul li a:hover {
      text-decoration:underline; /* enable the underline style */
      color:#FFF; /* change the color differently */
}
```

Save your *index.html* and *style.css* files and hit refresh in your web browser.

The results would be:



7 Code the *#page*

The *#page* gradient would be coded in the same way with CSS gradients, just as *#header*.

What's different about the *#page* gradient is that it's created using three transitioning colors, rather than two, which was the case of the *#header*.

The *#page* gradient transitions from blue to light green and then to white.

That's not an issue for you. Like the gradients in Photoshop Blending Effects panel, you can choose many colors for your CSS3 gradient.

So, the CSS for this looks like:

```
#page {
      background:rgb(92,168,186);
      background:-moz-linear-gradient(top, rgb(92,168,186) 0%, rgb(223,246,234) 55%,
                  rgb(255,255,255) 100%);
      background:-webkit-gradient(linear, left top, left bottom, color-stop(0%,
                  rgb(92,168,186)), color-stop(55%, rgb(223,246,234)), color-
                   top(100%,rgb(255,255,255)));
      background:-webkit-linear-gradient(top, rgb(92,168,186) 0%, rgb(223,246,234) 55%,
                   rgb(255,255,255) 100%);
      background:-o-linear-gradient(top, rgb(92,168,186) 0%, rgb(223,246,234) 55%,
                  rgb(255,255,255) 100%);
      background:-ms-linear-gradient(top, rgb(92,168,186) 0%, rgb(223,246,234) 55%,
                  rgb(255,255,255) 100%);
}
```

Use the little awesome tool from *ColorZilla* tool and paste the output generated inside the *style.css* file.

It's easy and fast.

Notice from the design of *Monoplate* that you have a 1px lighter line between the *#header* and *#page* ?

You can create that using the *border* property.

Append it to either *#header* as a *border-bottom* or to *#page* as a *border-top*.

Beside that line there is also a space between the *#header* and the *#page* actual content. Make use of the *padding* property to create that space.

```
#page {
      background:rgb(92,168,186);
      background:-moz-linear-gradient(top, rgb(92,168,186) 0%, rgb(223,246,234) 55%,
                  rgb(255,255,255) 100%);
      background:-webkit-gradient(linear, left top, left bottom, color-stop(0%,
                  rgb(92,168,186)), color-stop(55%, rgb(223,246,234)), color-
                   top(100%,rgb(255,255,255)));
      background:-webkit-linear-gradient(top, rgb(92,168,186) 0%, rgb(223,246,234) 55%,
                  rgb(255,255,255) 100%);
      background:-o-linear-gradient(top, rgb(92,168,186) 0%, rgb(223,246,234) 55%,
                 rgb(255,255,255) 100%);
      background:-ms-linear-gradient(top, rgb(92,168,186) 0%, rgb(223,246,234) 55%,
                 rgb(255,255,255) 100%);
      border-top:1px solid #74b7c3;
      padding:49px 0 0 0;
}
```

The above was the code needed for the *#page* CSS; now add the HTML for it.

Save your *style.css* file and add the basic HTML structure for *#page*, just as you added it for the *#header* container.

```
<div id="page">
     <div class="container">

     </div><!-- end .container -->
</div><!-- end #page -->
```

Monoplate shows a 3 pixels border line above the content of the *#page* container.

There is a *div.container* inside the *#page*, responsible for centering the content, so you can include the border line with that selector: *#page div.container*.

The CSS - simple and straight forward:

```
#page div.container {
      border-top:3px solid #367d8d;
}
```

The *#page* has the gradient set, but the content in the *#page* sits on a white background, as it is shown in *Monoplate.psd*.

Declare that background. It can be applied to the content inside *#page*, so the container that centers it: the *#page div.container* selector.

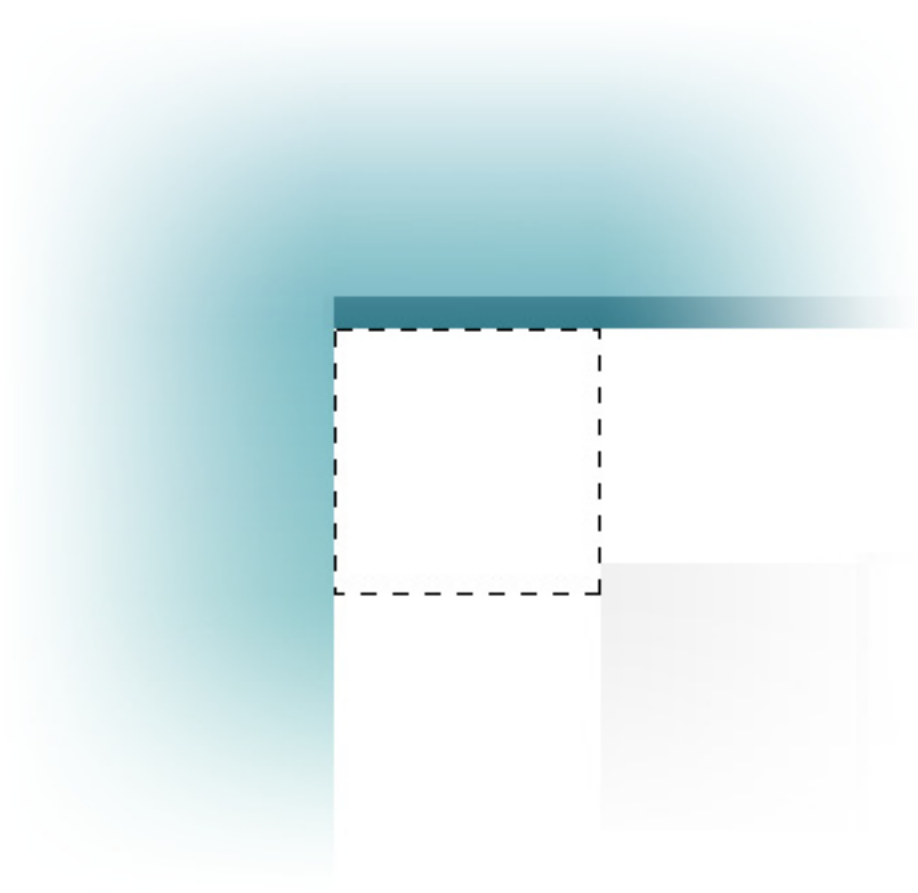
The new updated CSS for this is:

```
#page div.container {
      border-top:3px solid #367d8d;
      background:#FFF;
}
```

It's good so far.

View the design back in Photoshop and start with the *#page*'s content: from the the first block, which was named as *#hello*.

The content within *#page* is positioned a bit far from the edges of its *div.container*, so add a new *div* and a *padding* property to move and position the content.



Use the Photoshop's *Marquee Tool* to get the width x height ratio.

That's 22x25px measured, which you can add in the new container called *content*.

```
#page div.content {
      padding:22px 25px;
}
```

Save *style.css* and update the HTML code:

```
<div id="page">
      <div class="container">
            <div class="content">

            </div><!-- end .content -->
      </div><!-- end .container -->
</div><!-- end #page -->
```

8 Code the *#hello*

Switch back to *Monoplate* and have a look at the *#hello* container just a little bit before moving to writing any code.

What's in for the *#hello* container:

- there's a bar that is going to be repeated in the *#blog* container too;
- text and buttons are arranged in two columns.

Because the "*hello*" bar is going to be included into another container (e.g. *blog*), code it outside *#hello*.

Name it simply *p.category:*

```
#page p.category {
      background:#f1f1f1; /* set the background color */
      height:18px; /* set the height */
      width:100%; /* make it a 100% width */
}
```

Above, you coded the *p.category* as a simple *p* tag with a 100% width and a specific height of 23px and filled its background.

Update the HTML for the *#hello* container and add the "*Hello*" word itself as you can see below.

The specific word, "*Hello*", is added with a white background, but since it is fluid (e.g. *Hello vs. Blog*), reproduce it from CSS.

This allows it to change easily.

```
<div id="hello">
     <p class="category"><a href="#" title="#">Hello</a></p>
</div><!-- end #hello -->
```

You can include it either by using an *a* tag, but it would become a link or switch over to a *span* tag.

Both of these tags are inline elements by default.

Monoplate uses "a".

Analyze the box surrounding the *hello* word and you can simply see that everything can be created using some CSS properties, such as:

- border;
- background with gradient;
- drop shadow effect for the text.

Use the same online tool for the gradient from *ColorZilla*: *www.colorzilla.com/gradient-editor/*

Besides the background color, you also need to:

- add a drop shadow effect with the CSS3's *text-shadow* feature;
- move the box more to the right;
- add border around the text;
- change the default color;
- set the font size.

The CSS for this part is going to look something like this one below.

It has everything it needs to be rendered correctly.

```
#page p.category a {
      background:rgb(255,255,255);
      background:-moz-linear-gradient(top, rgb(255,255,255) 0%, rgb(244,244,244) 100%);
      background:-webkit-gradient(linear, left top, left bottom, color-stop(0%,
                  rgb(255,255,255)), color-stop(100%, rgb(244,244,244)));
      background:-webkit-linear-gradient(top, rgb(255,255,255) 0%, rgb(244,244,244) 100%);
      background:-o-linear-gradient(top, rgb(255,255,255) 0%, rgb(244,244,244) 100%);
      background:-ms-linear-gradient(top, rgb(255,255,255) 0%, rgb(244,244,244) 100%);
      background:linear-gradient(top, rgb(255,255,255) 0%, rgb(244,244,244) 100%);
      padding:6px 15px 5px 15px; /* create the space around the text */
      border:1px solid #e7e7e7; /* add the border */
      font:normal 11px/16px Arial, sans-serif; /* set the font size */
      text-shadow:0 2px 0 #FFF; /* add the drop shadow effect on the text */
      text-decoration:none; /* disable the underline */
      margin:0 0 0 17px; /* move the text to the right */
      color:#999; /* change the default color */
}
```

Maybe add a *:hover* state the *a* tag, too:

```
#page p.category a:hover {
      color:#000;
}
```

Back to your *index.html*, save it (*as often as possible*) and refresh your web page to see the results.

The main part of the *#hello* container is made out of an article on the left and a slider on the right.

The *index.html* converted file of *Monoplate* wraps these two components into a *div.article*, as such:

```
<div class="article c2 cf">
      <div class="excerpt c21">

      </div><!-- end .c21 -->
      <div class="slider c22">

      </div><!-- end .slider -->
</div><!-- end .article -->
```

Using the "*article*" wrapper, add a *div.excerpt* on the left side to include the article content (e.g. *title, text etc.*) and a *div.slider* on the right side.

Lets see what the *c2*, *c21* and *c22* are and how these classes from *F2fw* can help you.

```
div.c2 {
     display:block;
}
div.c2 div.c21 {
     float:left;
     width:49%;
}
div.c2 div.c22 {
     float:right;
     width:49%;
}
```

c2 (columns: 2)
Any "*c21*" and "*c22*" part of the "*c2*" selector are being floated on the left and, respectively, on the right with a 49% width set for both.

It's similar to having two *div*, one with *float:left;* and the other one with *float:right*.

Yet again, the "*cf*" class is there to automatically clear the floats after the *c2* container.

Another option you could have used is to float the *div.excerpt* on the left with a *float:left;* and the *div.slider* on the right with the *float:right;* property.

F2fw already gives you the two columns grid with the *float* properties included.

Move ahead on the styling of the *div.article*.

The *Monoplate* design shows a line with space around it at the bottom of the *#hello* container.

The CSS for the border and space:

```
#hello {
      border-bottom:1px dotted #d2d2d2; /* set the dotted bottom line */
      padding:0 0 25px 0; /* add the bottom padding */
      margin:0 0 25px 0; /* add the bottom margin */
}
```

Save your *style.css* and move further.

Start with the article.

The title and the paragraphs are simple and straight to the point to implement in your CSS:

```
#hello div.article h2 {
      font:normal 25px/33px “Marvel”, Arial, sans-serif; /* set the custom font */
}
#hello div.article p {
      font:normal 14px/23px Arial, sans-serif; /* set the font and size */
      color:#777; /* change the default color */
}
```

Update your *index.html* with some dummy content for the *h2* tag and *p* tags.

The next paragraph after the article is made out of the three buttons, the custom ones.

You already analyzed the buttons in *Chapter 1* and cropped their background.

It’s time to implement them.

As always, be sure to save your *index.html* and *style.css* files regularly, after each block of code you write.

9 Code the custom buttons

The start point on these custom buttons is the HTML structure inside the *div.article* container.

Add the anchors and define a special class:

```
<p class="buttons">
     <a href="#" title="#" class="btn">Learn from it</a>
     <a href="#" title="#" class="btn">Read more</a>
     <a href="#" title="#" class="btn">Grab the file</a>
</p>
```

Everything is added within a new paragraph with the class of "*buttons*", while each button has their own general class, called "*btn*", the same name you used for the sliced image.

Before moving on to the buttons conversion, go and disable the bottom margin of the paragraph and create the space between buttons.

```
#hello p.buttons {
     margin:0; /* disable the default bottom margin */
}
#hello p.buttons a {
     margin:0 20px 0 0; /* create the space between the buttons */
}
```

The next paragraphs in these pages are mainly about these 3 custom buttons.

To keep things simple, the buttons receive the class as "*btn*", while the image cropped from the design is also called "*btn*" (*btn.png*).

Make sure you don't add "*btt*" but "*btn*", since the first is used by *F2fw* and it's similiar to "*ico*" (see the header social icons).

Lets recap your early work on the analysis and what can be created using CSS magic:

- font styling and text shadow with *text-shadow;*
- the button's shadow with *box-shadow;*
- the button's border with *border.*

The CSS:

```
a.btn {
      background:url(../images/btn.png) repeat-x top left;
      -webkit-box-shadow:0 0 3px #CCC;
      -moz-box-shadow:0 0 3px #CCC;
      box-shadow:0 0 3px #CCC;
      border:1px solid #90c8d4;
      padding:4px 10px;
      font:normal bold 14px/21px Arial, sans-serif;
      text-shadow:0 1px 0 #FFF;
      text-decoration:none;
      color:#5e9ba8;
}
```

The *box-shadow* property adds the shadow of the button; the *-webkit- box-shadow* and *-moz-box-shadow* properties targets early versions of Webkit-based browsers and Firefox browser.

The border of the button is easily created with a *border:1px solid #90c8d4;* property and the rest of the font styling is very simple, as well (*font, text-decoration, color etc.*).

Remember that both states of "*btn*' are in the same *btn.png* file?

First part is the *default* state and the secondary part is for the *:hover* state to show whenever you move the cursor on it.

While you could have moved them in two separate files, two issues would have arrised:

- you need to download two separate images, thus adding an other resource to download;
- a flash of white background would appear when *:hover* is activated because only when you activate the *:hover* state, you actually download that image.

Adding them in the same *btn.png* file proves to be the most effective solution considering that the amount of text won't change in height, but in width.

It also improves the size of the website by reducing the images to the minimum size needed (*remember that we've trimmed at a max width of 2px*) and grouped into one image, *btn.png*.

After displaying the button, just add the *:hover* state in the *style.css* by changing the position of *btn.png* background: from *top left* to *bottom left*.

```
a.btn:hover {
      background-position:bottom left;
      color:#3b636b;
}
```

This is going to swap the *btn.png* background and display the last part of background when the *:hover* state is active.

That's about it.

The custom buttons are now succesfully converted for the *#hello* container.

10 Code the custom buttons for the slider.

The thumbnail of the slider is handy to implement, so create a quick HTML code for it before moving on to the slider buttons:

```
<div class="slider c22">
     <p class="thumbnail">
          <img src="includes/pictures/sample_462_225.png" alt="#" />
     </p>
</div><!-- end .slider -->
```

The thumbnail is also in the */pictures* folder and named as a "*sample_462_225.png*" image.

In your *style.css*, just remove the default margin that is being applied to the *p* tag.

The code would be something like this:

```
#hello div.slider p {
      margin:0;
}
#hello div.slider p.thumbnail img {
      display:block;
}
```

It's time to focus on the slider navigation now.

Since the buttons are displayed on top of the thumbnail itself, lets have the container with a *position:relative;*

Then create a new paragraph to include buttons, but with *position:absolute;* property added so it can stay on top the thumbnail.

```
#hello div.slider {
      position:relative;
}
```

The buttons will go into a new paragraph, with the class as *navigation*.

```
<div class="slider c22">
     <p class="thumbnail">
          <img src="includes/pictures/sample_462_225.png" alt="#" />
     </p>
     <p class="navigation">
          <a href="#" title="#" class="btn">Prev</a>
          <a href="#" title="#" class="btn">Next</a>
     </p>
</div><!-- end .slider -->
```

You may have noticed the "*btn*" class for these buttons. Here is why you should use it:

Whenever *:hover* state for these buttons is active, the background shown is actually the first part of the *btn.png* image or the *default* state of the buttons done for the *#hello* container.

To save time on coding, use the same class and just overwrite the background position for the slider buttons only.

This means that with "*btn*" applied here the slider buttons also takes its properties, such as box shadow, border etc. All of these can be overwritten.

The complete CSS code would be as:

```
#hello div.slider p.navigation a.btn {
      border-color:#b9b9b9;
      background-color:#FFF;
      background-position:-100px -100px;
      padding:3px 6px;
      display:block;
      float:left;
      margin:0 5px 0 0;
}
```

Here's what you overwritten from the *btn* class to apply to the slider buttons:

- the border: since it uses the same width for the border, you've just replaced the color with the *border-color* property;
- the padding: the space is a bit different;

• the background: the *default* state of the slider buttons uses a plain white background.

You added *background-color:#FFF* to overwrite the *btn* default background, but also added the property *background-position:-100px -100px;* which isn't going to show anything because the *btn.png* doesn't extend up to a 100px width or height.

It's an intentional action to remove the image and just leave only the *background-color:#FFF;* property.

The other properties in the above code (e.g. *float and margin*) are added to position the buttons.

Before including the arrows, position the *p.navigation* selector with the help of *position:absolute* and some *right* and *top* values (e.g. *right:15px; and top:20px;*).

The buttons now have the style applied, but they don't have the arrows found in the design.

For this you could use the *ico* class of *F2fw*, just as you did with the social icons.

Use the same file, *ico.png*, to load the entire list of icons: social icons (*Facebook and Twitter*) and these new arrows for the slider.

Open *ico.png* again and move the arrows there. It should look like this afterwards:

This is the new *ico.png* with the arrows added.

Make sure that the arrows are on a transparent background. When the *:hover* is active, the user needs to see the stripped lines behind the arrow, not the arrow surronded by white background.

Including them into HTML is easy.

Do the same as you did for the social icons: define your classes and insert them with the general "*ico*" class along the individual classes.

Wrap your words (*prev and next*) inside a new *span* tag with the classes defined as "*ico-arr-left*" and as the "*ico-arr-right*" and add the "*ico*" class.

The HTML:

```
<p class="navigation">
     <a href="#" title="#" class="btn"><span class="ico ico-arr-left">Prev</span></a>
     <a href="#" title="#" class="btn"><span class="ico ico-arr-right">Next</span></a>
</p>
```

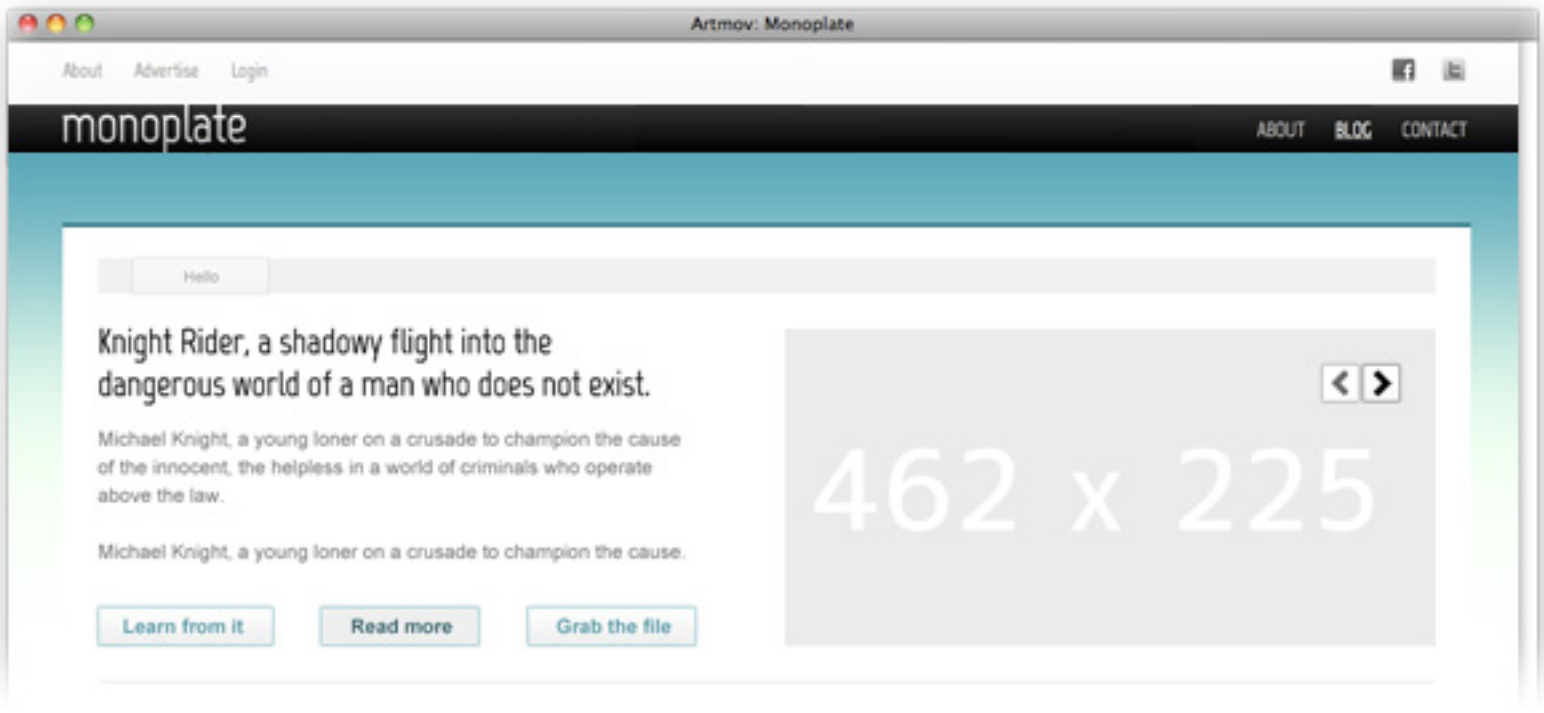
In *style.css* define the new classes of "*ico-arr-left*" and "*ico-arr-right*" in the same way as you did you with the header social icons.

The CSS code for this is going to look very familiar and easy:

```
span.ico-arr-left,
span.ico-arr-right {
      height:19px;
      width:12px;
}
span.ico-arr-left {
      background-position:0 -16px;
}
span.ico-arr-right {
      background-position:-12px -16px;
}
```

As with the social icons, they both have the same width and height, so you've grouped them together and then defined them individually for the background position property.

Save your *index.html*, *style.css* and refresh the web browser to check the final results.



Congrats.

This is how your *#hello* container should look in your browser. Everything is aligned and looks the same as was designed in *Monoplate*.

What is the next container to code?

As you move to the next battlefield, below the *#hello* container there is *#albums,* as it's being called in *Monoplate.*

11 Code the *#albums*

The container has 4 thumbnails floated one after another. An unordered list could work here (e.g. *ul li*).

The HTML is simple:

```
<div id="albums">
     <ul class="nostyle cf">
          <li>

          </li>
     </ul>
</div><!-- end #albums -->
```

Before adding the content inside the *li*, create the styles for each of these *li* in your *style.css* file.

The *#albums* container has a bottom border, just as *#hello* did. Style the *#albums* in the same way.

```
#albums {
      border-bottom:1px dotted #d2d2d2;
      margin:0 0 25px 0;
}
```

Back to the unordered list from *index.html*, the CSS would be fairly easy to do.

Because the *ul* already gets a "*nostyle*" class from *F2fw* there is no need to add a *list-style-type* property to the *li*. More on this in the next page.

```
#albums ul {
      padding:0;
}
#albums ul li {
      margin:0 24px 0 0;
      display:block;
      width:219px;
      float:left;
}
#albums ul li p {
      margin:0;
}
```

You can use the *Marque Tool* to measure the distance between each thumbnail (*24px in this case*).

Have a look at the "*nostyle*" class that is been added in the converted *index.html* and what *F2fw* brings to this class.

```
ul.nostyle {
     padding:0;
     margin:0;
}
ul.nostyle li {
     list-style-type:none;
}
```

nostyle (no style)
Disables *padding* and *margin* of *ul*, as well any bullets for inner *li*.

This class comes in handy whenever you want to disable the margin or the padding of any given *ul* (or *ol*) and its inner *li list-style-type* property.

What you did so far is to convert the thumbnails from the *#albums* section using the *li* tags, which are being floated on the left with *display:block;* and *float:left;* properties.

The distance between each *li* is set at 24 pixels, as measured from *Monoplate*.

The *margin:0;* is added for the *p* tag to remove the margin. The content of an item inside *#albums* is going to be positioned with paragraphs.

The HTML for adding the thumbnail and its below text into sits in paragraphs, as such:

```
<li>
     <p class="thumbnail">
          <img src="includes/pictures/sample_170_300.png" alt="#" />
     </p>
     <p class="content">
          <a href="#" title="#">see more about this</a>
     </p>
</li>
```

Focus on the white/gray split that each *li* has. That portion is easily measured at 24 pixels.

A work around is to create a *1x24px* image with a full white background and insert it for the *p.thumbnail* element. That's an additional resource to download.

Monoplate focus is on speed and another solution for this is to make use of the *:after/:before* selectors.

You can easily replicate that portion using the *:before* selector, as shown below:

```
#albums li p.thumbnail {
      padding:0 0 20px 0;
      position:relative;
      background:#EEE;
}
#albums li p.thumbnail:before {
      position:absolute;
      background:#FFF;
      height:24px;
      width:100%;
      content:'';
      left:0;
      top:0;
}
```

Before you add the *:before* selector, set the thumbnail paragraph to be relative positioned.

The *p.thumbnail* is set with a full height background (gray) and a bottom padding property of 20 pixels to pull that gray background more to the bottom, to replicate it from *Monoplate*.

Using the *:before* selector you position a 24px tall of white background on top of that background, but because of the *:before* absolute position, the white background gets priority and it would sit on top of any content inside the *li* (e.g. *including the thumbnail*).

To disable that, position the thumbnail itself on top of the white background which is easy with a *position:relative* and a *z-index* value of 1, as such:

```
#albums li p.thumbnail img {
      position:relative;
      z-index:1;
}
```

A few more changes left to do on the thumbnail.

Create the 1px border around the image and show the image as a block element, to make sure there aren't any margins below it. Then center it with a *margin:0 auto;* property.

```
#albums li p.thumbnail img {
      position:relative;
      z-index:1;
      border:1px solid #FFF;
      display:block;
      margin:0 auto;
}
```

The link below the thumbnail of a *li* is dropped into a paragraph with a "*content*" class, so focus to do the styling of that link and later on move to the icon that is on it's left side.

You can use the "*ico*" class for that icon too. It would speed things up very quickly.

Create the stylesheet code for that content styling in your *style.css*.

It should look like this one:

```
#albums li p.content {
      font:normal bold 18px/21px “Marvel”, Arial, sans-serif;
      text-align:center;
      padding:20px 0;
      margin:0;
}
#albums li p.content a {
      color:#367d8d;
}
#albums li p.content a:hover {
      color:#999;
}
```

Aside from the *padding:20px 0;* property, used to do the space around the top and bottom parts of the text, everything seems pretty straight forward:

- font styling;
- color changed for the links;
- disabled the default bottom margin for the paragraph.

That's about it for this content.

Moving on to the link left icon. Here's the HTML code with the "*ico*" class added in the same as you did with the other icons:

```
<p class="content">
     <a href="#" title="#"><span class="ico ico-more"></span>see more about this</a>
</p>
```

Use the same *ico.png* file and drop the "*plus*" icon in there, but before saving that image, get each icon size and background position.

Your new *ico.png* should look like this one:

Switch back to your *style.css* file and include the data for this icon, such as width, height and background position. In the same way as the other icons:

```
span.ico-more {
      background-position:0 -35px;
      height:11px;
      width:11px;
}
```

The icon needs to be aligned with the text and, similar to the header social icons, you can either use the "*display:inline-block;*" property or "*float:left*".

Here's the code for the first one:

```
#albums li p.content a span.ico {
      display:inline-block;
}
```

Everything is almost ready. Some small details that needs to be done and it's done.

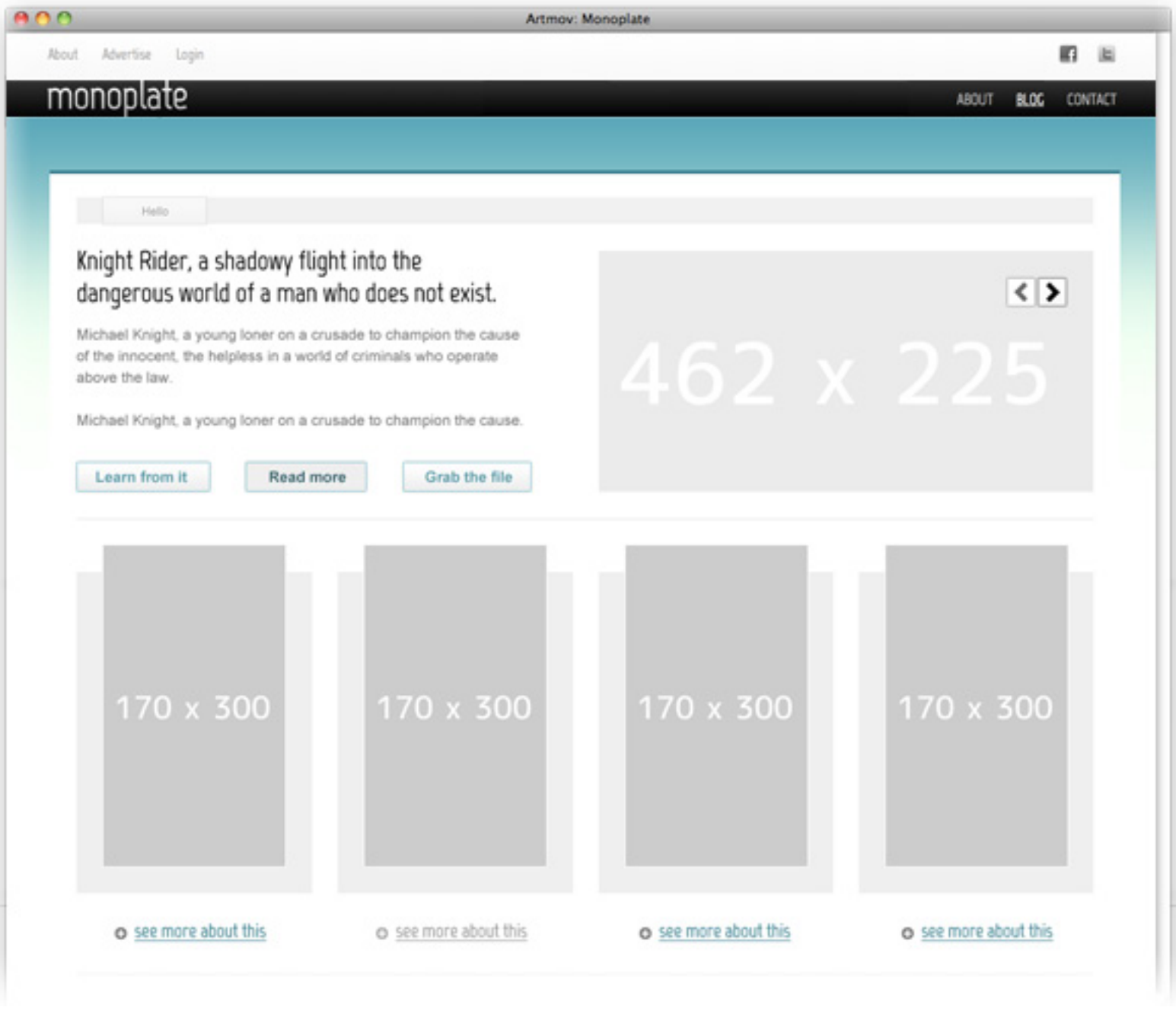
You need to style the icon as shown in the *Monoplate* design. Meaning to reduce opacity on the *default* state, just like the social icons.

```
#albums li p.content a span.ico {
      display:inline-block;
      margin:0 10px 0 0;
      opacity:0.5;
}
#albums li p.content a:hover span.ico {
      opacity:1;
}
```

And here's a sneak preview on its final look:

Done? That's great!

Save your *index.html* and *style.css* and hit refresh on your web browser. Here's how it should look:



That's about the end for the *#albums* container. It was easy and straight to the point.

Moving on to the next one, *#blog*.

12 Code the *#blog*

The *#blog* container also has the *p.category* used from the *#hello* container.

That's a good point to start with:

```
<div id="blog">
     <p class="category"><a href="#" title="#">Blog</a></p>
</div><!-- end #blog -->
```

Lets go to the blog article now.

It has a space between its left and right edges, so wrap that up inside a new *div* named *"article"*.

The HTML code for this part would be:

```
<div id="blog">
     <p class="category"><a href="#" title="#">Blog</a></p>
     <div class="article">

     </div><!-- end .article -->
</div><!-- end #blog -->
```

Before stepping forward to adding the icons, add the content of this article inside a *h2* and a *p* tag. Leave the post icons for later.

Add some random dummy content that you have at hand inside the tags.

The *p* tags also get some classes, such as:

- *post-data;*
- *post-content;*
- *buttons*.

The HTML code is shown in the next page.

Here's the code for the HTML without the actually dummy text inside with the CSS classes mentioned in the previous page:

```
<div id="blog">
     <p class="category"><a href="#" title="#">Blog</a></p>
     <div class="article">
          <h2>[..]</h2>
          <p class="post-data">[..]</p>
          <p class="post-content">[..]</p>
          <p class="buttons">[..]</p>
     </div><!-- end .article -->
</div><!-- end #blog -->
```

Halfway there.

Measure the space that surrounds the article. You can use the *Marquee Tool* again and measure the left space and then the right space.

Doing so would provide about 200px on the right side and 17px on the left. With these values, include the new *padding* in the *div.article* selector.

Here's the *style.css* code for the *div.article* with the *padding* property added to have the space:

```
#blog div.article {
      padding:0 200px 0 17px;
}
```

Your next move would be to style the content and the move on to the icons.

This is going to change:

- colors used for links inside the *"post-data"* paragraphs by overwriting default color;
- font sizes and families;
- then add the icons.

You're going to slice these post icons by appeding them to the original *ico.png* file, in the same way as you did with the slider arrows and header icons.

This is how your new *ico.png* image should look after you include the post icons.

Just as before, get their sizes and positions before saving the file.

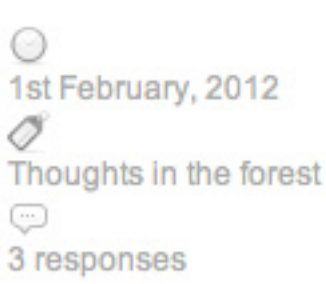
The CSS for these new icons:

```
span.ico-date {
      background-position:-11px -35px;
      height:16px;
      width:16px;
}
span.ico-tags {
      background-position:-27px -35px;
      height:16px;
      width:16px;
}
span.ico-comments {
      background-position:-43px -35px;
      height:16px;
      width:16px;
}
```

After adding the icons into the HTML code, save and check your results. The HTML code:

```
<p class="post-data">
     <a href="#" title="#"><span class="ico ico-date"></span>[..]</a>
     <a href="#" title="#"><span class="ico ico-tags"></span>[..]</a>
     <a href="#" title="#"><span class="ico ico-comments"></span>[..]</a>
</p>
```

Refresh your page and you should have something like this displayed in your browser:



If that's what being shown for you, perfect.

The icons are being shown correctly, but you need to align them and add the space between the icon and anchor text.

Use the same approach with the *display:inline-block;* property to align the icons along the text. Maybe add a *position:relative;* to make a more perfect alignment (when that's needed).

Have a look below:

```
#blog div.article p.post-data a span.ico {
      display:inline-block;
      margin:0 5px 0 0;
      position:relative;
      top:2px;
}
```

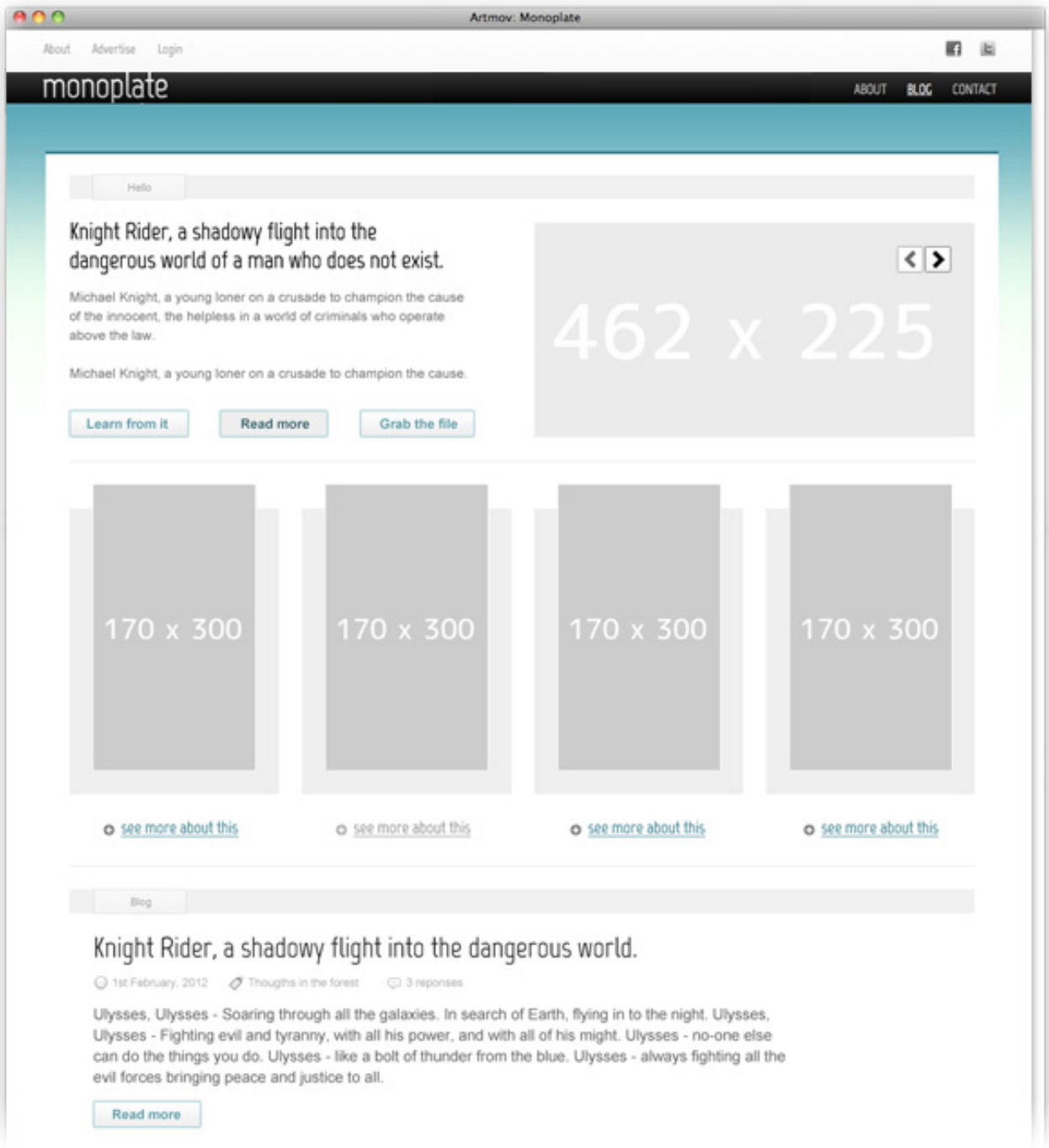
The space between the icon and the text inside the link is done and working, but you also need to apply a space between the links (e.g. *date - tags - responses*).

A *margin:0 20px 0 0;* would do:

```
#blog div.article p.post-data a {
      margin:0 20px 0 0;
      text-decoration:none;
      color:#aaa;
}
```

Save your *index.html* and *style.css*.

Here's how the final result should look:



Almost done.

What is left to do now is to convert the *#footer* and then moving on to the third chapter of this guide where you check the results.

13 Code the *#footer*

The main container here, called *#footer*, can fill the gray background of *Monoplate*, while a new *div.content* adds the necessary space around the content inside the footer.

Also, use *#footer* to add the darkened border line that sits on the top, while you can apply a *padding* property to the *div.content* container to create the space you need.

Next page shows you the basic HTML for this.

Here's the basic HTML code for the *#footer* container. Start from here:

```
<div id="footer">
      <div class="content">

      </div><!-- end .content -->
</div><!-- end #footer -->
```

The CSS code:

```
#footer {
      border-top:3px solid #cdcdcd;
      background:#efefef;
}
#footer div.content {
      padding:25px;
}
```

Done.

Now just add some dummy content inside this new container, style it as shown in *Monoplate* and check the results afterwards.

Use an unordered list for the dummy content.

Here's a sample of a *ul* found in the converted file *index.html* of *Monoplate*; you can replicate it further more for the rest of the content.

```
<ul class="nostyle">
     <li><h6>About us</h6></li>
     <li><p><a href="#" title="#">Where are we located?</a></p></li>
     <li><p><a href="#" title="#">Want to join us?</a></p></li>
     <li><p><a href="#" title="#">How we work</a></p></li>
</ul>
```

The "*nostyle*" class should already be familiar for you since it was used earlier for the *#albums* container.

Float these *ul* on the left and, as *Monoplate* shows you, add some space in between them.

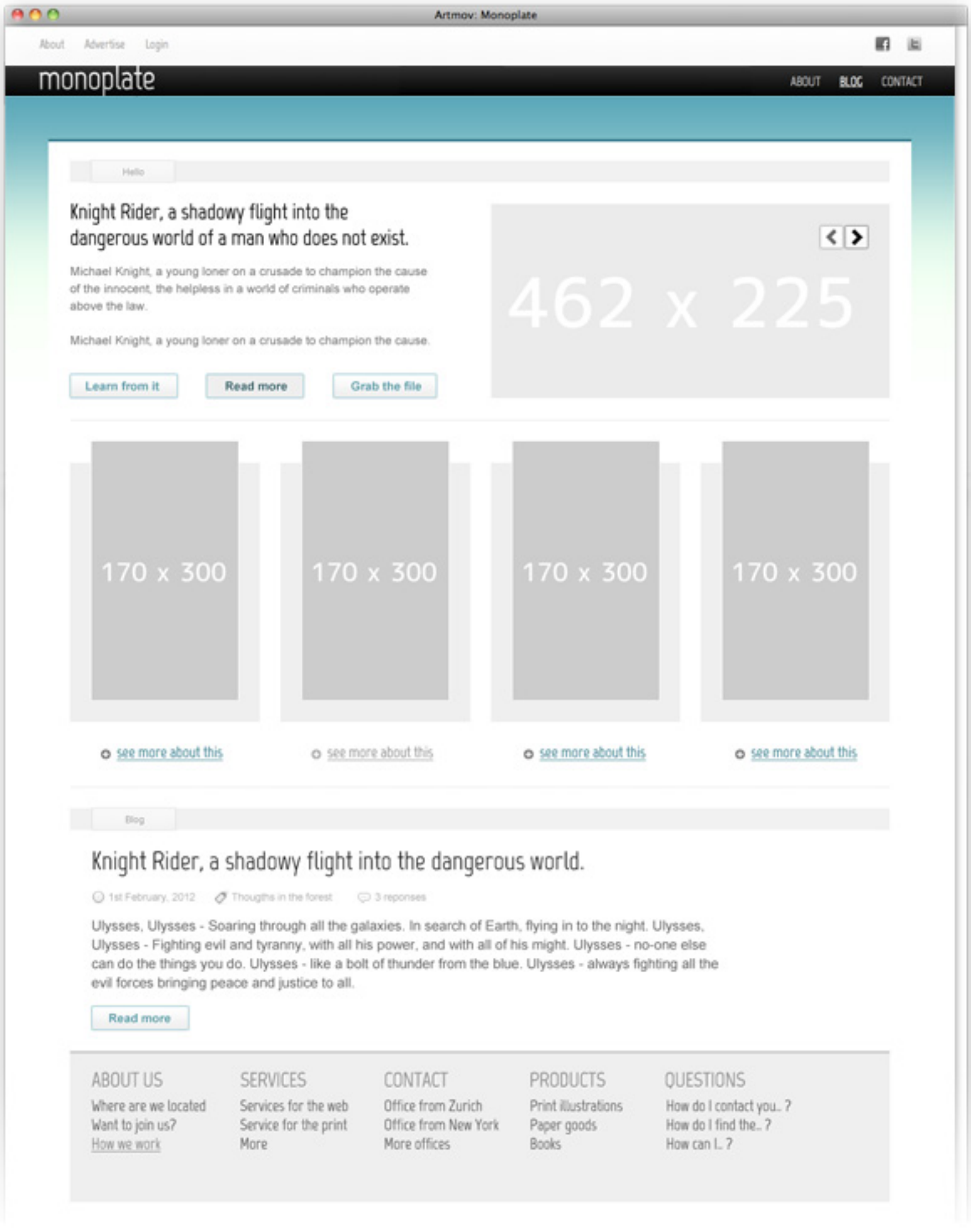
```
#footer ul {
     margin:0 50px 0 0;
     float:left;
}
#footer ul li {
     margin:0;
}
```

Here's what tags needs to be styled for the content:

- the *h6* tags: the title of a column;
- the *p* tags: the paragraph with links;
- the *a* tags: the links;
- the *:hover* states of the links.

```
#footer ul li h6 {
      font:normal 24px/29px “Marvel”, Arial, sans-serif;
      text-shadow:0 1px 0 #FFF;
      text-transform:uppercase;
      margin:0 0 5px 0;
      color:#666;
}
#footer ul li p {
      font:normal 18px/24px “Marvel”, Arial, sans-serif;
      color:#656565;
      margin:0;
}
#footer ul li p a {
      text-decoration:none;
      color:#656565;
}
#footer ul li p a:hover {
      text-decoration:underline;
      color:#898989;
}
```

Save your *index.html* and *style.css*.


Artmov: Monoplate
About
Advertise
Login
monoplate
ABOUT
BLOG
CONTACT
Hello
Knight Rider, a shadowy flight into the dangerous world of a man who does not exist.
Michael Knight, a young loner on a crusade to champion the cause of the innocent, the helpless in a world of criminals who operate above the law.
Michael Knight, a young loner on a crusade to champion the cause.
Learn from it
Read more
Grab the file
462 x 225
170 x 300
170 x 300
170 x 300
170 x 300
see more about this
see more about this
see more about this
see more about this
Blog
Knight Rider, a shadowy flight into the dangerous world.
1st February, 2012
Thougths in the forest
3 reponses
Ulysses, Ulysses - Soaring through all the galaxies. In search of Earth, flying in to the night. Ulysses, Ulysses - Fighting evil and tyranny, with all his power, and with all of his might. Ulysses - no-one else can do the things you do. Ulysses - like a bolt of thunder from the blue. Ulysses - always fighting all the evil forces bringing peace and justice to all.
Read more
ABOUT US
Where are we located
Want to join us?
How we work
SERVICES
Services for the web
Service for the print
More
CONTACT
Office from Zurich
Office from New York
More offices
PRODUCTS
Print illustrations
Paper goods
Books
QUESTIONS
How do I contact you.. ?
How do I find the.. ?
How can I.. ?

Congratulations!

You officialy converted a Photoshop design into functional HTML/CSS files.

Lets recap what you've done so far:

- you have analyzed Monoplate before writing any HTML or CSS code to grasp the big picture of what it is about;

- you did an overview of *Monoplate* design and decided which parts can be reproduced using CSS properties and which can not;

- you have cropped a few images along with the analysis done, just enough to get the process starting on the right track;

- you wrote many HTML and CSS lines of code that fully reproduced *Monoplate*;

- you used the *F2fw* framework to speed things up quickly and write less code;

- you switched back-and-forth between Photoshop, your text editor and your web browser to often check the results and make neccessary changes, when needed.

It's time to check your converted files and to move on to the last step of this guide: check.

Check

Verify your results for maximum compatibility

In this last step, called "*check*", you will verify your converted *index.html* that it matches the .psd design and check for bugs in other browsers.

And, if you find any, you start to fix them.

Here's the list of the web browsers you're going to check the converted design and see if it's okay:

- Chrome 19 (19.0 - used when coding);
- Internet Explorer 7 and 8;
- Opera 11 (11.64);
- Firefox 12 (12.0);
- Safari 5 (5.1.7).

In Internet Explorer 7 and 8, it's important to have an accessible design rather than a pixel-perfect one.

Having a pixel-perfect design in all browsers would be nice, but the cost of supporting *Internet Explorer 7 or 8* is simply higher and requires more time.

If you haven't done the coding in *Google Chrome*, open it and see if it matches with *Monoplate*.

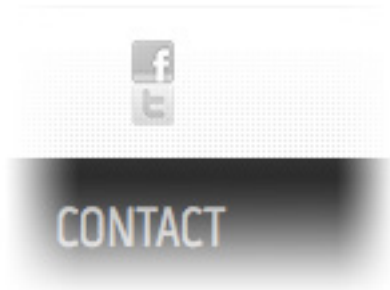
If it doesn't, list below which parts don't match:

Firefox 12 (12.0)

List below which parts don't match:

the header social icons aren't aligned.

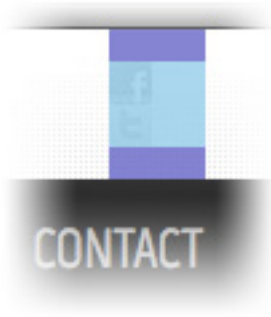
So in Firefox 12 the converted *Monoplate* has a bug: the social icons, from the *#header*, aren't aligned.



The fastest way to debug this bug is to get *Firebug* for Firefox and inspect that element to see why it's not being aligned properly.

Firebug for Firefox can be found here:
http://getfirebug.com/

Once installed, go and inspect the paragraph.
(e.g. *right click on an icon > inspect element with Firebug*)



Now notice that the Facebook icon isn't being pushed to the left although it has the *fl* class.

Assuming that if both icons would have the *float* property included then they're are going to be aligned properly.

Testing this theory is simple.

Go to the Twitter icon and add a *float:right;* property right from *Firebug* to test it easily.

```
<a class="ico ico-facebook
fl" title="#" href="#">Facebook</a>
<a class="ico ico-twitter
nmr" title="#" href="#" style="float: right;">Twitter</a>
```

The above shows that Twitter is being selected in *Firebug* and here below is the the *float:right;* being added live in the right side of the *Firebug* panel:

```
element.style {
    float: right;
}
```

The theory is correct. You can find and test more solutions for this bug, if you want to.

Whatever you do to fix a bug in *Firefox 12*, you need to check again that it displays correctly in the other browsers you support (e.g. *Chrome* vs. *Firefox*), unless you use conditional comments.

Other options:

- declare a width for the right paragraph to limit it at a specified size, so the Facebook icon doesn't go on the right side, but on the left;
- what other solutions do you think can work here?

Switch to your *index.html* file and include a "*fr*" class for the Twitter icon.

```
<p class="floatright">
     <a href="#" title="#" class="ico ico-facebook fl">Facebook</a>
     <a href="#" title="#" class="ico ico-twitter nmr fr">Twitter</a>
</p>
```

Save it and go back to *Google Chrome* and see if it looks aligned. If *Chrome* and *Firefox* display the icons properly, move on.

Opera 11 (11.64)

List below which parts don't match in Opera:

Safari 5 (5.1.7)

List below which parts don't match in Safari:

Internet Explorer 8
(bugs also applies to IE7)

List below which parts don't match in IE8:

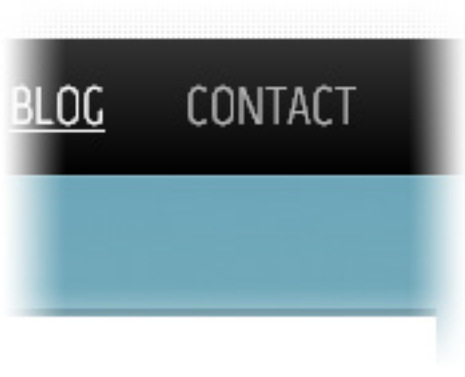
The "contact" link is positioned far from the right edge;

The #header, #page and p.category background gradients aren't showing;

the icons opacity isn't working.

Bug 1:

The *contact* link is positioned far from the right edge.



F2fw automatically disables the last item of the *#menu ul* container with a *:last-of-type* declaration.

IE8 (and 7) doesn't support that selector and so the "*contact*" link has the right margin of the other links.

You can either search for a JavaScript solution that would make IE8 use that declaration or simply edit *index.html* and include a "*nmr*" class to the last item from the menu, the "*Contact*" link.

It's similiar to the Twitter icon, where you added that same class: *nmr*.

Selectivzr is a jQuery library that can be added to have Internet Explorer 8 suport the *:last-of-type*: *http://selectivizr.com/*

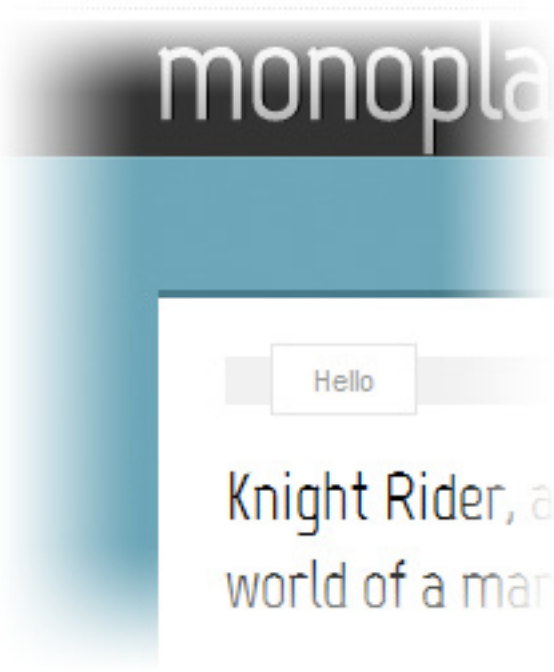
If you choose the HTML option, switch back to your text editor and open *index.html* to include the *nmr* class - apply it the last item.

Your updated HTML should look like this:

```
<div id="menu">
     <ul class="cf">
          <li><a href="#" title="#">About</a></li>
          <li class="active"><a href="#" title="#">Blog</a></li>
          <li class="nmr"><a href="#" title="#">Contact</a></li>
     </ul>
</div><!-- end #menu -->
```

Bug 2:

The *#header*, *#page* and *p.category* background gradients aren't showing.



Like with the *:last-of-type* selector, Internet Explorer 8 (and 7) doesn't support gradients properly.

The *ColorZilla* code included a *filter* property for the *#header* gradient background (same as for the *#page* and *p.category*), but it's removed because it's buggy and highly unrecommended.

Because of this, opt for a solid color background fallback whenever dealing with gradients for IE7-8.

The IE hack to make the gradient work would be:

```
#header {
      filter:progid:DXImageTransform.Microsoft.gradient(startColorstr='#333333',
      endColorstr='#000000', GradientType=0);
}
#page {
      filter:progid:DXImageTransform.Microsoft.gradient(startColorstr='#5ca8ba',
      endColorstr='#ffffff', GradientType=0);
}
#page p.category a {
      filter:progid:DXImageTransform.Microsoft.gradient(startColorstr='#ffffff',
      endColorstr='#f4f4f4', GradientType=0);
}
```

Bug 3:
The icons opacity isn't working.

IE doesn't have full alpha channel transparency; if you include the IE hack to have the opacity, a side effect would be the black corners:

The *filter* property to make the opacity work in IE is below, but remember that it's going to show the black corners.

```
#top p a.ico {
      filter:alpha(opacity=40);
}
#top p a.ico:hover {
      filter:alpha(opacity=100);
}
```

So, don't apply any IE specific hacks.

Go with a 100% opacity for IE8-7, regardless of the icon states, either *default* or *:hover.*

Whenever you are coding for IE8 or 7, go for the fallbacks, such as: solid colors whenever you have a gradient background, no box shadow effect (e.g. *maybe replace with a border property*) and so on.

The work-arounds to make it pixel-perfect requires a lot of time and can be performance drainers.

Internet Explorer 7
(besides IE8 bugs)

List below which parts don't match:

the social icons are misplaced;

the header bar adds a bottom space;

the p.category is being cut;

the custom buttons are being cut also;

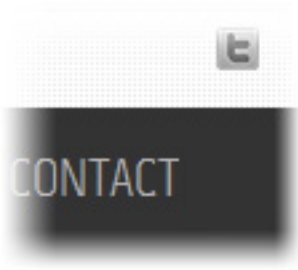
the gray/white split isn't displayed;

the footer doesn't have the bottom space;

the #blog icons aren't being displayed.

Bug 1:
The social icons are misplaced.

IE7 has a lot of bugs, doesn't it? Here is how the icons are being misplaced in Internet Explorer 7:



The right paragraph doesn't have any width set, so IE7 extends it from the right edge of the paragraph on the left to the right edge of the *#top div.container*.

The Facebook icon is floated on the left, so it goes on the left edge of the paragraph.

Open up *style-ie7.css* and include a width limit:

```
#top p.fr {
      width:52px;
}
```

This limits the right *p* to a 52px width - the size of both icons plus the marging.

It can added in *style.css* as an alternative solution to *Firefox*'s bug discovered early on.

Bug 2:
The header bar adds a bottom space.

First, make sure you have the *Developer Tools* for IE8

Hit F12 to open it. It's similar to *Firebug* for *Firefox* or *Developer Tools* in *Chrome*, but it's not that powerful.

Inspect the header bar and go through the entire list of elements that it contains and see from where is the gap being added. The path would be:

#header - div.container - #logo - h3.incl - a

When you've reached the *#logo h3.incl a*, you may have noticed that the gap isn't visible anymore.

Somewhere between the *h3.incl* and *a* element the gap is added.

The easiest fix would be to set a height for *h3.incl*, as you did with the *a* tag.

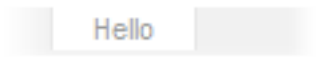
The *style-ie7.css* code for this is easy:

```
#logo h3.incl {
      height:51px;
}
```

Bug 3:

The *p.category* is being cut.

Here's how the *p.category* container looks in the Internet Explorer 7 browser:



A fix is to set a relative position for the *a* of *p.category*:

```
#page p.category a {
      position:relative;
}
```

Many IE7 bugs can be fixed by giving an element a "*layout*". More about that - the concept of *hasLayout*: *http://www.satzansatz.de/cssd/onhavinglayout.html*

Bug 4:

The custom buttons are also cut.

This bug is also fixed by adding a *position:relative;* to the custom buttons, as you did with the *p.category a*:

```
a.btn {
      position:relative;
}
```

Actually, you can group them together:

```
a.btn,
#page p.category a {
      position:relative;
}
```

Bug 5:

The gray/white split isn't being displayed.

Remember that you've used the *:before* selector to

make the gray/white work without attaching a cropped image or an additional div?

IE7 doesn't recognize the *:before* selector. You can recreate it using a mix of negative positioning on the thumbnail or extend the gray background to have the top part to look as the bottom part.

The code is simply a *padding-top* property:

```
#albums li p.thumbnail {
     padding-top:20px;
}
```

The other option you can do is to move the image with a negative *top* value, disable the paragraph *padding-bottom* property and add a *padding-top*

property to the main container to make up on the negative position of the thumbnail.

The CSS code:

```
#albums {
      padding-top:20px
}
#albums li p.thumbnail {
      padding-bottom:0;
}
#albums li p.thumbnail img {
      top:-20px;
}
```

Bug 6:
The footer doesn't have the bottom space.

This is mostly due to the *hasLayout* issue mentioned earlier. This is how it looks in IE7:

Give the *#footer* a layout for IE7 by adding the *zoom* property in your *style-ie7.css*:

```
#footer {
      zoom:1;
}
```

Save and refresh.

Bug 7:

The *#blog* icons aren't being displayed.

IE7 doesn't support the *inline-block* properly, so it's best to just disable the icons and leave the text only.

```
#blog div.article p.post-data a span.ico {
      display:none;
}
```

It's an acceptable compromise for IE7 - the most important content, the text, is still shown and there is no need for work arounds to display the icons.

Congratulations!

You've completed the last remaining step of this guide and tested your newly converted design in the following browsers: Firefox, Opera, Safari and Internet Explorer.

A quick recap:

- convert your project in one single browser until it's finished and then move it to other browsers for testing;
- a bug fix in one browser isn't always isolated from the other browsers, so always check your files in all browsers that you indent to support;
- implement a fallback method for IE7-8 browsers and don't go for specific hacks if they aren't that important (e.g. *a solid background color works better than a buggy and unrecommended hack to have gradients shown*)

Ending

Putting it all together.

Done.

You successfully converted *Monoplate* into functional HTML/CSS file.

What are the steps for a new project?

1. Analyze the design to have the big picture on structure, how it works etc.;

2. Crop what can not be reproduced with CSS, but always try to find if there is a mix between CSS properties and cropped portions;

3. Always crop the images to the most minimum functional size;

4. It helps a lot if you develop a naming system or do a list on how you're going to name your containers;

5. Always save your files - regularly;

6. Do your coding in one browser until it's fully converted and then switch over to the other browsers you're going to support and test if you have the same results;

7. Start with the width limit of the design (*if any*) and move towards each portion from top to bottom;

8. Adopt fallback methods for Internet Explorer browsers (usually 7 and 8) rather than pursuing a pixel-perfect design;

9. Save your files - seriously;

Thank you.