5. How do you optimize a website’s assets?

Ans. Here’s the list of rules in question:

##### **Basic Optimization Rules**

*Decrease Download Sizes*

Decreasing download sizes isn’t even in Yahoo!’s list of rules — probably because it’s so obvious. However I don’t think it hurts to reiterate the point — let’s call it Rule #0.

When we look at a simple web page we see:

* some HTML code
* different page components (assets) referenced by the HTML

The assets are images, scripts, styles, and perhaps some external media such as Flash movies or Java applets (remember those?). So, when it comes to download sizes, you should aim to have all the assets as lightweight as possible — advice which also extends to the page’s HTML content. Creating lean HTML code often means using better (semantic) markup, which also overlaps with the SEO (search engine optimization) efforts that are a necessary part of the site creation process. As most professional web developers know, a key characteristic of good markup is that it only describes the content, not the presentation of the page (no layout tables!). Any layout or presentational elements should be moved to CSS.

Here’s an example of a good approach to HTML markup for a navigation menu:

<ul id="menu">

<li><a href="home.html">Home</a></li>

<li><a href="about.html">About</a></li>

<li><a href="contact.html">Contact</a></li>

</ul>

This sort of markup should provide “hooks” to allow for the effective use of CSS and make the menu look however you want it to — whether that means adding fancy bullets, borders, or rollovers, or placing the menu items into a horizontal menu. The markup is minimal, which means there are fewer bytes to download; it’s semantic, meaning it describes the content (a navigation menu is a list of links); and finally, being minimal, it also gives you an SEO advantage: it’s generally agreed that search engines prefer a higher content-to-markup ratio in the pages that they index.

Once you’re sure your markup is lightweight and semantic, you should go through your assets and make sure they are also of minimal size. For example, check whether it’s possible to compress images more without losing too much quality, or to choose a different file format that gives you better compression. Tools such as [PNGOUT](http://advsys.net/ken/utils.htm) and [pngcrush](http://pmt.sourceforge.net/pngcrush/) are a good place to start.

*Make Fewer HTTP Requests*

Making fewer HTTP requests turns out to be [the most important optimization technique, with the biggest impact](http://yuiblog.com/blog/2006/11/28/performance-research-part-1/). If your time is limited, and you can only complete one optimization task, pick this one. HTTP requests are generally the most “expensive” activity that the browser performs while displaying your page. Therefore, you should ensure that your page makes as few requests as possible.

How you can go about that, while maintaining the richness of your pages?

* Combine scripts and style sheets: Do you have a few <script> tags in your head? Well, merge the .js files into one and save your visitors some round trips; then do the same with the CSS files.
* Use image sprites: This technique allows you to combine several images into one and use CSS to show only the part of the image that’s needed. When you combine five or ten images into a single file, already you’re making a huge [saving in the request/response overhead](http://www.alistapart.com/articles/sprites).
* Avoid redirects: a redirect adds another client-server round trip, so instead of processing your page immediately after receiving the initial response, the browser will have to make another request and wait for the second response.
* Avoid frames: if you use frames, the browser has to request at least three HTML pages, instead of just one — those of the frameset as well as each of the frames.

You’ve got the basics now. In summary, make your page and its assets smaller in size, and use fewer assets by combining them wherever you can. If you concentrate on this aspect of optimization only, you and your visitors will notice a significant improvement.

Now let’s explore some of the Yahoo! recommendations in more detail, and see what other optimizations can be made to improve performance.

##### **Optimizing Assets**

*Use a Content Delivery Network*

A [Content Delivery Network (CDN)](http://en.wikipedia.org/wiki/Content_Delivery_Network) is a network of servers in different geographical locations. Each server has a copy of a site’s files. When a visitor to your site requests a file, the file is delivered from the nearest server (or the one that’s experiencing the lightest load at the time).

This setup can have a significant impact on your page’s overall performance, but unfortunately, using a CDN can be pricey. As such, it’s probably not something you’d do for a personal blog, but it may be useful when a client asks you to build a site that’s likely to experience high volumes of traffic. Some of the most widely known CDN providers are [Akamai](http://www.akamai.com/) and [Amazon, through its S3 service](http://www.amazon.com/gp/browse.html?node=16427261).

There are some non-profit CDNs in the market; check [the CDN Wikipedia article](http://en.wikipedia.org/wiki/Content_Delivery_Network) to see if your project might qualify to use one of them. For example, one free non-profit peer-to-peer CDN is [Coral CDN](http://www.coralcdn.org/), which is extremely easy to integrate with your site. For this CDN, you take a URL and append “nyud.net” to the hostname. Here’s an example:

http://example.org/logo.png

becomes:

http://example.org.nyud.net/logo.png

*Host Assets on Different Domains but Reduce DNS Lookups*

After your visitor’s browser has downloaded the HTML for a page and figured out that a number of components are also needed, it begins downloading those components. [Browsers restrict the number of simultaneous downloads that can take place](http://yuiblog.com/blog/2007/04/11/performance-research-part-4/); as per the HTTP/1.1 specification, the limit is two assets per domain.

Because this restriction exists on a per-domain basis, you can use several domains (or simply use subdomains) to host your assets, thus increasing the number of parallel downloads. Most shared hosts will allow you to create subdomains. Even if your host places a limit on the number of subdomains you can create (some restrict you to a maximum of five), it’s not that important, as you won’t need to utilize too many subdomains to see some noticeable performance improvements.

However, as [Rule #9](http://developer.yahoo.com/performance/rules.html#dns_lookups) states, you should also reduce the number of DNS lookups, because these can also be expensive. For every domain or subdomain that hosts a page asset, the browser will need to make a DNS lookup. So the more domains you have, the more your site will be slowed down by DNS lookups. Yahoo!’s research suggests that two to four domains is an optimal number, but you can decide for yourself what’s best for your site.

As a general guideline, I’d suggest you use one domain to host HTML pages and two other domains for your assets. Here’s an example:

* www.sitepoint.com – hosts only HTML (and maybe content images)
* i1.sitepoint.com – hosts JS, CSS, and some images
* i2.sitepoint.com – hosts most of the site’s images

Different hosting providers will probably offer different interfaces for creating subdomains, and ideally they should provide you with an option to specify the directory that holds the files for the subdomain. For example, if your canonical domain is www.sitepoint.com, and it points to /home/sitepoint/htdocs, ideally you should be able to create the subdomain i1.sitepoint.com (either via an administration control panel or by creating a symbolic link in the file system) and point it to the same folder, /home/sitepoint/htdocs. This way, you can keep all files in the same location, just as they are in your development environment, but reference them using a subdomain.

However, some hosts may prevent you from creating subdomains, or may restrict your ability to point to particular locations on the file system. In such cases, your only real options is to physically copy the assets to the new location. Don’t be tempted to create some kind of redirect in this case — it will only make things worse, as it creates two requests for each image.

If your hosting provider doesn’t allow subdomains at all, you always have the option of buying more domains and using them purely to host assets — after all, that’s what a lot of big sites do. Yahoo! uses the domain yimg.com, Amazon has images-amazon.com, and SitePoint has sitepointstatic.com. If you own several sites, or manage the hosting of your client’s sites, you might consider buying two domains, such as yourdomain-i1.com and yourdomain-i2.com, and using them to host the components for all the sites you maintain.

*Place Assets on a Cookie-free Domain*

If you set a lot of cookies, the request headers for your pages will increase in size, since those cookies are sent with each request. Additionally, your assets probably don’t use the cookies, so all of this information could be repeatedly sent to the client for no reason. Sometimes, those headers may even be bigger than the size of the asset requested — these are extreme cases of course, but it happens. Consider downloading those small icons or smilies that are less than half a kB, and requesting them with 1kB worth of HTTP headers.

If you use subdomains to host your assets, you need to make sure that the cookies you set are for your canonical domain name (e.g. www.example.org) and not for the top-level domain name (e.g. example.org). This way, your asset subdomains will be cookie-free. If you’re attempting to improve the performance of an existing site, and you’ve already set your cookies on the top-level domain, you could consider the option of hosting assets on new domains, rather than subdomains.

*Split the Assets Among Domains*

It’s completely up to you which assets you decide to host on i1.example.org and which you decide to host on i2.example.org — there’s no clear directive on this point. Just make sure you don’t randomize the domain on each request, as this will cause the same assets to be downloaded twice — once from i1 and once from i2.

You could aim to split your assets evenly by file size, or by some other criterion that makes sense for your pages. You may also choose to put all content images (those that are included in your HTML with <img /> tags) on i1 and all layout images (those referenced by CSS’s background-image:url()) on i2, although in some cases this solution may not be optimal. In such cases, the browser will download and process the CSS files and then, depending on which rules need to be applied, will [selectively download only images that are needed by the style sheet](http://www.phpied.com/smart-browsers-dont-download-unneeded-images/). The result is that the images referenced by CSS may not download immediately, so the load on your asset servers may not be balanced.

The best way to decide on splitting assets is by experimentation; you can use [Firebug](http://www.getfirebug.com/)‘s Net panel to monitor the sequence in which assets download, then decide how you should spread components across domains in order to speed up the download process.

*Configure DNS Lookups on Forums and Blogs*

Since you should aim to have no more than four DNS lookups per page, it may be tricky to integrate third-party content such as Flickr images or ads that are hosted on a third-party server. Also, hotlinking images (by placing on your page an <img /> tag whose src attribute points to a file on another person’s server) not only steals bandwidth from the other site, but also harms your own page’s performance, causing an extra DNS lookup.

If your site contains user-generated content (as do forums, for example), you can’t easily prevent multiple DNS lookups, since users could potentially post images located anywhere on the Web. You could write a script that copies each image from a user’s post to your server, but that approach can get fairly complicated.

Aim for the low-hanging fruit. For example, in the [phpBB forum software](http://www.phpbb.com/), you can configure whether users need to hotlink their avatar images or upload them to your server. In this case, uploaded avatars will result in better performance for your site.

*Use the Expires Header*

For best performance, your static assets should be exactly that: static. This means that there should be no dynamically generated scripts or styles, or <img> tags pointing to scripts that generate dynamic images. If you had such a need — for example, you wanted to generate a graphic containing your visitor’s username — the dynamic generation could be taken “offline” and the result cached as a static image. In this example, you could generate the image once, when the member signs up. You could then store the image on the file system, and write the path to the image in your database. An alternative approach might involve scheduling an automated process (a cron job, in UNIX) that generates dynamic components and saves them as static files.

Having assets that are entirely static allows you to set the Expires header for those files to a date that is far in the future, so that when an asset is downloaded once, it’s cached by the browser and never requested again (or at least not for a very long time, as we’ll see in a moment).

Setting the Expires header in Apache is easy: add an .htaccess file that contains the following directives to the root folder of your i1 and i2 subdomains:

ExpiresActive On

ExpiresDefault "modification plus 10 years"

The first of these directives enables the generation of the Expires header. The second sets the expiration date to 10 years after the file’s modification date, which translates to 10 years after you copied the file to the server. You could also use the setting “access plus 10 years”, which will expire the file 10 years after the user requests the file for the first time.

If you want, you can even set an expiration date per file type:

ExpiresActive On

ExpiresByType application/x-javascript "modification plus 2 years"

ExpiresByType text/css "modification plus 5 years"

For more information, check the [Apache documentation on mod\_expires](http://httpd.apache.org/docs/2.0/mod/mod_expires.html).

*Name Assets*

The problem with the technique that we just looked at (setting the Expires header to a date that’s far into the future) occurs when you want to modify an asset on that page, such as an image. If you just upload the changed image to your web server, new visitors will receive the updated image, but repeat visitors won’t. They’ll see the old cached version, since you’ve already instructed their browser never to ask for this image again.

The solution is to modify the asset’s name — but it comes with some maintenance hurdles. For example, if you have a few CSS definitions pointing to img.png, and you modify the image and rename it to img2.png, you’ll have to locate all the points in your style sheets at which the file has been referenced, and update those as well. For bigger projects, you might consider writing a tool to do this for you automatically.

You’ll need to come up with a naming convention to use when naming your assets. For example, you might:

* Append an epoch timestamp to the file name, e.g. img\_1185403733.png.
* Use the version number from your source control system (cvs or svn for example), e.g. img\_1.1.png.
* Manually increment a number in the file name (e.g. when you see a file named img1.png, simply save the modified image as img2.png).

There’s no one right answer here — your decision will be depend on your personal preference, the specifics of your pages, the size of the project and your team, and so on.

If you use CVS, here’s a little PHP function that can help you extract the version from a file stored in CVS:

function getVersion($file) {

$cmd = 'cvs log -h %s';

$cmd = sprintf($cmd, $file);

exec($cmd, $res);

$version = trim(str\_replace('head: ', '', $res[3]));

return $version;

}

// example use

$file = 'img.png';

$new\_file = 'img\_' . getVersion($file) . '.png';

*Serve gzipped Content*

Most modern browsers understand gzipped (compressed) content, so a well-performing page should aim to serve all of its content compressed. Since most images, swf files and other media files are already compressed, you don’t need to worry about compressing them.

You do, however, need to take care of serving compressed HTML, CSS, client-side scripts, and any other type of text content. If you make XMLHttpRequests to services that return XML (or JSON, or plain text), make sure your server gzips this content as well.

If you open the Net panel in Firebug (or use [LiveHTTPHeaders](http://livehttpheaders.mozdev.org/) or some other packet sniffer), you can verify that the content is compressed by looking for a Content-Encoding header in the response, as shown in the following example:

Example request:

GET /2.2.2/build/utilities/utilities.js HTTP/1.1

Host: yui.yahooapis.com

User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.1.5) Gecko/20070713 Firefox/2.0.0.5

Accept-Encoding: gzip,deflate

Example response:

HTTP/1.x 200 OK

Last-Modified: Wed, 18 Apr 2007 17:36:33 GMT

Vary: Accept-Encoding

Content-Type: application/x-javascript

Content-Encoding: gzip

Cache-Control: max-age=306470616

Expires: Sun, 16 Apr 2017 00:01:52 GMT

Date: Mon, 30 Jul 2007 21:18:16 GMT

Content-Length: 22657

Connection: keep-alive

In this request, the browser informed the server that it understands gzip and deflate encodings (Accept-Encoding: gzip,deflate) and the server responded with gzip-encoded content (Content-Encoding: gzip).

There’s one gotcha when it comes to serving gzipped content: you must make sure that proxies do not get in your way. If an ISP’s proxy caches your gzipped content and serves it to all of its customers, chances are that someone with a browser that doesn’t support compression will receive your compressed content.

To avoid this you can use the Vary: Accept-Encoding response header to tell the proxy to cache this response only for clients that send the same Accept-Encoding request header. In the example above, the browser said it supports gzip and deflate, and the server responded with some extra information for any proxy between the server and client, saying that gzip-encoded content is okay for any client that sends the same Accept-Encoding content.

There is one additional problem here: some browsers (IE 5.5, IE 6 SP 1, for instance) claim they support gzip, but can actually experience problems reading it (as described on the [Microsoft downloads site](http://www.microsoft.com/downloads/details.aspx?familyid=85BB441A-5BB1-4A82-86EC-A249AF287513&displaylang=en), and the [support site](http://support.microsoft.com/kb/871205)). If you care about people using these browsers (they usually account for less than 1% of a site’s visitors) you can use a different header — Cache-Control: Private — which eliminates proxy caching completely. Another way to prevent proxy caching is to use the header Vary: \*.

*To gzip or to Deflate?*

If you’re confused by the two Accept-Encoding values that browsers send, think of deflate as being just another method for encoding content that’s less popular among browsers. It’s also less efficient, so gzip is preferred.

*Make Sure you Send gzipped Content*

Okay, now let’s see what you can do to start serving gzipped content in accordance with what your host allows.

Option 1: mod\_gzip for Apache Versions Earlier than 2

If you’re using Apache 1.2 and 1.3, the [mod\_gzip module](http://sourceforge.net/projects/mod-gzip/) is available. To verify the Apache version, you can check Firebug’s Net panel and look for the Server response header of any request. If you can’t see it, check you provider’s documentation or create a simple PHP script to echo this information to the browser, like so:

<?php echo apache\_get\_version(); ?>

In the Server header signature, you might also be able to see the mod\_gzip version, if it’s installed. It might look like something like this:

Server: Apache/1.3.37 (Unix) mod\_gzip/1.3.26.1a.....

Okay, so we’ve established that we want to compress all text content, PHP script output, static HTML pages, JavaScripts and style sheets before sending them to the browser. To implement this with mod\_gzip, create in the root directory of your site an .htaccess file that includes the following:

mod\_gzip\_on Yes

mod\_gzip\_item\_include mime ^application/x-javascript$

mod\_gzip\_item\_include mime ^application/json$

mod\_gzip\_item\_include mime ^text/.\*$

mod\_gzip\_item\_include file .html$

mod\_gzip\_item\_include file .php$

mod\_gzip\_item\_include file .js$

mod\_gzip\_item\_include file .css$

mod\_gzip\_item\_include file .txt$

mod\_gzip\_item\_include file .xml$

mod\_gzip\_item\_include file .json$

Header append Vary Accept-Encoding

The first line enables mod\_gzip. The next three lines set compression based on MIME-type. The next section does the same thing, but on the basis of file extension. The last line sets the Vary header to include the Accept-Encoding value.

If you want to send the Vary: \* header, use:

Header set Vary \*

Note that some hosting providers will not allow you to use the Header directive. If this is the case, hopefully you should be able to substitute the last line with this one:

mod\_gzip\_send\_vary On

This will also set the Vary header to Accept-Encoding.

Be aware that there might be a minimum size condition on gzip, so if your files are too small (less than 1kb, for example), they might not be gzipped even though you’ve configured everything correctly. If this problem occurs, your host has decided that the gzipping process overhead is unnecessary for very small files.

Option 2: mod\_deflate for Apache 2.0

If your host runs Apache 2 you can use mod\_deflate. Despite its name, mod\_deflate also uses gzip compression. To configure mod\_deflate, add the following directives to your .htaccess file:

AddOutputFilterByType DEFLATE text/html text/css text/plain text/xml application/x-javascript application/json

Header append Vary Accept-Encoding

Option 3: php.ini

Ideally we’d like Apache to handle the gzipping of content, but unfortunately some hosting providers might not allow it. If your hosting provider is one of these, it might allow you to use custom php.ini files. If you place a php.ini file in a directory, it overwrites PHP configuration settings for this directory and its subdirectories.

If you can’t use Apache’s mod\_gzip or mod\_deflate modules, you might still be able to compress your content using PHP. In order for this solution to work, you’ll have to configure your web server so that all static HTML, JavaScript and CSS files are processed by PHP. This means more overhead for the server, but depending on your host, it might be your only option.

Add the following directives in your .htaccess file:

AddHandler application/x-httpd-php .css

AddHandler application/x-httpd-php .html

AddHandler application/x-httpd-php .js

This will ensure that PHP will process these (otherwise static) files. If it doesn’t work, you can try renaming the files to have a .php extension (like example.js.php, and so on) to achieve the same result.

Now create a php.ini file in the same directory with the following content:

[PHP]

zlib.output\_compression = On

zlib.output\_compression\_level = 6

auto\_prepend\_file = "pre.php"

short\_open\_tag = 0

This enables compression and sets the compression level to 6. Values for the compression level range from 0 to 9, where 9 is the best (and slowest) compression. The last line sets up a file called pre.php to be executed at the beginning of every script, as if you had typed <?php include "pre.php"; ?> at the top of every script. You’ll need this file in order to set Content-Type headers, because some browsers might not like it when you send a CSS file that has, for example, a text/html content type header.

The short\_open\_tag setting is there to disable PHP short tags (<? ... ?>, as compared to <?php ... ?>). This is important because PHP will attempt to treat the <?xml tag in your HTML as PHP code.

Finally, create the file pre.php with the following content:

<?php

$path = pathinfo($\_SERVER['SCRIPT\_NAME']);

if ($path['extension'] == 'css') {

header('Content-type: text/css');

}

if ($path['extension'] == 'js') {

header('Content-type: application/x-javascript');

}

?>

This script will be executed before every file that has a .php, .html, .js or .css file extension. For HTML and PHP files, the default Content-Type text/html is okay, but for JavaScript and CSS files, we change it using PHP’s header function.

Option 3 (Variant 2): PHP Settings in .htaccess

If your host allows you to set PHP settings in your .htaccess file, then you no longer need to use php.ini file to configure your compression settings. Instead, set the PHP setting in .htaccess using php\_value (and php\_flag).

Looking at the modified example from above, we would have the same pre.php file, no php.ini file, and a modified .htaccess that contained the following directives:

AddHandler application/x-httpd-php .css

AddHandler application/x-httpd-php .html

AddHandler application/x-httpd-php .js

php\_flag zlib.output\_compression on

php\_value zlib.output\_compression\_level 6

php\_value auto\_prepend\_file "pre.php"

php\_flag short\_open\_tag off

Option 4: In-script Compression

If your hosting provider doesn’t allow you to use php\_value in your .htaccess file, nor do they allow you to use a custom php.ini file, your last resort is to modify the scripts to manually include the common pre.php file that will take care of the compression. This is the least-preferred option, but sometimes you may have no other alternative.

If this is your only option, you’ll either be using an .htaccess file that contains the directives outlined in Option 3 above, or you’ll have had to rename every .js and .css file (and .xml, .html, etc.) to have a .php extension. At the top of every file, add <?php include "pre.php"; ?> and create a file called pre.php that contains the following content:

<?php

ob\_start("ob\_gzhandler");

$path = pathinfo($\_SERVER['SCRIPT\_NAME']);

if ($path['extension'] == 'css') {

header('Content-type: text/css');

}

if ($path['extension'] == 'js') {

header('Content-type: application/x-javascript');

}

?>

As I indicated, this is the least favorable option of all — you should try Option 1 or 2 first, and if they don’t work, consider Option 3 or 4, or a combination of both, depending on what your host allows.

Once you’ve established the degree of freedom your host permits, you can use the technique that you’ve employed to compress your static files to implement all of your Apache-related settings. For example, earlier I showed you how to set the Expires header. Well, guess what? Some hosts won’t allow it. If you find yourself in this situation, you can use PHP’s header function to set the Expires header from your PHP script.

To do so, you might add to your pre.php file something like this:

<?php

header("Expires: Mon, 25 Dec 2017 05:00:00 GMT");

?>

*Disable ETags*

Compared to the potential hassles that can be encountered when implementing the rule above, the application of this rule is very easy. You just need to add the following to your .htaccess file:

FileETags none

Note that this rule applies to sites that are in a server farm. If you’re using a shared host, you could skip this step, but I recommend that you do it regardless because:

* Hosts change their machines for internal purposes.
* You may change hosts.
* It’s so simple.

*Use CSS Sprites*

Using a technique known as CSS sprites, you can combine several images into a single image, then use the CSS background-position property to show only the image you need. The technique is not intended for use with content images (those that appear in the HTML in <img /> tags, such as photos in a photo gallery), but is intended for use with ornamental and decorative images. These images will not affect the fundamental usability of the page, and are usually referenced from a style sheet in order to keep the HTML lean (Rule #0).

Let’s look at an example. We’ll take two images. The first is help.png; the second is rss.png. From these, we’ll create a third image, sprite.png, which contains both images.



The resulting image is often smaller in size than the sum of the two files’ sizes, because the overhead associated with an image file is included only once. To display the first image, we’d use the following CSS rule:

#help {

background-image: url(sprite.png);

background-position: -8px -8px;

width: 16px;

height: 16px;

}

To display the second image, we’d use the following rule:

#rss {

background-image: url(sprite.png);

background-position: -8px -40px;

width: 16px;

height: 16px;

}

At first glance, this technique might look a bit strange, but it’s really useful for decreasing the number of HTTP requests. The more images you combine this way, the better, because you’re cutting the request overhead dramatically. For an example of this technique in use “in the wild”, check out [this image, used on Yahoo!’s homepage](http://us.i1.yimg.com/us.yimg.com/i/ww/sp/trough_1.4.gif), or [this one from Google’s](http://www.google.com/images/nav_logo3.png).

In order to produce sprite images quickly, without having to calculate pixel coordinates, feel free to use the [CSS Sprites Generator](http://www.csssprites.com/) tool that I’ve developed. And for more information about CSS sprites, be sure to read Dave Shea’s article, titled [CSS Sprites: Image Slicing’s Kiss of Death](http://www.alistapart.com/articles/sprites).

Use Post-load Pre-loading and Inline Assets

If you’re a responsible web developer, you’re probably already adhering to [the separation of concerns](http://www.sitepoint.com/article/simply-javascript/) and using HTML for your content, CSS for presentation and JavaScript for behavior. While these distinct parts of a page should be kept in separate files at all times, for performance reasons you might sometimes consider breaking the rule on your index (home) page. The homepage should always be the fastest page on your site — many first-time visitors may leave your site, no matter what content it contains, if they find the homepage slow to load.

When a visitor arrives at your homepage with an empty cache, the [fastest way to deliver the page](http://yuiblog.com/blog/2007/01/04/performance-research-part-2/) is to have only one request and no separate components. This means having scripts and styles inline (gasp)! It’s actually possible to have inline images as well (although it’s not supported in IE) but that’s probably taking things too far. Apart from being semantically incorrect, using inline scripts and styles prevents those components from being cached, so a good strategy will be to load components in the background after the home page has loaded — a technique with the slightly confusing name of post-load preloading. Let’s see an example.

Let’s suppose that the file containing your homepage is named home.html, that numerous other HTML files containing content are scattered throughout your site, and that all of these content pages use a JavaScript file, mystuff.js, of which only a small part is needed by the homepage.

Your strategy might be to take the part of the JavaScript that’s used by the homepage out of mystuff.js and place it inline in home.html. Then, once home.html has completed loading, make a behind-the-scenes request to pre-load mystuff.js. This way, when the user hits one of your content pages, the JavaScript has already been delivered to the browser and cached.

Once again, this technique is used by some of the big boys: both Google and Yahoo! have inline scripts and styles on their homepages, and they also make use of post-load preloading. If you visit Google’s homepage, it loads some HTML and one single image — the logo. Then, once the home page has finished loading, there is a request to get [the sprite image](http://www.google.com/images/nav_logo3.png), which is not actually needed until the second page loads — the one displaying the search results.

[The Yahoo search page](http://search.yahoo.com/) performs conditional pre-loading — this page doesn’t automatically load additional assets, but waits for the user to start typing in the search box. Once you’ve begun typing, it’s almost guaranteed that you’ll submit a search query. And when you do, you’ll land on a search results page that contains some components that have already been cached for you.

Preloading an image can be done with a simple line of JavaScript:

new Image().src='image.png';

For preloading JavaScript files, use the JavaScript [include\_DOM technique](http://www.phpied.com/javascript-include/) and create a new <script> tag, like so:

var js = document.createElement('script');

js.src = 'mysftuff.js';

document.getElementsByTagName('head')[0].appendChild(js);

Here’s the CSS version:

var css = document.createElement('link');

css.href = 'mystyle.css';

css.rel = 'stylesheet';

document.getElementsByTagName('head')[0].appendChild(css);

In the first example, the image is requested but never used, so it doesn’t affect the current page. In the second example, the script is added to the page, so as well as being downloaded, it will be parsed and executed. The same goes for the CSS — it, too, will be applied to the page. If this is undesirable, you can still pre-load the assets using XMLHttpRequest.

##### **JavaScript Optimizations**

Before diving into the JavaScript code and micro-optimizing every function and every loop, let’s first look at what big-picture items we can tackle easily that might have a significant impact on a site’s performance. Here are some guidelines for improving the impact that JavaScript files have on your site’s performance:

1. Merge .js files.
2. Minify or obfuscate scripts.
3. Place scripts at the bottom of the page.
4. Remove duplicates.

*Merge .js Files*

As per the basic rules, you should aim for your JavaScripts to make as few requests as possible; ideally, this also means that you should have only one .js file. This task is as simple as taking all .js script files and placing them into a single file.

While a single-file approach is recommended in most cases, sometimes you may derive some benefit from having two scripts — one for the functionality that’s needed as soon as the page loads, and another for the functionality that can wait for the page to load first. Another situation in which two files might be desirable is when your site makes use of a piece of functionality across multiple pages — the shared scripts could be stored in one file (and thus cached from page to page), and the scripts specific to that one page could be stored in the second file.

*Minify or Obfuscate Scripts*

Now that you’ve merged your scripts, you can go ahead and minify or obfuscate them. Minifying means removing everything that’s not necessary — such as comments and whitespace. Obfuscating goes one step further and involves renaming and rearranging functions and variables so that their names are shorter, making the script very difficult to read. Obfuscation is often used as a way of keeping JavaScript source a secret, although if your script is available on the Web, it can never be 100% secret. Read more about minification and obfuscation in [Douglas Crockford’s helpful article on the topic](http://yuiblog.com/blog/2006/03/06/minification-v-obfuscation/).

In general, if you gzip the JavaScript, you’ll already have made a huge gain in file size, and you’ll only obtain a small additional benefit by minifying and/or obfuscating the script. On average, gzipping alone can result in savings of 75-80%, while gzipping and minifying can give you savings of 80-90%. Also, when you’re changing your code to minify or obfuscate, there’s a risk that you may introduce bugs. If you’re not overly worried about someone stealing your code, you can probably forget obfuscation and just merge and minify, or even just merge your scripts only (but always gzip them!).

An excellent tool for JavaScript minification is [JSMin](http://www.crockford.com/javascript/jsmin.html) and it also has a [PHP port](http://code.google.com/p/jsmin-php/), among others. One obfuscation tool is [Packer](http://dean.edwards.name/packer/) — a free online tool that, incidentally, is used by [jQuery](http://www.jquery.com/).

Changing your code in order to merge and minify should become an extra, separate step in the process of developing your site. During development, you should use as many .js files as you see fit, and then when the site is ready to go live, substitute your “normal” scripts with the merged and minified version. You could even develop a tool to do this for you. Below, I’ve included an example of a small utility that does just this. It’s a command-line script that uses the PHP port of JSMin:

<?php

include 'jsmin.php';

array\_shift($argv);

foreach ($argv AS $file) {

echo '/\* ', $file, ' \*/';

echo JSMin::minify(file\_get\_contents($file)), "n";

}

?>

Really simple, isn’t it? You can save it as compress.php and run it as follows:

$ php compress.php source1.js source2.js source3.js > result.js

This will combine and minify the files source1.js, source2.js, and source3.js into one file, called result.js.

The script above is useful when you merge and minify as a step in the site deployment process. Another, lazier option is to do the same on the fly — check out [Ed Eliot’s blog post](http://www.ejeliot.com/blog/72), and [this blog post by SitePoint’s Paul Annesley](http://www.sitepoint.com/faster-page-loads-bundle-your-css-and-javascript/) for some ideas.

Many third-party JavaScript libraries are provided in their uncompressed form as well as in a minified version. You can therefore download and use the minified versions provided by the library’s creator, and then only worry about your own scripts. Something to keep in mind is the licensing of any third-party library that you use. Even though you might have combined and minified all of your scripts, you should still retain the copyright notices of each library alongside the code.

*Place Scripts at the Bottom of the Page*

The third rule of thumb to follow regarding JavaScript optimization is that the script should be placed at the bottom of the page, as close to the ending </body> tag as possible. The reason? Well, due to the nature of the scripts (they could potentially change anything on a page), browsers block all downloads when they encounters a <script> tag. So until a script is downloaded and parsed, no other downloads will be initiated.

Placing the script at the bottom is a way to avoid this negative blocking effect. Another reason to have as few <script>tags as possible is that the browser initiates its JavaScript parsing engine for every script it encounters. This can be expensive, and therefore parsing should ideally only occur once per page.

*Remove Duplicates*

Another guideline regarding JavaScript is to avoid including the same script twice. It may sound like strange advice (why would you ever do this?) but it happens: if, for example, a large site used multiple server-side includes that included JavaScript files, it’s conceivable that two of these might double up. The duplicate script would cause the browser’s parsing engine to be started twice and possibly (in some IE versions) even request the file for the second time. Duplicate scripts might also be an issue when you’re using third party libraries. Let’s suppose you had a carousel widget and a photo gallery widget that you downloaded from different sites, and they both used jQuery. In this case you’d want to make sure that you didn’t include jQuery twice by mistake. Also, if you use [YUI](http://developer.yahoo.com/yui/articles/hosting/), make sure you don’t include a library twice by including, for example, the DOM utility (dom-min.js), the Event utility (event-min.js) and the utilities.js library, which contains both DOM and Event.

##### **CSS Optimizations**

*Merge and Minify*

For your CSS files you can follow the guidelines we discussed for JavaScripts: minify and merge all style sheets into a single file to minimize download size and the number of HTTP requests taking place. Merging all files into one is a trivial task, but the job of minification may be a bit harder, especially if you’re using CSS hacks to target specific browsers — since some hacks exploit parsing bugs in the browsers, they might also trick your minifier utility.

You may decide not to go through the hassle of minifying style sheets (and the associated re-testing after minification). After all, if you decide to serve the merged and gzipped style sheet, that’s already a pretty good optimization.

If you do decide to minify CSS, apart from the option of minifying manually (simply removing comments and whitespace), you can use some of the available tools, such as [CSSTidy](http://csstidy.sourceforge.net/), PEAR’s HTML\_CSS library (http://pear.php.net/package/HTML\_CSS/), or SitePoint’s own [Dust-me Selectors Firefox plugin](http://www.sitepoint.com/dustmeselectors/).

*Place Styles at the Top of the Page*

Your single, gzipped (and optionally minified) style sheet is best placed at the beginning of the HTML file, in the <head>section — which is where you’d usually put it anyway. The reason is that most browsers (Opera is an exception) won’t render anything on the page until the all the style sheets are duly downloaded and parsed. Additionally, none of the images referenced from the CSS will be downloaded unless the CSS parsing is complete. So it’s better to include the CSS as early on the page as possible.

You might think about distributing images across different domains, though. Images linked from the CSS won’t be downloaded until later, so in the meantime, your page can use the available download window to request content images from the domain that hosts the CSS images and is temporarily “idle”.

*Ban Expressions*

IE allows JavaScript expressions in CSS, like this one:

#content {

left: expression(document.body.offsetWidth)

}

You should avoid JavaScript expressions for a number of reasons. First of all, they’re not supported by all browsers. They also harm the “separation of concerns”. And, when it comes to performance, expressions are bad because they’re recalculated every time the page is rendered or resized, or simply when you roll your mouse over the page. There are [ways to make expressions less expensive](http://webfx.eae.net/dhtml/cssexpr/cssexpr.html) — you can cache values after they’re initially calculated, but you’re probably better off simply to avoid them.

##### **Tools for Performance Optimization**

A number of tools can help you in your performance optimization quest. Most importantly, you’d want to monitor what’s happening when the page is loaded, so that you can make informed decisions. Try these utilities:

* Firebug’s Net panel for Firefox, at [http://www.getfirebug.com](http://www.getfirebug.com/)
* YSlow, Yahoo!’s performance extension to Firebug, at <http://developer.yahoo.com/yslow/>
* LiveHTTP Headers for Firefox, at <http://livehttpheaders.mozdev.org/>
* Fiddler — for IE, but also a general-purpose packet sniffer, at <http://www.fiddlertool.com/fiddler/>
* HTTPWatch for IE (commercial, free version), at <http://www.httpwatch.com/>
* Web Inspector for Safari, at <http://webkit.org/blog/?p=41>

##### **Summary**

Whew! If you’ve made it this far, you now know quite a lot about how to approach a site optimization project (and more importantly, how to build your next web site with performance in mind). Remember the general rule of thumb that, when it comes to optimization, you should concentrate on the items with the biggest impact, as opposed to “micro-optimizing”.

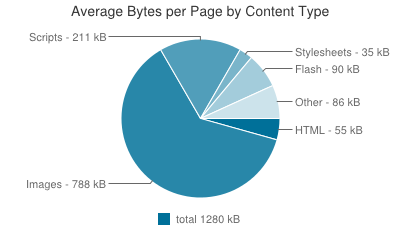
You may choose not to implement all the recommendations discussed above, but you can still make quite a difference by focusing on the really low-hanging fruit, such as:

* making fewer HTTP requests by combining components — JavaScript files, style sheets and images (by using CSS Sprites)
* serving all textual content, including HTML, scripts, styles, XML, JSON, and plain text, in a gzipped format
* minifying and placing scripts at the bottom, and style sheets at the top of your files
* using separate cookie-free domains for your components

Good luck with your optimization efforts — it’s very rewarding when you see the results!

6. What are three ways to reduce page load time?

The average size of a web page has been growing at an accelerating rate over the last few years. Research carried out by [HTTP Archive](http://httparchive.org/interesting.php#bytesperpage) in January reports that the average page size has now hit 1.25mb. For users with a decent broadband connection this is not too troublesome, but for users that still rely on slow dial-up connections (like in many developing countries) or crippled mobile data networks this poses a real issue.



Average Web Page Size 2013 [Source: HTTP Archive]

In 2011, KISSmetrics created an [infographic](http://blog.kissmetrics.com/loading-time/) showing how page load times can have a direct impact on user engagement. The data is most likely a little out of date now but the general principle is still valid.

In this blog post you are going to learn about a number of techniques that you can use to help reduce the size of your web pages and speed up your page load times.

## **Optimize Images**

One of the easiest ways to decrease the size of your web pages and increase load times is to properly resize and optimize your images. Using images that are larger than they will be displayed in your web page means that there is a lot of unnecessary data being transferred.

Once you have resized your images there are a number really handy tools like [Smush.it](http://www.smushit.com/ysmush.it/) and [TinyPNG](http://tinypng.org/) to reduce the size of the files further. These tools work by removing unnecessary data from the images and optimizing the use of colors.

## **Minify CSS & JavaScript Code**

Minifying your CSS and JavaScript code can help to reduce the size of these assets by removing unneeded whitespace and optimizing your code. Minifying your code does make it incredibly difficult for humans to read, so I would only minify when you are ready to launch a site.

[CSS Compressor](http://www.minifycss.com/css-compressor/) and [JavaScript Compressor](http://www.minifyjavascript.com/) are two handy web apps that can minify your code for you.

## **Avoid using @import**

The @import CSS rule allows you to load other CSS files into your stylesheet. Whilst this can sometimes be useful, files loaded through @import cannot be downloaded at the same time as the main stylesheet which increases the overall load time of the page.

Instead of using @import consider copying the CSS from the external file into your main stylesheet or use multiple <link> tags.

## **Reduce The Number of HTTP Requests**

Another great way of speeding up your web pages is to simply reduce the number of files that need to be loaded. There a few ways that you can do this.

### **Combine Files**

Combining multiple stylesheets into one file is a really useful way of eliminating extra HTTP requests. This strategy can also be adopted for your JavaScript files. A single larger CSS or JavaScript file will often load quicker because more time can be spent downloading the data rather than establishing multiple connections to a server.

### **Use CSS Sprites**

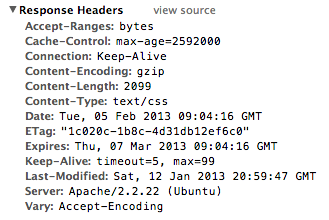
Using CSS sprites can be a great way of reducing the number of HTTP requests your page needs to load. A CSS sprite is a collection of images within a single image file. They are particularly useful for instances where you have a lot of small icons. You set the background image of an HTML element to the sprite using CSS and then use some clever CSS code to position the background image so that only the image within the sprite that you wish to use is visible.

For more information on using CSS sprites check out [this article](http://css-tricks.com/css-sprites/) on CSS-Tricks.

## **Leverage Browser Caching**

Your website probably contains files that don’t change very often. Things like images, CSS files and JavaScript assets. You can help to speed up your page load times by instructing the browse to cache a copy of these files on the hard drive so that next time the user visits your web page these resources will not need to be downloaded again.

To leverage browser caching you need to update your web server configuration to serve up resources with Expires headers. This is what tells the browser how long it should keep the cached files for.



Inspecting Response Headers in Chrome Dev Tools

The easiest way to update your server configuration for Apache servers is to use a .htaccess file. The code below shows an example configuration that would set all image, CSS and JavaScript resources to be cached for 1 month.

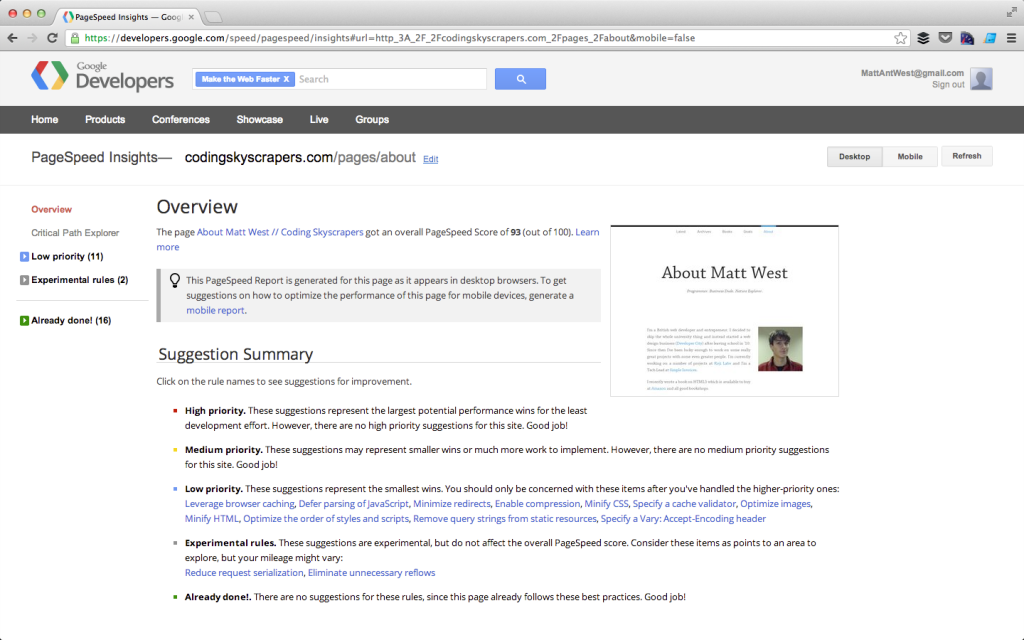
<IfModule mod\_expires.c>  
  
# Enable Expirations  
ExpiresActive On  
  
# Default Expiration Time  
ExpiresDefault "access plus 1 month"  
  
# Expiration for Images  
ExpiresByType image/gif "access plus 1 month"  
ExpiresByType image/png "access plus 1 month"  
ExpiresByType image/jpg "access plus 1 month"  
ExpiresByType image/jpeg "access plus 1 month"  
  
# Expiration for CSS  
ExpiresByType text/css "access plus 1 month”  
  
# Expiration for JavaScript  
ExpiresByType application/javascript "access plus 1 month"  
  
</IfModule>

For more information on browser caching check out [this page](https://developers.google.com/speed/docs/best-practices/caching) on the Google Developers website.

## **Tools for Testing Page Load Times**

There are some fantastic tools available that can help you test the performance of your websites. In this section you are going to learn about three of my favorites.

### **PageSpeed**

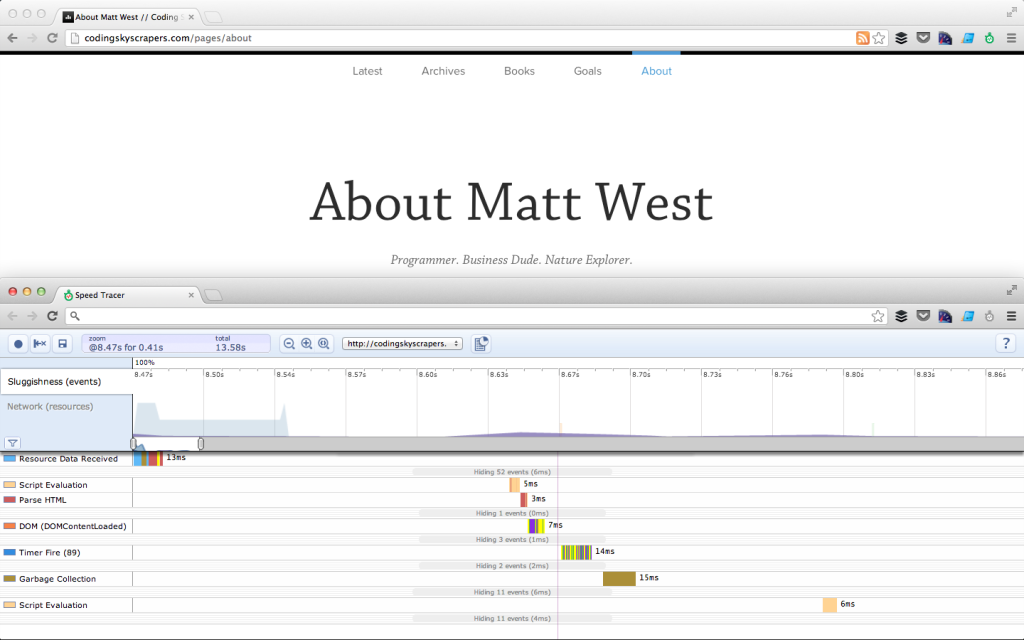


Analysing a Web Page with PageSpeed (Web App)

PageSpeed is a tool from Google that can analyze your web pages and give you pointers on what to do to improve performance. The tool will provide you with an overall ‘PageSpeed Score’ that allows you to quickly see how well optimized a page is.

You can either install PageSpeed as an [extension for Chrome or Firefox](https://developers.google.com/speed/pagespeed/insights_extensions), or use the [web app](https://developers.google.com/speed/pagespeed/insights).

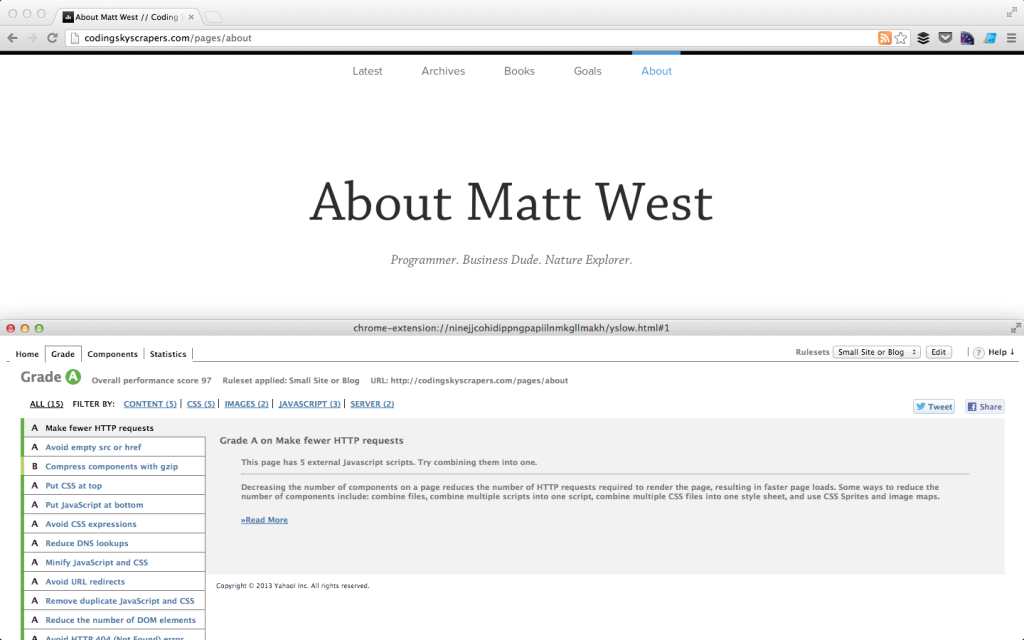
### **Speed Tracer**



Speed Tracer Chrome Extension

[Speed Tracer](https://developers.google.com/web-toolkit/speedtracer/) is another tool from Google that allows you to take a more in-depth look into the performance of your web pages. Instead of making recommendations this tool just provides you with the data that you need to identify performance bottlenecks in your web pages.

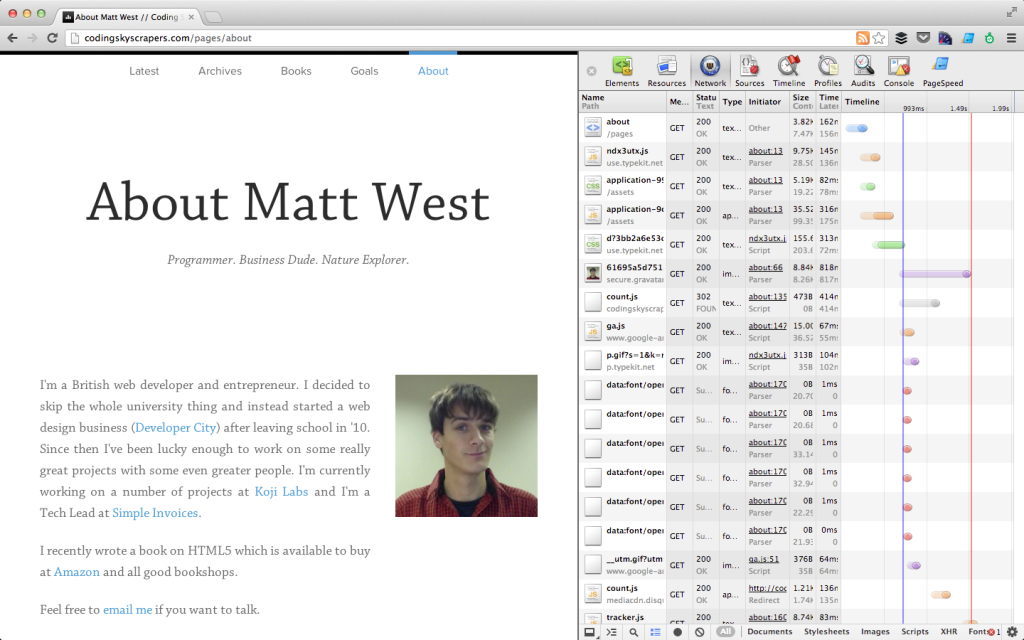
### **YSlow**



YSlow Extension in Chrome

[YSlow](http://developer.yahoo.com/yslow/) is a browser extension from Yahoo! that is very similar to PageSpeed. Running YSlow against your web pages will give you a score and a grade that can be used to track the impact of your performance optimizations. Like PageSpeed, YSlow also gives you recommendations of techniques to use to increase performance.

### **Developer Tools**



Using Chrome Dev Tools to Examine Asset Load Times

As well as using the tools mentioned above, many browsers like Chrome and FireFox come with built-in developer tools that you can use to help find areas where performance could be increased. The ‘Network’ tab in the Chrome developer tools for example can be really useful for identifying large files that take a long time to load.

## **Summary**

In this post we have covered a number of techniques that you can adopt to increase the performance of your web pages. However this has by no means been an exhaustive list. Using the tools that are outlined above you will be able to identify loads more tips for enhancing the performance of your web pages. Take a look at some of the links below for more information on enhancing the performance of your websites.

## **Useful Links**

* [YSlow](http://developer.yahoo.com/yslow/)
* [PageSpeed](https://developers.google.com/speed/pagespeed/)
* [Speed Tracer](https://developers.google.com/web-toolkit/speedtracer/)
* [Google Performance Best Practices](https://developers.google.com/speed/docs/best-practices/)
* [CSS Sprites](http://css-tricks.com/css-sprites/)

7. What kind of things must you be wary of when design or developing for multilingual sites?

# **7 Tips and Techniques For Multi-lingual Website Accessibility**

Published on:

22nd December 2010

by:

[**Emily Coward**](https://www.nomensa.com/blog/author/ecoward)

Category:

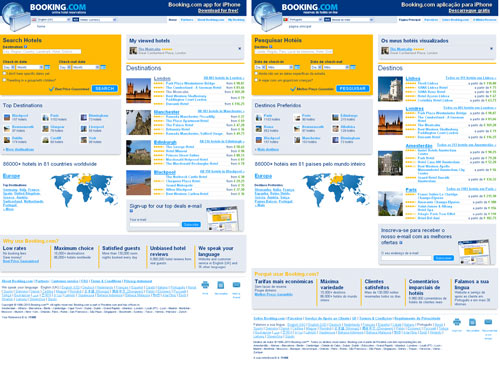
[**Web Accessibility**](https://www.nomensa.com/blog/category/web-accessibility)

[**View 7 comments**](https://www.nomensa.com/blog/2010/7-tips-for-multi-lingual-website-accessibility#comments)

If you've ever worked on or built a multi-lingual website you will know there are a million and one things to keep in mind. Sorting out domain names, web server configuration, URL structure, page layout and the translation of content are likely to be high on your 'to do' list. With all that keeping you busy, meeting the accessibility requirements for your website may slip to the bottom of the pile. This shouldn't be the case as making your multi-lingual website accessible is easy to achieve.

## **What is a multi-lingual website?**

A multi-lingual website is a website where the content is written in more than one language. The information displayed in different languages is often the same, but maybe tailored for different audiences. [**Booking.com**](http://www.booking.com/) is an example of a multi-lingual website as its content is available in 35 different languages.



Booking.com homepage shown in English (left) and Portuguese (right)

## **1: Language Codes**

The first thing to get right when working with multiple languages on a website is making sure the language is identified in the code of the page. The Web Content Accessibility Guidelines require under success criterion 3.1.1 [**Language of Page**](http://www.w3.org/TR/UNDERSTANDING-WCAG20/meaning-doc-lang-id.html) that:

*The default human language of each Web page can be programmatically determined.*

Assistive technologies such as screen readers and Braille devices can not automatically identify the language being used on a website from the text alone. The language must be identified in the code of the page in order for assistive technologies to interpret it correctly. Once recognized, these technologies can automatically switch to that language, adjusting the accent, pitch and speaking rate of the content depending on the language in question. Modern screen readers such as Jaws and Window Eyes are able to speak multiple languages in appropriate accents with proper pronunciation.

### **Setting the primary language**

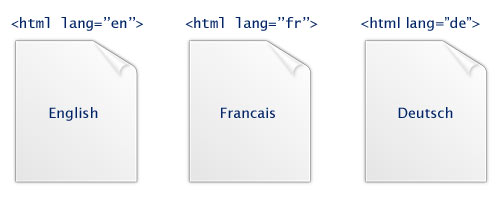
First of all you will need to choose the primary language for your pages. This is the language the majority of your content is written in. For example if your page is predominantly written in English then your primary language should be set as English. Once the primary language of your pages has been chosen, you will need to find the language code which corresponds to that language. Language codes usually consist of two letters, however four letter codes can be used for further defining the language into different dialects. A two letter language code 'en' could be used to define 'English', whereas the four letter language code 'en-GB' could be used to distinguish British English from American English 'en-US'. Please note 'en-UK' is not a valid four letter language code. Next we need to apply the language code to our page. To set the primary language of our page as English we use the 'lang' attribute along with our 'en' language code and apply this to the HTML element at the beginning of each page.

<html lang="en">

If you are using XHTML, you will need to apply an additional attribute to set the language used in an XML document. The 'xml:lang' attribute serves the same purpose as the 'lang' attribute and should use the same language code. Your pages will not pass the W3C HTML validation check without this attribute.

<html xmlns="[**http://www.w3.org/1999/xhtml**](http://www.w3.org/1999/xhtml)" lang="en" xml:lang="en">

For a multi-lingual website consisting of English, French and German pages, the primary language should be reset for each language.



The primary language of the page should correspond with the human language of the page

### **Using the wrong language code**

If you don't provide a primary language code or set the code incorrectly it may be impossible for someone using a screen reader or Braille device to understand the content. If the primary language of the web page has not been identified, screen reading software in general will read out the content in the same language as the default setting for the screen reader. So if your screen reader has English set as the default language, it will read out web page content in English. This isn’t too much of a problem if you are only viewing web pages written in English. However if the page uses the wrong language code, for instance if the content is written in English, but identified in the code as French the screen reader will attempt to speak the English content using a French accent and pronunciation. For those of you familiar with 'Allo Allo', your website could end up sounding something like Officer Crabtree. While amusing, this might not be very easy to understand.



Officer Crabtree

**2: Multiple Languages**

The majority of web pages use a single language at a time, however there maybe occasions when you want to include a language other than the primary one on your pages. If this is the case, the Web Content Accessibility Guidelines success criterion 3.1.2 Language of Parts requires that:

*The human language of each passage or phrase in the content can be programmatically determined except for proper names, technical terms, words of indeterminate language, and words or phrases that have become part of the vernacular of the immediately surrounding text.*

If you wanted to include a passage in French on your page you would need to use the 'lang' attribute to mark the change in language. The 'lang' attribute can be used with almost every HTML element, making it very easy to change languages within a page. To include a French quotation on an English page you would simply add the lang attribute to the blockquote tag:

<blockquote lang=”fr”>  
  
<p>Le plus grand faible des hommes, c'est l'amour qu'ils ont de la vie.</p>  
  
</blockquote>

### **Links**

If you are creating a multi-lingual website you may also need to provide links to the other language versions of your site. If the page you are linking to is written in a different language to the current page, you need to let people using assistive technologies know about this. This can be done by using the 'hreflang' attribute. The 'hreflang' attribute allows you to inform people the primary language of the page found when following the link is different to the current page. For example to link to a page written in French from a page written in English, you would use:

<a href="" hreflang="fr">French</a>

If you need to identify both the text of the link, and the links target as different languages you need to use both the 'lang' and 'hreflang' attributes:

<a href="" lang="fr" hreflang="fr">Francais</a>

Please note the 'hreflang' attribute should only be used for links.

## **3: Google and language recognition**

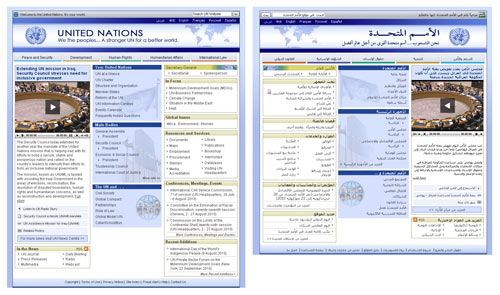
Unlike assistive technologies such as screen readers, Google does not recognise language identifiers such as 'lang' attributes in the code of the page. Google tries to work out the main languages of your pages itself. In order to make language identification easier for Google, Google recommends only using one language per page.

## **4: Language direction**

If you are creating a multi-lingual website which caters for languages written from right-to -left rather than left-to-right, you will need to make sure the direction of text is specified correctly in the code of the page. You can set the direction of text by using the 'dir' attribute on the HTML element. For languages such as Arabic, Persian and Urdu the 'dir' attribute should be set to be set to rtl (right-to-left):

<html dir="rtl">

A 'dir' attribute is not needed for pages written using left-to-right languages such as English as this is the default direction of text. Different page layouts are often required for right-to-left languages, as most right-to-left languages should be right aligned rather than left aligned. This means the page layout will need to be adapted for these languages, essentially mirroring the layout of the left to right language pages. For example the United Nations website adapts its layout for the Arabic language which is written from right-to-left. The whole layout of the page is reversed when compared to the English language version.These types of layout changes can be achieved using CSS.



The layout of the United Nations pages change depending on the language used

## **5: Character encoding**

A character encoding is essentially a key to decipher an encrypted collection of letters and symbols used in a writing system. There are many different types of character encodings so it's really important to make sure you use the right character encoding otherwise people may not be able to read the text on your pages. Character encoding also helps computers understand your information, if you use the wrong encoding your pages may not be found by some search engines. The most widely used character encoding is 'Unicode'. 'Unicode' contains characters for most languages and scripts in the world and is supported on a large number of operating systems. This means Unicode can display multiple languages and scripts within a page, which makes it an excellent choice to use for multi-lingual websites. To specify Unicode for pages written in HTML 4 put the following line in the head of your pages:

<meta http-equiv="Content-Type" content="text/html;charset=UTF-8">

For HTML 5 use:

<meta charset="UTF-8">

Please see '[**Declaring character encoding in HTML**](http://www.w3.org/International/questions/qa-html-encoding-declarations)' for more information about character encoding.

## **6: Font sizes**

When designing your multi-lingual website, it is important to realise that the font size you chose for your default language may not be suitable for all languages. Different languages such as Chinese, Japanese and Arabic might be difficult to read at font sizes that are suitable for English, French and German languages. For web pages displaying Chinese, Japanese or Arabic languages the default font size will need to be increased so the text is legible on screen. There are two ways this can be achieved. The first uses the CSS 'lang' pseudo class to set different font sizes and font families depending on the value of the 'lang' attribute: **HTML**

<html lang="en"> or <html lang="zh">

**CSS**

:lang(en) {  
  
font-size: 85%;  
  
font-family: arial, verdana, sans-serif;  
  
}  
  
:lang(zh) {  
  
font-size: 125%;  
  
font-family: helvetica, verdana, sans-serif;  
  
}

This technique is supported in Firefox, Opera and Internet Explorer 8 and higher. Chrome and Safari do not support this pseudo class. If you want to support web-kit browsers and earlier versions of Internet Explorer as well, the second option would be to use classes on the body element for each language required: **HTML**

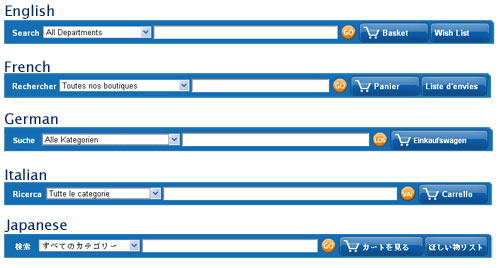
<body class="english"> or <body class="chinese">

**CSS**

.english {  
  
font-size: 85%;  
  
font-family: arial, verdana, sans-serif;  
  
}  
  
.chinese {  
  
font-size: 125%;  
  
font-family: helvetica, verdana, sans-serif;  
  
}

## **7: Length of words**

The length of words varies from language to language. Content written in one language may take up more or less space on the page than another language. The design of the website should cater for different length words used through the site. Taking Amazon as an example, the length of content in the search bar of the website varies between languages. The word 'search' takes up 10 characters in French but only two characters in Japanese. The word 'basket' takes up 6 characters in English but when translated to German takes up a massive 13 characters. Amazon have adapted the design of this area of their pages, removing the wish list button from the search bar for those languages which use longer words such as German and Italian.



Different languages take up different amounts of space on the page

Depending on the content on your multi-lingual website, it may not be possible to change the layout and design of your site in this way. You can overcome these types of problems by using shorter words to fit in to the space available on your page and making sure you have your content translated before making essential design decisions.

## **References**

* [**http://www.w3.org/International/questions/qa-lang-why**](http://www.w3.org/International/questions/qa-lang-why)