# Text Content

**By** Dave Gash
Dave is a Tech Writer

Text: letters and numbers, words and phrases, sentences and paragraphs. It's how we convey most of the meaning in web pages. Text content informs, describes, explains concepts and procedures to our readers (not just "visitors"). It is the very basis of web communication.

Text content on the web always has some structure – even if only top-down, and some formatting – even if only by default. It can also exhibit behavior, moving or changing, appearing or disappearing in response to a reader's action or the author's intent. But in and of itself, text content doesn't have these qualities and abilities; its structure, appearance, and behavior are implemented and affected by other text-based resources: HTML, CSS, and JavaScript code.

In a web page, every character of that content, structure, formatting, and behavior must be fetched from the server and downloaded to the browser, a decidedly non-trivial task. In this section we'll look at some effective methods for speeding up text content loading.

## Separate Development from Deployment

As you reduce the size of text resources and take other actions that affect their readability, it's important to remember that once you modify a chunk of code for deployment, you usually can't read it any longer, let alone maintain it. Always keep development and deployment files separate to avoid replacing a development file with a deployment version. Although, if it does happen by accident, a code beautifier or "unminifier" (for example, http://unminify.com/) might save the day.

## Minify Your Code

One simple and effective method is *minification*, which is essentially compressing a text resource by removing its whitespace and unnecessary characters without changing its validity or functionality. It doesn't sound that useful, but it is. For example, this little function (part of a table-sorting script) initially contains 348 characters.

```
function sortables_init() {
    // Find all tables with class sortable and make them sortable
    if (!document.getElementsByTagName) return;
    var tbls = document.getElementsByTagName("table");
    for (ti=0;ti<tbls.length;ti++) {
     thisTbl = tbls[ti];
     if ((' '+thisTbl.className+' ').indexOf("sortable") != -1) && (thisTbl.id))
         ts_makeSortable(thisTbl);
     }
    }
}
```

After minification, it looks like this, and only contains 257 characters.

```
function sortables_init(){if(!document.getElementsByTagName)return;var tbls
getElementsByTagName("table");for(ti=0;ti<tbls.length;ti++){thisTbl=tbls[ti];
if((''+thisTbl.className+'').indexOf("sortable")!=-1)&&(thisTbl.id)){ts_makeSort
```

Sure, we can't read it now, but the browser still can. That's a 26% reduction in size and required download time; pretty significant, even for this small sample.

Looking at the bigger picture, the entire script block from which that snippet came is over 10k, but after minification it's down to 5,411 characters, a whopping 48% reduction.

HTML and CSS can be minified in the same way, so that you can achieve improved load times for both formatting- and behavior-related code.

Many (many!) online and desktop minification tools are available; one of the most popular and highly recommended online tools, due to its longevity and stability, is Kangax HTML Minifier, which offers a broad array of output customization options for the minified code.

Other minification tools include:

- Minifier: An online tool minifies JavaScript or CSS via copy and paste.
- HTML Minifier: This online tool also handles HTML, and automatically identifies the code type(s).
- Node module for Grunt: An NPM minification package that integrates into the Grunt workflow.
- Node module for Gulp: An NPM minification package that integrates into the Gulp workflow.
- Node module for HTML Minifier: An NPM package that includes a useful chart comparing its compression results with other methods.

## Frameworks

Of course, chances are good that you do (or will) use a framework, IDE, or other structured environment as you write, rather than copy/pasting your code into a web app one file at a time. Most modern systems have built-in facilities for keeping development files separate from deployment during the build process, and are able to perform a variety of transformations, such as minification, along the way.

For example, a Gulp development-to-deployment task that includes HTML minification might look like this.

```
var gulp = require('gulp');
var htmlmin = require('gulp-html-minifier');
gulp.task('minify', function() {
  gulp.src('./src/*.html') //development location
    .pipe(htmlmin({collapseWhitespace: true}))
    .pipe(gulp.dest('./dist')) //deployment location
});
```

The execution command for that task would then be:

```
gulp minify
```

(Source: npmjs)

Note that the task pipes the HTML files from their original location to the deployment location, minifying them during the build. This not only prevents potentially irreversible modification of the source files, but avoids contaminating the production environment during later development and testing.

# Compress Text Resources

So far we've talked about compression in terms of individual image and text files. But it would also be helpful if we could get our server to automatically compress entire file sets as well, and that's where Gzip comes in.

Gzip is an application (and its file format) for compressing and decompressing files. Like solo file compression, it reduces the time required to deliver a server response by reducing the size of the resource. It is available via the GNU Project site.

https://www.gnu.org/software/gzip/

Gzip performs best on text resources, and can regularly achieve up to 70% compression (even higher for large files). However, Gzipping non-text resources, such as images, that have already been individually compressed generally provides no significant size reduction.

Unlike desktop or browser-based local compression, Gzip works at the server to identify and process certain file types that you specify. While all modern browsers support Gzip compression for HTTP requests, you must properly configure your server to deliver the compressed resource when it is requested. Different server types, of course, have different setup requirements. You configure the Apache server, for example, via the **.htaccess** file, which would include something like this.

```
<IfModule deflate_module>
    # Enable compression for the following file types
    AddOutputFilterByType           \
     DEFLATE                        \
      application/javascript        \
      text/css                      \
      text/html                     \
      text/javascript               \
      text/plain                    \
      text/xml
</IfModule>
```

BetterExplained has a very good article on Gzip compression, including background information, examples, and caveats.

So, having enabled GZip on your server, how do you know whether it's actually serving compressed files? An easy way to find out is to check it at GIDNetwork's Gzip test site.

http://www.gidnetwork.com/tools/gzip-test.php

The report includes general information about site compression, an interesting "What if" chart that shows how much compression the site would gain at various Gzip compression levels, and the response headers and page source.

Here, we ran the test on GIDNetwork's own root URL – interestingly, it did not appear to be compressed.

| | |
|---|---|
| Web page compressed? | No |
| Compression type? | **none** |
| Size, Markup (bytes) | **13,299** |
| Size, Compressed (bytes) | **13,299** |
| Compression % | **0.0** |

## What If Scenario Analysis

What if ***http://www.gidnetwork.com*** was compressed / gzip encoded?

Also, please read the article: [How do I compress my web site?]

| GZIP Compression Level | Size, Compressed (bytes) | Compression % | Download (seconds) |
|---|---|---|---|
| 0 | 13,299 | 0.0 | 0.08 |
| 1 | 4,891 | 63.2 | 0.03 |
| 2 | 4,797 | 63.9 | 0.03 |
| 3 | 4,754 | 64.3 | 0.03 |
| 4 | 4,541 | 65.9 | 0.03 |
| 5 | 4,491 | 66.2 | 0.03 |
| 6 | 4,467 | 66.4 | 0.03 |
| 7 | 4,468 | 66.4 | 0.03 |
| 8 | 4,468 | 66.4 | 0.03 |
| 9 | 4,468 | 66.4 | 0.03 |

## Response Headers

| | |
|---|---|
| HTTP Status Code | HTTP/1.1 200 OK |
| Date | Wed, 09 Aug 2017 20:48:13 GMT |
| Server | Apache/2.2.15 |
| X-Powered-By | PHP/5.3.3 |
| Set-Cookie | dex=YToyOntzOjE6Imki O2k6MDtzOjE6ImsiO047 fQ%3D%3D; expires=Fri, 08-Sep-2017 20:48:13 GMT; path=/; domain=.gidnetwork.c om |
| Connection | close |
| Transfer-Encoding | chunked |
| Content-Type | text/html |

## Page Source

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1
<html xml:lang="en" xmlns="http://www.w3.c
<title>GIDNetwork - Everything's connectec
<meta name="generator" content="GIDTemplat
<meta name="copyright" content="J de Silva
<meta http-equiv="content-type" content="t
<link rel="stylesheet" href="/global.css"
<link rel="stylesheet" href="/gid_default.
<link rel="shortcut icon" href="/favicon.i
<link rel="alternate" href="http://www.gic
<script type="text/javascript" src="http:/
<script type="text/javascript"><!--
_uacct = "UA-55009-3"; urchinTracker(); //
<script type="text/javascript"><!--
var googletag = googletag || {};
googletag.cmd = googletag.cmd || [];
(function() {
var gads = document.createElement('script'
gads.async = true;
gads.type = 'text/javascript';
var useSSL = 'https:' == document.locatior
```

Gzip can further compress already-compressed files, and that's a valid approach. In fact, to get the greatest compression ratio for text-based resources, first minify them individually prior to deployment and then compress them at delivery via a Gzip-enabled server.

## Reduce Library Use

Popular CSS and JavaScript libraries do their best to minify and compress their download files, but generally they're still pretty serious bandwidth consumers. jQuery, for example – depending on the version and the compression algorithms applied – might range from 28k to over 250k. If you need most of a given library's features, fine; but if you only need one or two specific things, you can save a lot of download time by replacing those features with single-use functions or CSS rules.

For example, a website might use jQuery's handy **`toggleClass`** feature to flip classes in order to do something specific.

```
$(el).toggleClass(className);
```

Yes, it works great and it's easy to code, but jQuery is a lot of download overhead for one effect. You might consider swapping out that huge library for a far smaller single-purpose function (source: You Might Not Need jQuery).

```
if (el.classList) {
  el.classList.toggle(className);
} else {
  var classes = el.className.split(' ');
  var existingIndex = classes.indexOf(className);

  if (existingIndex >= 0)
    classes.splice(existingIndex, 1);
  else
    classes.push(className);

  el.className = classes.join(' ');
}
```

The point is, if you don't need an entire 250k library, don't download it. Instead, find and use small, single-purpose routines that do only what you need. (And don't forget to minify them!)

You can find many interesting and to-the-point alternatives to jQuery code at You Might Not Need jQuery, which explores how the modern web has evolved to provide many of the same capabilities for which we might have previously used jQuery.

## Summary

While physically large items like images tend to get a lot of speed-improvement attention, text resources -- undoubtedly the primary content of most websites -- are often overlooked. Don't ignore text-based components, both visible and behind-the-scenes, as you look for ways to speed up your pages.

---



*Last updated July 2, 2018.*