SAMPLE CHAPTER

FAST ASP.NET WEBSITES

Dean Alan Hume

MANNING

Fast ASP.NET Websites

by Dean Alan Hume

Chapter 3

brief contents

3 Compression

This chapter covers

- The pros and cons of compression
- The types of compression available
- The sample application used in this book
- How to apply compression to a website

In this chapter we'll look at the impact compression can have on your site—optimizing front-end performance and taking you closer to your goal of a grade A performance website.

By using compression, you'll reduce the size of each HTTP request a web page makes, and each reduction will lighten, as it were, the overall weight of the page. By the end of the chapter, you'll be able to optimize your website with a number of compression techniques in the .NET web technology stack. We'll go through examples that are applicable to IIS, ASP.NET MVC, and Web Forms.

3.1 What is compression?

Compression is an algorithm that eliminates unwanted redundancy from a file in order to create one that is smaller than the original representation. If both the server and the browser understand this algorithm, it can be applied to the response

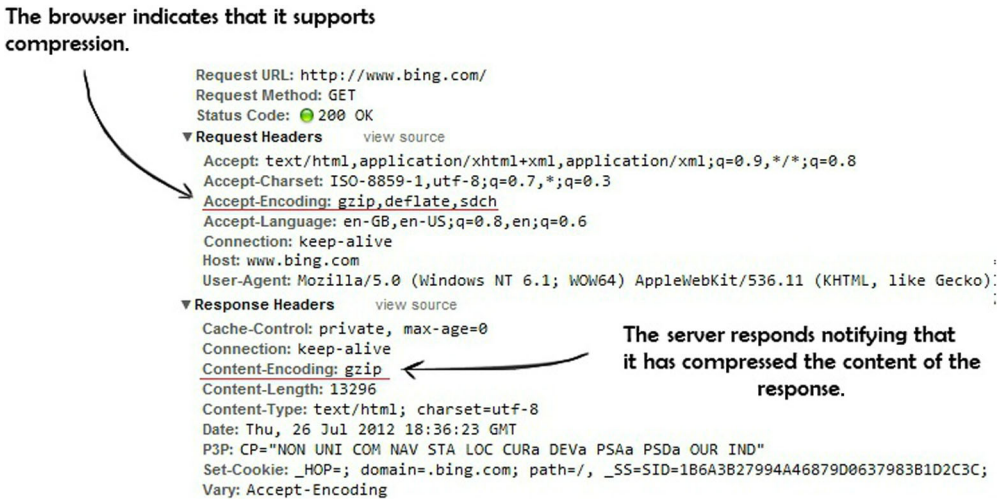


Figure 3.1 A typical HTTP request and response for www.bing.com. Notice the Accept-Encoding and Content-Encoding headers.

and request. Web browsers indicate that they support compression in the headers that are sent to the server in the HTTP request. The web server sees this header in the request and will try to compress the response that it sends back.

Compression is extremely easy to implement and is a quick win. You're about to get up and running using compression, but first it's important to understand the types of compression.

Look at the HTTP request in figure 3.1 and you'll notice that the browser sent the server a header called Accept-Encoding. This header notifies the server that it supports compression as well as the types of compression it supports, in this case, Gzip, Deflate, and SDCH. Depending on the type of compression your browser supports, the server will then compress its content accordingly and return a header in the HTTP response called Content-Encoding. In the case of figure 3.1, the server has returned a response notifying the browser that it compressed the data in Gzip format.

3.2 Why should I use compression?

In order to test the effectiveness of compression and evaluate the savings it could have on different file types, I took a few common files that you'll encounter and compressed them using Gzip. The results are shown in table 3.1.

Table 3.1 Gzip compression's effect on file sizes

File type	File size without compression	File size after compression with Gzip	Savings
HTML	6.52 KB	2.43 KB	62.74%
CSS	91.57 KB	21.12 KB	76.93%

Table 3.1 Gzip compression's effect on file sizes *(continued)*

File type	File size without compression	File size after compression with Gzip	Savings
CSS	13.51 KB	3.89 KB	71.21%
JavaScript	1.75 KB	1.18 KB	32.58%
Image	6.76 KB	6.51 KB	3.7%

In two of the files, there is a massive difference, with savings of over 70%. Gzip compression works by finding similar strings within a text file, and replacing those strings temporarily to make the overall file size smaller. This form of compression is particularly well suited for the web because HTML and CSS files usually contain plenty of repeated strings, such as whitespace, tags, and style definitions. You'll notice that the biggest savings were made on text-based files and there were hardly any gains on the image file. Image files are already compressed, so applying compression to them doesn't bring much reward.

There is also a direct correlation between the size of the file and the amount of savings (compression) that takes place. As a general rule the larger the file the greater the savings, primarily because larger files contain more whitespace and character repetition.

Chrome Developer tools allow you to compare file sizes before and after applying compression. Figure 3.2 shows these differences.

NOTE In chapter 2 you used a variety of developer tools. By hitting the F12 key on most computers, you can easily bring up the developer tools in your browser of choice.

Twitter Bootstrap is a popular CSS framework that will help you develop your CSS styles quickly so you can get your project running in no time. The CSS files contain a lot of whitespace and style tags, which make them perfect for compression. On the main bootstrap.css file the file size was cut down to 21.12 KB, saving almost 76% on the file size! This is a perfect example of why you should compress your CSS and JavaScript files.

Figure 3.2 The difference in file sizes before and after using Gzip

3.3 Pros and cons of compression

When first working with compression, some web developers will compress only the HTML file they're sending to the browser. They don't compress all the other components that make up a web page, such as the CSS, JavaScript, XML, and JSON files. It's far better to compress as many components as you can, whenever you can, to reduce the total page weight as much as possible.

There is also a cost associated with compression; the server needs to apply an algorithm to compress the files, and this can affect the CPU usage on the server. There is a bit of a tradeoff between CPU usage and the savings in file size, but if the compression settings are controlled and fine-tuned, the benefit of compressing files can far outweigh the extra CPU usage on the server.

As a general rule, it's best to compress files that are larger than 1–2 KB. Even if the server manages to compress the file by 50%, you'll save only 1 KB. Every byte counts on a web page, but the server still needs to work hard to process each file, regardless of its size. By asking the server to compress files this small, you're placing an unnecessary load on the server, which could have a detrimental effect on server response time.

You should also make sure that you don't try to compress PDF, zip, and image files, because these types of files have already been highly compressed. Processing them again on the server only adds an unnecessary load on the CPU that won't reap any rewards and can affect the client wait time. Fortunately, IIS 7+ offers a great feature that throttles CPU usage.

Throttling CPU usage means that when CPU usage gets beyond a certain level, IIS will automatically stop compressing pages. When usage drops to an acceptable level, the CPU will start compressing pages again. This feature is a great safety net because it ensures that your application won't be affected if it does come under heavy load.

IIS trades bandwidth and page load times for server CPU usage. So when the load is high enough that IIS starts throttling the CPU, the site reverts to its precompression performance levels, sacrificing the benefits of compression in order to keep the application running. You'll learn about this in more detail later in the chapter.

3.4 Types of compression

Compression capability has been built into web servers and web clients to make better use of available bandwidth and to provide faster transmission speeds between both. There are many types of compression, but almost all of them have little browser support. When you're looking at all the options, it's important not to become too worried because ultimately IIS will take care of compression for you. You'll look at the most common compression options now, which are the ones you'll most likely encounter in your day-to-day life as a developer.

3.4.1 Gzip

Gzip is a lossless data compression algorithm that compresses files and data without affecting the integrity of its contents. It allows the original data to be reconstructed

from the compressed data. When compared to other types of compression commonly used in HTTP traffic, Gzip is known to achieve higher compression ratios, resulting in smaller file sizes. To date, it's also the most widely used format because it's available freely and isn't blocked by any patents or other restrictions. Because this is the most common compression method and almost every browser that you'll encounter today supports it, this is the compression format you'll focus on in this chapter.

3.4.2 Deflate

Another type of compression you might see browsers using in the Accept-Encoding request headers is Deflate. Much like Gzip, Deflate is a lossless data compression algorithm that compresses files and data without affecting the overall integrity of its contents. There isn't a lot of browser support for this type of compression, so it isn't as widely used. You'll often see that browsers that support Deflate will also support Gzip, but because Gzip achieves a higher compression ratio and smaller file sizes, the latter is often the preferred compression method. Fortunately for IIS users, IIS 7+ supports Deflate. This widens the variety of compression methods that you're able to serve your users.

3.4.3 SDCH

A newer type of compression that has recently started to appear in the Google Chrome Browser is SDCH (Shared Dictionary Compression over HTTP). Google has proposed it as a modification to HTTP 1.1. The support for it is minimal; however, it's still worth mentioning because it's a new type of compression that could become more widespread in the future.

3.5 Accept-Encoding

Let's look at a typical HTTP response before it's decompressed by the browser. Figure 3.3 shows an HTTP response from msdn.microsoft.com.

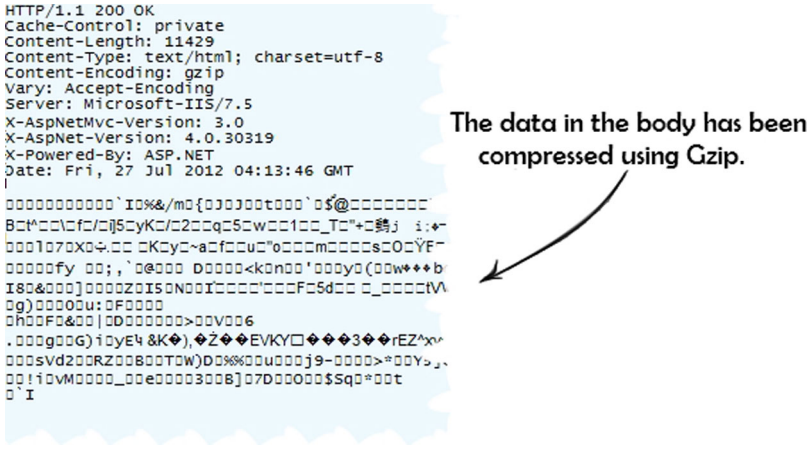


Figure 3.3 An HTTP Response for msdn.microsoft.com. The image is cropped because of its size, but note the encoded text in the response body.

As you can see, the content is encoded and is garbled. This is what the browser sees before it's decompressed and displayed on the screen.

What happens when an error occurs on the server? What happens if the browser doesn't support the compression type even if the server sends it back? Mistakes can happen and the user might see a garbled response on their screen like the data in figure 3.3. This rarely happens, but it's important to have a fallback plan for such instances. This is where the Vary header comes in.

The Vary header allows the server to return different cached versions of the page depending on whether or not the browser requested Gzip encoding. The Vary header instructs the cache to store a different version of the page if there's any variation in the indicated header. I like to think of the Vary header as fail-safe because it instructs the server to vary its cache based on the Content-Encoding type. The server then creates a separate cached copy of the page for each Content-Encoding type, which means that each Content-Encoding type requested by the browser will have a specific cached version ready for it.

3.6 The Surf Store application

To help you apply what you're learning in this book, you'll create a sample application called Surf Store in this chapter and build upon it throughout the remainder of the book. You'll be able to apply each technique you learn and run through the performance cycle that we discussed in chapter 1. The Surf Store application simulates an e-commerce store and will be similar to many sites you see online today. The sample application may seem rather basic, but its web pages contain all the elements that a standard website would have.

The Surf Store application shown in figure 3.4 contains three main pages: a Home page, an About page, and a Contact page. The Home page lists the products that are for sale and the product categories. On the left side of the Home page, you'll notice

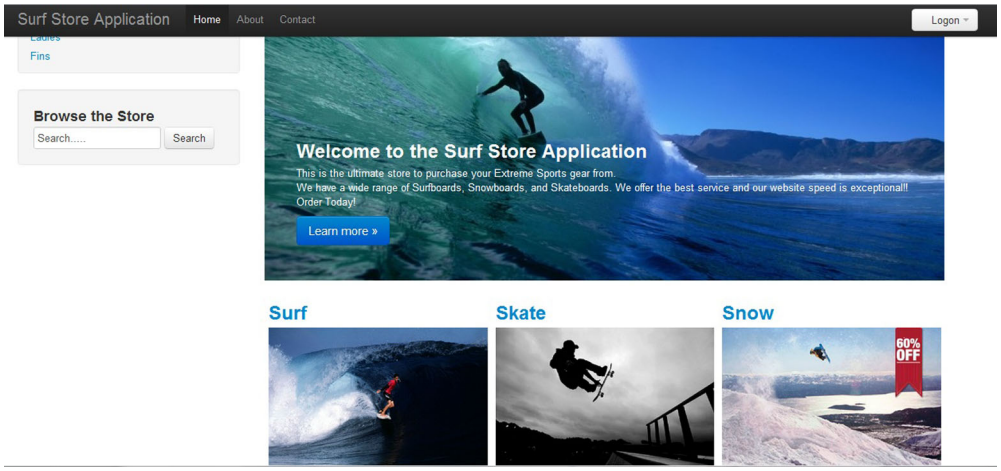


Figure 3.4 The Surf Store application

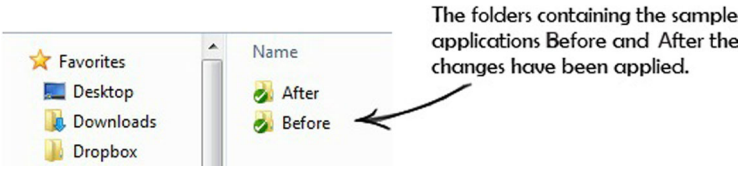


Figure 3.5 The Surf Store's download folder structure

that top-level categories are represented by links in a menu. These links, as well as the images on the page, allow you to navigate to a Product page that lists the products in that particular category. The About page contains a short description of the website, and the Contact page contains a form that allows website users to contact the Surf Store staff. Each page contains JavaScript, CSS, HTML, and images, and all of these are the front-end elements that we're trying to improve upon.

You'll have the choice of building the sample application in either ASP.NET Web Forms or ASP.NET MVC. Then you'll be able to use whichever part of the ASP.NET development framework you're most comfortable with. In order to use the Surf Store application, you'll need a copy of Visual Studio 2012 or a free copy of Visual Studio Express 2012, which is available for download at www.microsoft.com/visualstudio/11/en-us/products/express. I've made the project source code available for download on Github at https://github.com/deanhume/FastASPNetWebsites. Once you've navigated to the source code on Github, you can download all the files you need as a single zip file.

As you progress through the book, you'll apply the different optimization techniques you'll learn about in each chapter and compare the pre- and post-optimization results. Once you've downloaded the code, most chapters in the book will have a corresponding folder that contains Before and After subfolders, as shown in figure 3.5. You'll be able to use the concepts described in each chapter and apply them to the code in the Before folder. If you're unsure or become stuck, refer to the completed code in the After folder. Once you've reached chapter 12, you'll notice a dramatically different site that is very fast!

3.7 Adding compression to your website

Adding compression to your website is extremely easy and rewarding. Your hosting solution will determine which techniques to use. If you have full control of your server and are able to log into the IIS settings, it's advisable to apply the changes to the IIS settings first.

There are two different types of content that get compressed on the server: static and dynamic. Static content typically is associated with file types that don't change often:

- HTML, HTM files
- JS, CSS
- TXT
- DOC (Word)

- XLS (Excel)
- PPT (PowerPoint)

Once IIS has compressed static content, it caches it, which increases compression performance. Once the static files have been compressed, IIS 7 serves any further requests from the compressed copy in the cache directory.

Dynamic content typically is associated with file types that change often and can be produced by server-side code. Examples include JSON, XML, and ASPX. Dynamic content is processed the same way as static content; the only difference is that once it's been processed, it isn't cached to disk. Due to the nature of the files and because they could change on every request, IIS processes and compresses them each time they're requested.

3.7.1 Using IIS to add compression to a website

Now we're going to run through an example, applying compression to the Surf Store application in no time with IIS. You may not always have direct access to IIS, but using IIS is the easiest and quickest way to add compression to your site. If you've never before set up a website using IIS and would like to learn how to use your local machine as a web development server, please refer to the appendix of this book.

To start adding compression to your website, navigate to the Internet Information Services (IIS) Manager (figure 3.6).

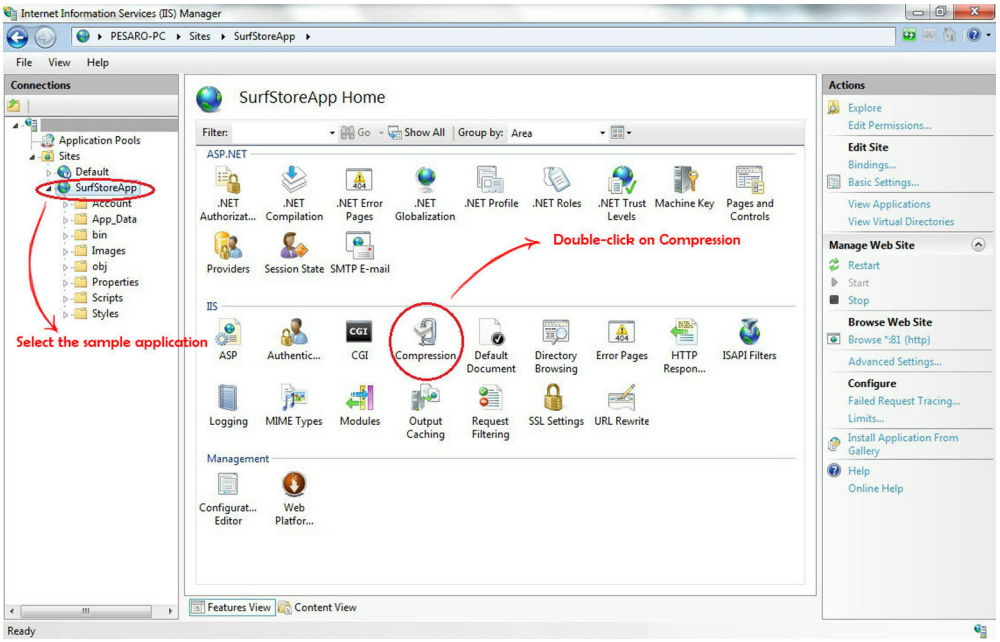


Figure 3.6 Step 1 in enabling compression in IIS 7

If you're using Windows Server 2008 or Windows Server 2008 R2:

- On the taskbar, click Start, point to Administrative Tools, then click Internet Information Services (IIS) Manager.

If you're using Windows Vista or Windows 7:

- On the taskbar, click Start, then click Control Panel.
- Double-click Administrative Tools, then double-click Internet Information Services (IIS) Manager.

Next, check the boxes to enable dynamic and static compression and click Apply (figure 3.7).

It's as simple as that! If you view the same website using your browser, you can immediately see the differences. If a client is not capable of HTTP compression, it will not pass that header and IIS 7 will always return uncompressed content. Figure 3.8 illustrates how the requested file sizes have undergone huge reductions. The largest JavaScript file (the jQuery library) went from 246.95 KB to 95.71 KB, reducing the total file size by 151.24 KB, a 61.25% savings. The CSS file that's being used as this application's base (Twitter Bootstrap) went from 97.55 KB to 21.97 KB, which works out to a 77.5% reduction of the total file size. That's a pretty impressive saving.

As you can see, the individual file sizes have been reduced, but how has this affected the overall page weight? It went from 985.55 KB to 745.29 KB, which is almost a 25% reduction in total page weight! To enable compression on the server, all you needed to do was check two check boxes, a simple act that shaved about 240 KB off the total weight of the page.

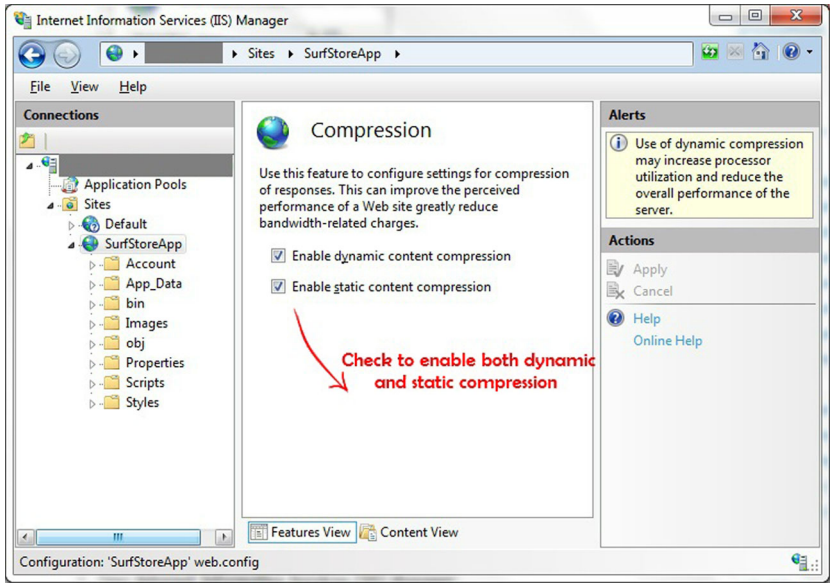


Figure 3.7 Step 2 in enabling compression in IIS 7

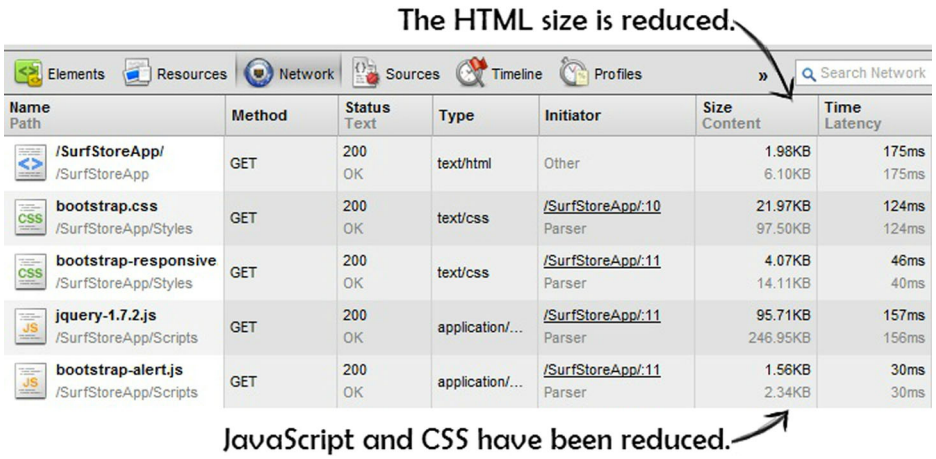
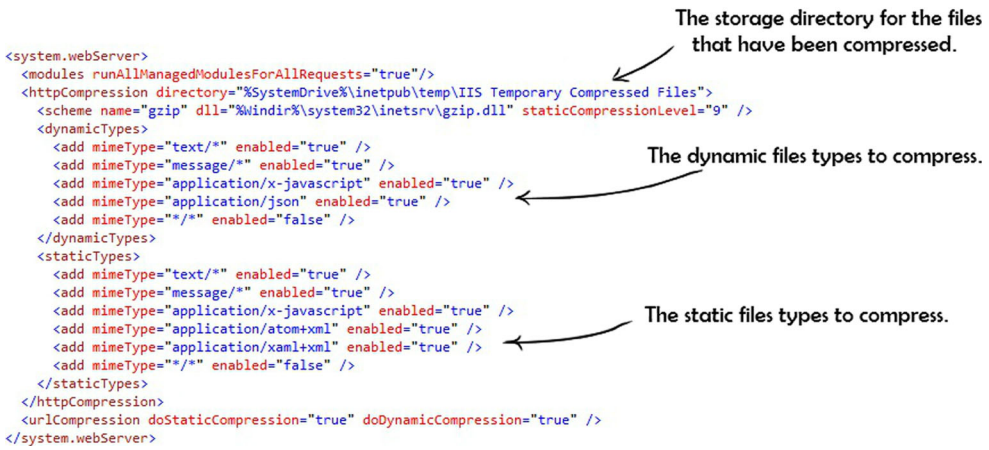


Figure 3.8 The file size savings after adding Gzip to the Surf Store application

3.7.2 Using a Web.config file to add compression to a website

Okay, so what happens when you don't have access to the server? You may be using shared hosting to host your website, or you may be working in an environment where you don't have access to IIS Manager. Fortunately, you can still configure and enable compression for your website by using the Web.config file. If you aren't already familiar with Web.config, it's a standard XML file that's included whenever you start building a new ASP.NET application. It contains all the configuration settings for the application and it allows you to control and fine-tune your application's compression settings. Figure 3.9 shows the code you'll need to add to your Web.config file in order to enable compression on the server.

You can specify the different file or MIME types that you want the server to compress under the `dynamicTypes` and `staticTypes` elements in the Web.config file. Both




```
<system.webServer>
  <modules runAllManagedModulesForAllRequests="true"/>
  <httpCompression directory="%SystemDrive%\inetpub\temp\IIS Temporary Compressed Files">
    <scheme name="gzip" dll="%Windir%\system32\inetsrv\gzip.dll" staticCompressionLevel="9" />
    <dynamicTypes>
      <add mimeType="text/*" enabled="true" />
      <add mimeType="message/*" enabled="true" />
      <add mimeType="application/x-javascript" enabled="true" />
      <add mimeType="application/json" enabled="true" />
      <add mimeType="*/*" enabled="false" />
    </dynamicTypes>
    <staticTypes>
      <add mimeType="text/*" enabled="true" />
      <add mimeType="message/*" enabled="true" />
      <add mimeType="application/x-javascript" enabled="true" />
      <add mimeType="application/atom+xml" enabled="true" />
      <add mimeType="application/xaml+xml" enabled="true" />
      <add mimeType="*/*" enabled="false" />
    </staticTypes>
  </httpCompression>
  <urlCompression doStaticCompression="true" doDynamicCompression="true" />
</system.webServer>
```

Figure 3.9 The Web.config settings that enable IIS compression

elements are visible in figure 3.9. It isn't advisable to process everything. As you'll recall, some file types are already compressed so you'll only waste valuable CPU resources if you attempt to compress them again. Remember that images and PDFs are already quite compressed, so the savings you'll gain from compressing these file types will be minimal. The most common file types that should be compressed are:

- CSS
- JavaScript
- JSON
- RSS
- HTML
- XML

The `urlCompression` element is another quick way to enable or disable compression on your website. The `urlCompression` element is shown in figure 3.9.

It only has three attributes: `doStaticCompression`, `doDynamicCompression`, and `dynamicCompressionBeforeCache`. The first two settings are simple on/off switches; the third attribute specifies whether IIS will dynamically compress content that has not been cached. When the `dynamicCompressionBeforeCache` attribute is enabled, IIS will dynamically compress the content the first time a request is made. Every request will be compressed dynamically until the response has been added to the cache directory. Once the compressed response is added to the cache directory, the cached response is sent to clients for subsequent requests.

Another compression setting that I find useful is `minFileSizeForComp`, which stands for "minimum file size for compression." Earlier in the chapter, we talked about how compression works best on larger files and how CPU levels are affected when it has to process a lot of smaller files. Using the `minFileSizeForComp` attribute allows you to set a minimum number of bytes a file must contain in order to use on-demand compression. The default size that it will compress in IIS 7.5 is 2,700 bytes and in IIS 7 it is 256 bytes.

Earlier in the chapter, I mentioned that IIS has the ability to throttle the CPU usage of the server. It gives you the ability to specify the percentage of CPU utilization at which dynamic compression will be disabled or re-enabled. You can specify the CPU percentage at which dynamic compression will be disabled by adding and setting the optional `DynamicCompressionDisableCpuUsage` attribute in your Web.config file. The attribute is set with an integer field and represents the percentage at which it will disable itself. On the opposite side, the `DynamicCompressionEnableCpuUsage` attribute specifies the percentage of CPU utilization at which dynamic compression will be re-enabled. These two attributes are useful when you have to keep CPU usage on your server to a minimum.

There are many other great attributes you can use to fine-tune your compression settings. For more information and the full list, please visit www.iis.NET/ConfigReference/system.webServer/httpCompression.

3.7.3 Adding compression with other techniques

There are other ways to add compression to your website in ASP.NET, but I wouldn't recommend them. Applying compression to your application with IIS and Web.config is the simplest and most effective way of optimizing your website. They both integrate with IIS and their settings can be maintained easily. If you're interested in other compression methods, there are a few NuGet packages, custom-written libraries, and techniques you can use to add compression to your website programmatically, but as a general rule, I try to stay clear of them. Writing code to add compression to your site can cause unwanted side effects and if there's an error, your users could be presented with a garbled page. Often you'll find that adding the compression programmatically will end up compressing only the HTML, but we want to compress as many components in the web page as possible. It's best to leave this up to the built-in server tools and let them handle the delicate details for you.

Another downside to using other compression techniques is that you can't control CPU usage or fine-tune processes the way you can with the Web.config method. With all that said, you may find yourself in a situation where you can't use the native IIS settings or your Web.config file. Proceed with caution!

3.8 The results

You've just applied compression to the Surf Store application using either IIS or Web.config. Now you can use some of the tools that were discussed in chapter 2 to monitor the results of the changes you've made. Before applying any compression to the Surf Store application, you should use the Google PageSpeed extension to generate a list of suggestions and best practices. Figure 3.10 shows that compression is at the top of the list, and that the PageSpeed score is low.

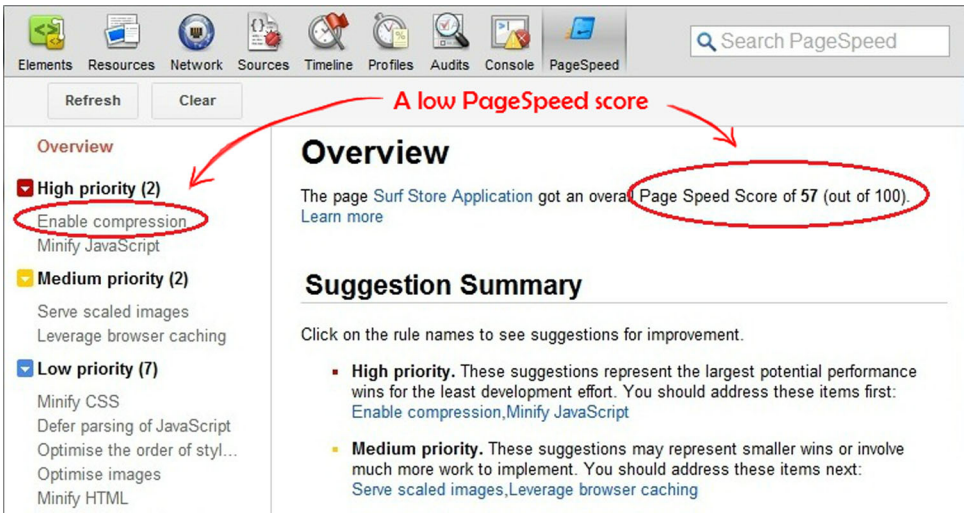


Figure 3.10 The performance score of the Surf Store application before adding compression. The Google PageSpeed tool was used to determine the score.

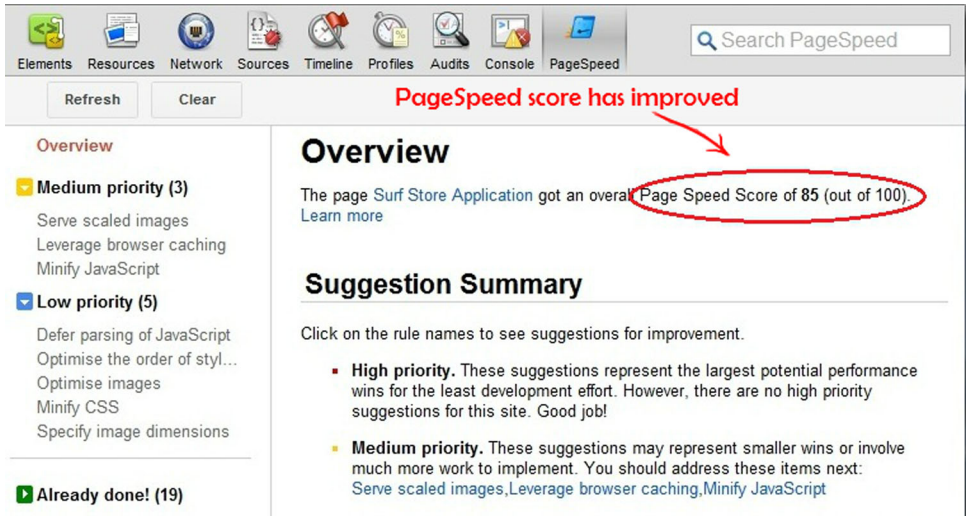


Figure 3.11 The Surf Store application's performance score, after compression, as calculated by the Google PageSpeed tool

Now that you've optimized the Surf Store application, you should run the PageSpeed tool to see how much the site's speed was improved. You can see the results in figure 3.11. After adding compression, the site jumped from a 57 PageSpeed score to an 85. You also managed to cut the total page weight down from 985.55 KB to 745.29 KB. That's a healthy reduction of 240.26 KB!

I suggest you run the Yahoo! YSlow tool on the same version of the site as a comparison. YSlow yielded similar results, as you can see in figure 3.12. The Yahoo! YSlow score jumped from a C to a B grade, and the score for adding compression jumped to an A.

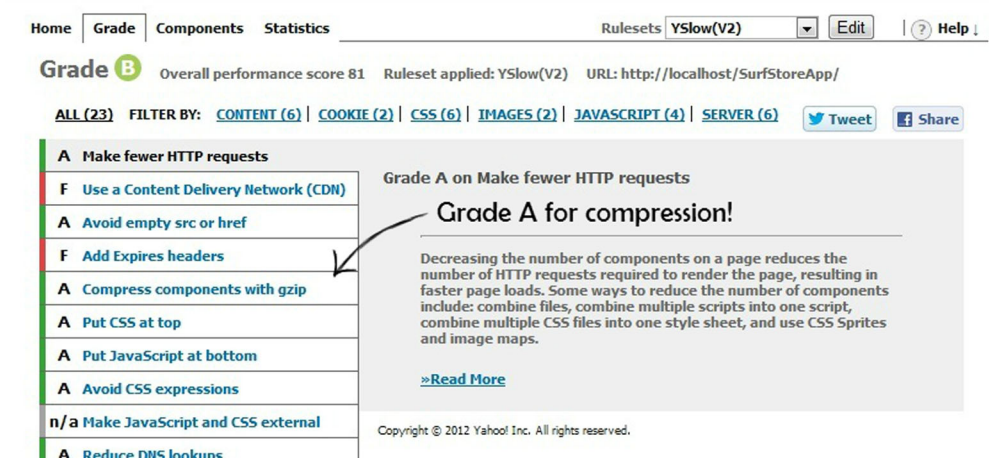


Figure 3.12 The Surf Store application's performance score as calculated by the Yahoo! YSlow tool

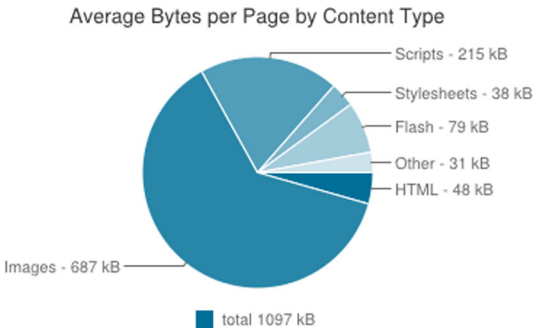


Figure 3.13 The average total page weight (source: HttpArchive.org)

You may be wondering why the total page weight originally came in at 985.55 KB and why the total weight is such a high number, even after compression.

Figure 3.13, a repeat of figure 1.3, shows us that, on average, the total weight for a web page is around 1.09 MB. The Surf Store application is intended to be as close to that number as possible, so you can simulate real-world examples when you run through these exercises.

3.9 Summary

In 2009, a Google blog[1] claimed that more than 99 human years are wasted every day because of uncompressed content. Such content wastes bandwidth and increases the amount of time users spend waiting for a web page to load. Although support for compression is a standard feature of all modern browsers, there are still many instances when users don't receive compressed content.

In this chapter you learned some simple techniques to enable compression on your website. Simply updating IIS or updating your application's Web.config settings applied compression to the Surf Store application. There was also a marked improvement in the reduction of the total page weight and the performance tools' scores.

By adding compression, you're moving one step closer to a speedy, optimized website. As you can see from the examples in this chapter, it takes almost no time to implement compression on your website. If you only add one performance feature to your website, I highly recommend that you add compression today.

[1] Arvind Jain and Jason Glasgow, "Use Compression to Make the Web Faster," Google Developers Blog, June 2013, http://mng.bz/e6bp.

ASP.NET/WEB DEVELOPMENT

FAST ASP.NET WEBSITES

Dean Alan Hume

There's a real cost to inefficient HTTP requests, overloaded data streams, and bulky scripts. Server throughput is a precious commodity, and seconds—even tiny fractions of a second—can seem like an eternity while a visitor waits for your site to load. As an ASP.NET developer, there are dozens of techniques you can apply immediately to make your sites and applications faster. You'll find them here.

Fast ASP.NET Websites delivers just what it promises—practical, hands-on guidance to create faster, more efficient ASP.NET sites and applications. This book offers step-by-step .NET-specific examples showing you how to apply classic page optimization tips, ASP.NET-specific techniques, and ways to leverage new HTML5 features.

What's Inside

- Drastically improved response times
- Tips for Webforms and ASP.NET MVC sites
- Optimizing existing pages
- .NET-specific examples

Readers should be familiar with basic HTML, CSS, and ASP.NET concepts.

Dean Hume is a software developer and blogger based in the U.K. A passionate techie, he created the ASP.NET HTML5 toolkit and blogs regularly about web performance at www.deanhume.com.

To download their free eBook in PDF, ePub, and Kindle formats, owners of this book should visit manning.com/FastASP.NETWebsites

"A clear and effective guide to the art of ASP.NET performance tuning."
—Bryn Keller, Jenkon

"Comprehensive, reader-friendly information on how to make your ASP.NET website fly."
—Danylo Kizyma
Advanced Utility Systems

"Demonstrates key concepts in clear detail."
—Michael Roberts, Sr.
Information Innovators

"An up-to-date guide ... focuses on client performance and user experience."
—Onofrio Panzarino
SBG Wolters Kluwe

ISBN 13: 978-1-617291-25-8
ISBN 10: 1-617291-25-0
53499
9 781617 291258

MANNING $34.99 / Can $36.99 [INCLUDING eBOOK]