

# **1 Web Content Compression FAQ**

## 1.1 Description

Everything you wanted to know about web content compression

## 1.2 Basics of Content Compression

Compression of outbound Web server traffic provides benefits both for Web clients who see shorter response times, as well as for content providers, who experience lower consumption of bandwidth.

Most recently, content compression for web servers has been provided mainly through use of the `gzip` encoding. Other (non perl) modules are available that provide so-called `deflate` compression. Both approaches are very similar recently and use the LZ77 algorithm combined with Huffman coding. Luckily for us, to make use of them, there is no real need for most of us to understand all the details of the obscure underlying mathematics of these techniques. Apache handlers available from CPAN can usually do the dirty work. Apache addresses content compression through handlers configured in its configuration file.

Compression is, by its nature, a content filter: It always takes its input as plain ASCII data that it converts to another binary form, and outputs the result to some destination. That is why every content compression handler usually belongs to a particular chain of handlers within the content generation phase of the request-processing flow.

A chain of handlers is one more common term that is good to know about when you plan to compress data. There are two of them recently developed for Apache 1.3: `Apache::OutputChain` and `Apache::Filter`. We have to keep in mind that the compression handler developed for one chain usually fails inside another.

Another important point deals with the order of execution of handlers in a particular chain. It's pretty straightforward in `Apache::Filter`. For example, when you configure...

```
PerlModule Apache::Filter
<Files ~ "*\*.blah">
  SetHandler perl-script
  PerlSetVar Filter On
  PerlHandler Filter1 Filter2 Filter3
</Files>
```

...the content will go through `Filter1` first, then the result will be filtered by `Filter2`, and finally `Filter3` will be invoked to make the final changes in outbound data.

However, when you configure `Apache::OutputChain` like...

```
PerlModule Apache::OutputChain
PerlModule Apache::GzipChain
PerlModule Apache::SSIChain
PerlModule Apache::PassHtml
<Files *.html>
SetHandler perl-script
  PerlHandler Apache::OutputChain Apache::GzipChain Apache::SSIChain Apache::PassHtml
</Files>
```

...execution begins with `Apache::PassHtml`. Then the content will be processed with `Apache::SSIChain` and finally with `Apache::GzipChain`. `Apache::OutputChain` will not be involved in content processing at all. It is there only for the purpose of joining other handlers within the chain.

It is important to remember that the content compression handler should always be the last executable handler in any chain.

Another important problem of practical implementation of web content compression deals with the fact that some buggy Web clients declare the ability to receive and decompress gzipped data in their HTTP requests, but fail to keep their promises when an actual compressed response arrives. This problem is addressed through the implementation of the `Apache::CompressClientFixup` handler. This handler serves the `fixup` phase of the request-processing flow. It is compatible with all known compression handlers and is available from CPAN.

## 1.3 Why it is important to compress Web content?

### *1.3.1 Reduced equipment costs and the competitive advantage of dramatically faster page loads.*

Web content compression noticeably increases delivery speed to clients and may allow providers to serve higher content volumes without increasing hardware expenditures. It visibly reduces actual content download time, a benefit most apparent to users of dialup and high-traffic connections.

Industry leaders like *Yahoo* and *Google* are widely using content compression in their businesses.

## 1.4 How much improvement can I expect?

### *1.4.1 Effective compression can achieve increases in transmission efficiency from 3 to 20 times.*

The compression ratio is highly content-dependent. For example, if the compression algorithm is able to detect repeated patterns of characters, compression will be greater than if no such patterns exist. You can usually expect to realize an improvement between of 3 to 20 times on regular HTML, JavaScript, and other ASCII content. I have seen peak HTML file compression improvements in excess of more than 200 times, but such occurrences are infrequent. On the other hand I have never seen ratios of less than 2.5 times on text/HTML files. Image files normally employ their own compression techniques that reduce the advantage of further compression.

On May 21, 2002 Peter J. Cranstone wrote to the `mod_gzip@lists.over.net` mailing list:

*"...With 98% of the world on a dial up modem, all they care about is how long it takes to download a page. It doesn't matter if it consumes a few more CPU cycles if the customer is happy. It's cheaper to buy a newer faster box, than it is to acquire new customers."*

## **1.5 How hard is it to implement content compression on an existing site?**

***1.5.1 Implementing content compression on an existing site typically involves no more than installing and configuring an appropriate Apache handler on the Web server.***

This approach works in most of the cases I have seen. In some special cases you will need to take extra care with respect to the global architecture of your Web application, but such cases may generally be readily addressed through various techniques. To date I have found no fundamental barriers to practical implementation of Web content compression.

## **1.6 Does compression work with standard Web browsers?**

***1.6.1 Yes. No client side changes or settings are required.***

All modern browser makers claim to be able to handle compressed content and are able to decompress it on the fly, transparent to the user. There are some known bugs in some old browsers, but these can be taken into account through appropriate configuration of the Web server.

I strongly recommend use of the `Apache::CompressClientFixup` handler in your server configuration in order to prevent compression for known buggy clients.

## **1.7 Is it possible to combine the content compression with data encryption?**

***1.7.1 Yes. Compressed content can be encrypted and securely transmitted over SSL.***

On the client side, the browser transparently unencrypts and uncompresses the content for the user. It is important to maintain the correct order of operations on server side to keep the transaction secure. You must compress the content first and then apply an encryption mechanism. This is the only order of operations current browsers support.

## **1.8 What software is required on the server side for content compression?**

### ***1.8.1 There are four known mod\_perl modules/packages for Web content compression available to date for Apache 1.3 (in alphabetical order):***

- **Apache::Compress**

a mod\_perl handler developed by Ken Williams (U.S.), `Apache::Compress`, can generate gzip output through the `Apache::Filter`. This module accumulates all incoming data and compresses the entire content body as a unit.

- **Apache::Dynagzip**

a mod\_perl handler developed by Slava Bizyayev, `Apache::Dynagzip` uses the gzip format to compress output dynamically through the `Apache::Filter` or through the internal Unix pipe.

`Apache::Dynagzip` is most useful when one needs to compress dynamic outbound Web content (generated on the fly from databases, XML, etc.) when content length is not known at the time of the request.

`Apache::Dynagzip`'s features include:

- **Support for both HTTP/1.0 and HTTP/1.1.**
- **Control over the chunk size on HTTP/1.1 for on-the-fly content compression.**
- **Support for Perl, Java, or C/C++ CGI applications.**
- **Advanced control over the proxy cache with the configurable `Vary` HTTP header.**
- **Optional control over content lifetime in the client's local cache with the configurable `Expires` HTTP header.**
- **Optional support for server-side caching of the dynamically generated (and compressed) content.**
- **Optional extra-light compression**

removal of leading blank spaces and/or blank lines, which works for all browsers, including older ones that cannot uncompress gzip format.

- **Apache::Gzip**

an example of the mod\_perl filter developed by Lincoln Stein and Doug MacEachern for their book *Writing Apache Modules with Perl and C* (U.S.), which like `Apache::Compress` works through `Apache::Filter`. `Apache::Gzip` is not available from CPAN. The source code may be found on the book's companion Web site at <http://www.modperl.com/>

- **Apache::GzipChain**

a mod\_perl handler developed by Andreas Koenig (Germany), which compresses output through `Apache::OutputChain` using the gzip format.

Apache::GzipChain currently provides in-memory compression only. Use of this module under perl-5.8 or higher is appropriate for Unicode data. UTF-8 data passed to Compress::Zlib::memGzip() are converted to raw UTF-8 before compression takes place. Other data are simply passed through.

## 1.9 What is the typical overhead in terms of CPU use for the content compression?

### *1.9.1 Typical CPU overhead that originates from content compression is insignificant.*

In my observations of data compression of files of up to 200K it takes less than 60 ms CPU on a P4 3 GHz processor. I could not measure the lower boundary reliably for dynamic compression, because there are no really measurable latency. From the perspective of global architecture and scalability planning, I would suggest allowing some 10 ms per request on *regular* Web pages in order to roughly estimate/predict the performance of the application server.

Estimation of connection times is an even less exact matter for a variety of possible network-related reasons. The worst-case scenario is pretty impressive: a slow dialup connection through an ISP with no proxy/buffering holds the provider's socket for a time interval proportionate to the size of the requested file. At present, gzip reduces this connection time by a factor of approximately 3-20. If the ISP buffers its traffic, however, the content provider might not feel a dramatic impact -- apart of the fact that they are paying their telecom providers for the transmission of considerable unnecessary data.

## 1.10 Is it possible to compress the output from Apache::Registry with Apache::Dynagzip?

### *1.10.1 Yes. This should be fairly easy to accomplish, as follows:*

If your page/application is initially configured like this:

```
<Directory /path/to/subdirectory>
  SetHandler perl-script
  PerlHandler Apache::Registry
  PerlSendHeader On
  Options +ExecCGI
</Directory>
```

you might replace it with the following:

```
PerlModule Apache::Filter
PerlModule Apache::Dynagzip
PerlModule Apache::CompressClientFixup
<Directory /path/to/subdirectory>
  SetHandler perl-script
  PerlHandler Apache::RegistryFilter Apache::Dynagzip
```

```
PerlSendHeader On
Options +ExecCGI
PerlSetVar Filter On
PerlFixupHandler Apache::CompressClientFixup
PerlSetVar LightCompression On
</Directory>
```

You should usually be all set after that.

In more common cases, you will need to replace the line:

```
PerlHandler Apache::Registry
```

in your initial configuration file with the following lines:

```
PerlHandler Apache::RegistryFilter Apache::Dynagzip
PerlSetVar Filter On
PerlFixupHandler Apache::CompressClientFixup
```

Optionally, you might add:

```
PerlSetVar LightCompression On
```

to reduce the size of the stream for clients unable to speak gzip (like *Microsoft Internet Explorer* over HTTP/1.0).

Finally, make sure you have somewhere declared

```
PerlModule Apache::Filter
PerlModule Apache::Dynagzip
PerlModule Apache::CompressClientFixup
```

This basic configuration uses many defaults. See `Apache::Dynagzip` POD for further fine tuning if required.

Note, however, that `Apache::RegistryFilter` is not *yet another* `Apache::Registry`. You may need to adjust your script in accordance with requirements of `Apache::Filter` first, especially when the script generates any CGI/1.1-specific HTTP headers. You can test your compatibility with the `Apache::Filter` chain using a temporary configuration like:

```
PerlModule Apache::Filter
<Directory /path/to/subdirectory>
  SetHandler perl-script
  PerlHandler Apache::RegistryFilter
  PerlSendHeader On
  Options +ExecCGI
  PerlSetVar Filter On
</Directory>
```

with no `Apache::Dynagzip` involved. See `Apache::Filter` documentation if you have any problems.

## 1.11 Is it possible to compress output from a Mason-driven application with Apache::Dynagzip?

### *1.11.1 Yes. HTML::Mason::ApacheHandler is compatible with the Apache::Filter chain.*

If your application is initially configured like:

```
PerlModule HTML::Mason::ApacheHandler
<Directory /path/to/subdirectory>
  <FilesMatch "\.html$">
    SetHandler perl-script
    PerlHandler HTML::Mason::ApacheHandler
  </FilesMatch>
</Directory>
```

you may wish to replace it with the following:

```
PerlModule HTML::Mason::ApacheHandler
PerlModule Apache::Dynagzip
PerlModule Apache::CompressClientFixup
<Directory /path/to/subdirectory>
  <FilesMatch "\.html$">
    SetHandler perl-script
    PerlHandler HTML::Mason::ApacheHandler Apache::Dynagzip
    PerlSetVar Filter On
    PerlFixupHandler Apache::CompressClientFixup
    PerlSetVar LightCompression On
  </FilesMatch>
</Directory>
```

You should be all set safely after that.

In more common cases, you will need to replace the line:

```
PerlHandler HTML::Mason::ApacheHandler
```

in your initial configuration file with the following lines:

```
PerlHandler HTML::Mason::ApacheHandler Apache::Dynagzip
PerlSetVar Filter On
PerlFixupHandler Apache::CompressClientFixup
```

Optionally, you might add:

```
PerlSetVar LightCompression On
```

to reduce the size of the stream for clients unable to speak gzip (like *Microsoft Internet Explorer* over HTTP/1.0).

Finally, make sure you have somewhere declared

```
PerlModule Apache::Dynagzip
PerlModule Apache::CompressClientFixup
```

This basic configuration uses many defaults. See `Apache::Dynagzip` POD for further fine tuning.

## 1.12 Is commercial support available for Apache::Dynagzip?

### *1.12.1 Yes. Slav Company, Ltd. provides commercial support for Apache::Dynagzip worldwide.*

Since the author of `Apache::Dynagzip` is employed by Slav Company, service is effective and consistent.

## 1.13 Why is it important to maintain a control over the chunk size?

### *1.13.1 It helps to reduce response latency.*

`Apache::Dynagzip` is the only handler to date that begins transmission of compressed data as soon as the initial uncompressed pieces of data arrive from their source, at a time when the source process may not even have completed generating the full document it is sending. Transmission can therefore take place concurrently with creation of later document content.

This feature is mainly beneficial for HTTP/1.1 requests, because HTTP/1.0 does not support chunks.

I would also mention that the internal buffer in `Apache::Dynagzip` always prevents Apache from the creating too short chunks over HTTP/1.1, or from transmitting too short pieces of data over HTTP/1.0.

## 1.14 Is it worthwhile to strip leading blank spaces prior to gzip compression?

### *1.14.1 Yes. It is usually worthwhile to do this.*

The benefits of blank space stripping are mostly significant for non-gzipped data transmissions. One can expect some 5-20% reduction in stream size on regular 'structured' HTML, JavaScript, CSS, XML, etc., in this case at negligible cost in terms of CPU overhead and response delay.

After applying gzip compression, the benefits of previously applied blank space stripping are usually reduced to some 0.5-1.0% of the resulting size, because gzip compresses blank spaces very effectively. It is still worthwhile, however, to perform blank space stripping because:

- chances are that your handler will ultimately have to send an uncompressed response back to a known buggy client;
- it really costs next-to-nothing, and every little bit helps to reduce the cost of data transmission, especially considering the cumulative effect of frequent repetitions.

## 1.15 Are there any content compression solutions for vanilla Apache 1.3?

*1.15.1 Yes. There are two compression modules written in C that are available for vanilla Apache 1.3:*

- **mod\_deflate**

an Apache handler written in C by Igor Sysoev (Russia).

- **mod\_gzip**

an Apache handler written in C, originally by Kevin Kiley, *Remote Communications, Inc.* (U.S.)

See their respective documentation for further details.

## 1.16 Can I compress the output of my site at the application level?

*1.16.1 Yes, if your Web server is CGI/1.1 compatible and allows you to create specific HTTP headers from your application, or when you use an application framework that carries its own handler capable of compressing outbound data.*

For example, vanilla Apache 1.3 is CGI/1.1 compatible. It allows development of CGI scripts/programs that can generate compressed outgoing streams accomplished with specific HTTP headers.

Alternatively, on mod\_perl enabled Apache, some application environments carry their own compression code that can be activated through appropriate configuration:

Apache : ASP does this with the CompressGzip setting;

Apache::AxKit uses the AxGzipOutput setting to do this.

See the documentation for the particular packages for details.

## 1.17 Are there any content compression solutions for Apache-2?

***1.17.1 Yes. A core compression module written in C, mod\_deflate, is available for Apache-2.***

mod\_deflate for Apache-2 was written by Ian Holsman (USA).

Despite its name, mod\_deflate for Apache-2 provides gzip-encoded content. In accordance with the concept of output filters that was introduced in Apache-2, mod\_deflate is capable of gzipping outbound traffic from any content generator, including CGI, Java, mod\_perl, etc.

- **This module supports flushing.**
- **It is output filter-compatible.**
- **It has its own set of configuration options to maintain control over buggy clients.**

## 1.18 When is Apache::Dynagzip supposed to be ported to Apache-2?

***1.18.1 There are no current plans to port Apache::Dynagzip to Apache-2:***

mod\_deflate for Apache-2 is capable of providing all basic functionality required for effective dynamic content compression. The rest can be easily addressed through implementation of the accompanying specific, tiny filters. For instance, Apache::Clean, which is already ported to Apache-2, can be used to strip unnecessary blank spaces from outbound streams.

## 1.19 Where can I read the original descriptions of the gzip and deflate formats?

***1.19.1 gzip format is published as rfc1952, and deflate format is published as rfc1951.***

You can find many mirrors of RFC archives on the Internet. Try, for instance, my favorite at <http://www.ietf.org/rfc.html>

## 1.20 Are there any known compression problems with specific browsers?

### *1.20.1 Yes. Netscape 4 has problems with compressed cascading style sheets and JavaScript files.*

You can use `Apache::CompressClientFixup` to disable compression for these files dynamically on Apache-1.3. `Apache::Dynagzip` is capable of providing so-called light compression for these files.

On Apache-2, `mod_deflate` can be configured to disable compression for these files dynamically, and the `Apache::Clean` filter can be used to strip unnecessary blank spaces.

## 1.21 Where can I find more information about the compression features of modern browsers?

### *1.21.1 Michael Schroepf maintains a highly valuable site*

See it at [http://www.schroepf.net/projekte/mod\\_gzip/browser.htm](http://www.schroepf.net/projekte/mod_gzip/browser.htm)

## 1.22 Acknowledgments

During this work, I received a great deal of real help from Kevin Kiley, Igor Sysoev, Michel Schroepf, and Henrik Nordstrom. I'm thankful to all subscribers of `mod_perl` users mailing list, `mod_gzip` mailing list, and `squid` users mailing list for the questions and discussions regarding the content compression. I'm especially thankful to Stas Bekman for the initiative to publish this FAQ on `mod_perl` Web site. I highly value patient efforts of Dan Hansen in making this text better English...

## 1.23 Maintainers

The maintainer is the person you should contact with updates, corrections and patches.

- Slava Bizyayev <slava (at) cpan.org>

## 1.24 Authors

- Slava Bizyayev <slava (at) cpan.org>

Only the major authors are listed above. For contributors see the Changes file.

## Table of Contents:

1	Web Content Compression FAQ . . . . .	1
1.1	Description . . . . .	2
1.2	Basics of Content Compression . . . . .	2
1.3	Why it is important to compress Web content? . . . . .	3
1.3.1	Reduced equipment costs and the competitive advantage of dramatically faster page loads. . . . .	3
1.4	How much improvement can I expect? . . . . .	3
1.4.1	Effective compression can achieve increases in transmission efficiency from 3 to 20 times. . . . .	3
1.5	How hard is it to implement content compression on an existing site? . . . . .	4
1.5.1	Implementing content compression on an existing site typically involves no more than installing and configuring an appropriate Apache handler on the Web server. . . . .	4
1.6	Does compression work with standard Web browsers? . . . . .	4
1.6.1	Yes. No client side changes or settings are required. . . . .	4
1.7	Is it possible to combine the content compression with data encryption? . . . . .	4
1.7.1	Yes. Compressed content can be encrypted and securely transmitted over SSL. . . . .	4
1.8	What software is required on the server side for content compression? . . . . .	4
1.8.1	There are four known mod_perl modules/packages for Web content compression available to date for Apache 1.3 (in alphabetical order): . . . . .	5
1.9	What is the typical overhead in terms of CPU use for the content compression? . . . . .	6
1.9.1	Typical CPU overhead that originates from content compression is insignificant. . . . .	6
1.10	Is it possible to compress the output from Apache::Registry with Apache::Dynagzip? . . . . .	6
1.10.1	Yes. This should be fairly easy to accomplish, as follows: . . . . .	6
1.11	Is it possible to compress output from a Mason-driven application with Apache::Dynagzip? . . . . .	8
1.11.1	Yes. HTML::Mason::ApacheHandler is compatible with the Apache::Filter chain. . . . .	8
1.12	Is commercial support available for Apache::Dynagzip? . . . . .	9
1.12.1	Yes. Slav Company, Ltd. provides commercial support for Apache::Dynagzip worldwide. . . . .	9
1.13	Why is it important to maintain a control over the chunk size? . . . . .	9
1.13.1	It helps to reduce response latency. . . . .	9
1.14	Is it worthwhile to strip leading blank spaces prior to gzip compression? . . . . .	9
1.14.1	Yes. It is usually worthwhile to do this. . . . .	9
1.15	Are there any content compression solutions for vanilla Apache 1.3? . . . . .	10
1.15.1	Yes. There are two compression modules written in C that are available for vanilla Apache 1.3: . . . . .	10
1.16	Can I compress the output of my site at the application level? . . . . .	10
1.16.1	Yes, if your Web server is CGI/1.1 compatible and allows you to create specific HTTP headers from your application, or when you use an application framework that carries its own handler capable of compressing outbound data. . . . .	10
1.17	Are there any content compression solutions for Apache-2? . . . . .	11
1.17.1	Yes. A core compression module written in C, mod_deflate, is available for Apache-2. . . . .	11
1.18	When is Apache::Dynagzip supposed to be ported to Apache-2? . . . . .	11

Table of Contents:

1.18.1	There are no current plans to port Apache::Dynagzip to Apache-2:	11
1.19	Where can I read the original descriptions of the gzip and deflate formats?	11
1.19.1	gzip format is published as rfc1952, and deflate format is published as rfc1951.	11
1.20	Are there any known compression problems with specific browsers?	12
1.20.1	Yes. Netscape 4 has problems with compressed cascading style sheets and JavaScript files.	12
1.21	Where can I find more information about the compression features of modern browsers?	12
1.21.1	Michael Schroepf maintains a highly valuable site	12
1.22	Acknowledgments	12
1.23	Maintainers	12
1.24	Authors	12