# Filters

Available Languages: en | es | fr | ja | ko | tr

This document describes the use of filters in Apache.

## Filtering in Apache 2

| Related Modules | Related Directives |
|---|---|
| - `mod_filter`<br>- `mod_deflate`<br>- `mod_ext_filter`<br>- `mod_include`<br>- `mod_charset_lite` | - `FilterChain`<br>- `FilterDeclare`<br>- `FilterProtocol`<br>- `FilterProvider`<br>- `AddInputFilter`<br>- `AddOutputFilter`<br>- `RemoveInputFilter`<br>- `RemoveOutputFilter`<br>- `ExtFilterDefine`<br>- `ExtFilterOptions`<br>- `SetInputFilter`<br>- `SetOutputFilter` |

The Filter Chain is available in Apache 2.0 and higher, and enables applications to process incoming and outgoing data in a highly flexible and configurable manner, regardless of where the data comes from. We can pre-process incoming data, and post-process outgoing data, at will. This is basically independent of the traditional request processing phases.

Some examples of filtering in the standard Apache distribution are:

- `mod_include`, implements server-side includes.
- `mod_ssl`, implements SSL encryption (https).
- `mod_deflate`, implements compression/decompression on the fly.
- `mod_charset_lite`, transcodes between different character sets.
- `mod_ext_filter`, runs an external program as a filter.

Apache also uses a number of filters internally to perform functions like chunking and byte-range handling.

A wider range of applications are implemented by third-party filter modules available from modules.apache.org and elsewhere. A few of these are:

- HTML and XML processing and rewriting
- XSLT transforms and XIncludes
- XML Namespace support
- File Upload handling and decoding of HTML Forms
- Image processing
- Protection of vulnerable applications such as PHP scripts
- Text search-and-replace editing

## Smart Filtering

`mod_filter`, included in Apache 2.1 and later, enables the filter chain to be configured dynamically at run time. So for example you can set up a proxy to rewrite HTML with an HTML filter and JPEG images with a completely separate filter, despite the proxy having no prior information about what the origin server will send. This works by using a filter harness, that dispatches to different providers according to the actual contents at runtime. Any filter may be either inserted directly in the chain and run unconditionally, or used as a provider and inserted dynamically. For example,

- an HTML processing filter will only run if the content is text/html or application/xhtml+xml
- A compression filter will only run if the input is a compressible type and not already compressed
- A charset conversion filter will be inserted if a text document is not already in the desired charset

## Using Filters

There are two ways to use filtering: Simple and Dynamic. In general, you should use one or the other; mixing them can have unexpected consequences (although simple Input filtering can be mixed freely with either simple or dynamic Output filtering).

The Simple Way is the only way to configure input filters, and is sufficient for output filters where you need a static filter chain. Relevant directives are `SetInputFilter`, `SetOutputFilter`, `AddInputFilter`, `AddOutputFilter`, `RemoveInputFilter`, and `RemoveOutputFilter`.

The Dynamic Way enables both static and flexible, dynamic configuration of output filters, as discussed in the `mod_filter` page. Relevant directives are `FilterChain`, `FilterDeclare`, and `FilterProvider`.

One further directive `AddOutputFilterByType` is still supported, but may be problematic and is now deprecated. Use dynamic configuration instead.

Available Languages: en | es | fr | ja | ko | tr