

Performance/Caching



- [tl;dr](#)
 - [Cache Everything](#)
 - [Cache the Entire Twig File and Data](#)
 - [Timber cache modes](#)
 - [Set Timber cache mode globally](#)
 - [Set Timber cache mode per compile or render method](#)
 - [Cache the Twig File \(but not the data\)](#)
 - [Cache Parts of the Twig File and Data](#)
 - [Extra: TimberKeyGeneratorInterface](#)
 - [Cache the PHP data](#)
 - [Measuring Performance](#)
 - [Important notes](#)
-

Timber, especially in conjunction with WordPress and Twig, offers a variety of caching strategies to optimize performance. Here's a quick rundown of some of the options, ranked in order of most-broad to most-focused.

tl;dr

In my tests with Debug Bar, Timber has no measurable performance hit. Everything compiles to PHP. @fabpot has an [overview of the performance costs on his blog](#) (scroll down to the table).

Cache Everything

You can still use plugins like [W3 Total Cache](#) in conjunction with Timber. In most settings, this will *skip* the Twig/Timber layer of your files and serve static pages via whatever mechanism the plugin or settings dictate.

Cache the Entire Twig File and Data

With Timber you can cache the full Timber render/compile calls. When you do this, the whole template you render and its data will be cached. This results in faster page rendering by skipping queries and Twig compilations. But here's the cool part: Timber hashes the fields in the view context. This means that **as soon as the data changes, the cache is automatically invalidated**. Yay!

For Timber caching to take effect on your `Timber::render()` and `Timber::compile()` calls, you need to set the `$expires` argument. If the `$expires` argument is not set, Timber will not cache that particular template, even if the (global)cache mode is set.

Example:

```
$context['posts'] = Timber::get_posts();

Timber::render('index.twig', $context, 600);
```

In this example, Timber will cache the template for 10 minutes ($600 / 60 = 10$) with the default cache mode which is "transient". You can change the cache mode for Timber globally or on a per method basis. See [Timber cache modes](#) for more information.

This caching method is very effective, but crude - the whole template is cached. So if you have any context dependent sub-views (eg. current user), this mode won't do.

Timber cache modes

Timber has 5 cache modes that it can use for the Timber `Timber::render()` and `Timber::compile()` methods. The following cache modes are available:

Mode	Description
<code>Timber\Loader::CACHE_NONE</code>	Disable caching
<code>Timber\Loader::CACHE_OBJECT</code>	WP Object Cache
<code>Timber\Loader::CACHE_TRANSIENT</code>	Transients
<code>Timber\Loader::CACHE_SITE_TRANSIENT</code>	Network wide transients
<code>Timber\Loader::CACHE_USE_DEFAULT</code>	Use whatever caching mechanism is set as the default for <code>Timber\Loader</code> , the default is <code>CACHE_TRANSIENT</code> .

By default the cache mode is set to transients. You can change the default cache mode globally by using a filter or on a per method basis. We will go over them both.

Set Timber cache mode globally

The default cache mode can be changed by using the `timber/cache/mode` filter. For example:

```
add_filter('timber/cache/mode', function () {  
    return Timber\Loader::CACHE_OBJECT;  
});
```

Sets the global/default cache mode to `CACHE_OBJECT`.

Set Timber cache mode per compile or render method

As a fourth parameter for [Timber::render\(\)](#) and [Timber::compile\(\)](#), you can set the `$cache_mode`.

For example:

```
Timber::render($filenames, $data, 600, Timber\Loader::CACHE_OBJECT);
```

Cache the Twig File (but not the data)

Every time you render a `.twig` file, Twig compiles all the HTML tags and variables into a big, nasty blob of function calls and echo statements that actually gets run by PHP. In general, this is pretty efficient. However, you can cache the resulting PHP blob by turning on Twig's cache via:

functions.php

```
add_filter('timber/twig/environment/options', function ($options) {  
    $options['cache'] = true;  
  
    return $options;  
});
```

You can look in your `/vendor/timber/timber/cache` directory to see what these files look like.

If you want to change the path where Timber caches the Twig files, you can pass in an absolute path for the `cache` option:

```
add_filter('timber/twig/environment/options', function ($options) {  
    $options['cache'] = '/absolute/path/to/twig_cache';  
  
    return $options;  
});
```

This does not cache the *contents* of the variables. But rather, the structure of the Twig files themselves (i.e. the HTML and where those variables appear in your template). Once enabled, any change you make to a `.twig` file (just tweaking the HTML for example) will not go live until the cache is flushed.

Thus, during development, you should enable the option for `auto_reload`:

```
add_filter('timber/twig/environment/options', function ($options) {  
    $options['cache'] = true;  
    $options['auto_reload'] = true;  
  
    return $options;  
});
```

Enabling `Timber::$cache` works best as a last step in the production process. Once enabled, any change you make to a `.twig` file (just tweaking the HTML for example) will not go live until the cache is flushed.

Note that when `WP_DEBUG` is set to `true`, changes you make to `.twig` files will be reflected on the site regardless of the `Timber::$cache` value.

To flush the Twig cache you can do this:

```
$loader = new Timber\Loader();  
$loader->clear_cache_twig();
```

Cache *Parts* of the Twig File and Data

If you want to use `cache tag` in Twig, you'll have to install the `twig/cache-extra` package:

```
composer require twig/cache-extra
```

And register it in Timber like this:

functions.php

```
use Symfony\Component\Cache\Adapter\FilesystemAdapter;
use Symfony\Component\Cache\Adapter\TagAwareAdapter;
use Twig\Extra\Cache\CacheExtension;
use Twig\Extra\Cache\CacheRuntime;
use Twig\RuntimeLoader\RuntimeLoaderInterface;

add_filter('timber/twig', function ($twig) {
    $twig->addRuntimeLoader(new class implements RuntimeLoaderInterface
    {
        public function load($class)
        {
            if (CacheRuntime::class === $class) {
                return new CacheRuntime(new TagAwareAdapter(new FilesystemAdapter('', 0, TIMBER_LOC .
                '/cache/twig')));
            }
        }
    });
    $twig->addExtension(new CacheExtension());

    return $twig;
});
```

You can then use it like this:

```
{% cache 'index;content' %}
    {% for post in posts %}
        {% include ['tease-' ~ post.post_type ~ '.twig', 'tease.twig'] %}
    {% endfor %}
{% endcache %}
```

Read more about it in [Twig's `cache` documentation](#).

If you want to use something meaningful for the cache key, you can also generate a cache key from the variables that you use. In the following example, we generate a cache key from `$posts` and then generate a key from it:

```
$generator = new Timber\Cache\KeyGenerator();  
$key = $generator->generateKey($posts);
```

Extra: TimberKeyGeneratorInterface

Instead of hashing a whole object, you can specify the cache key in the object itself. If the object implements `TimberKeyGeneratorInterface`, it can pass a unique key through the method `get_cache_key()`. That way a class could for example pass a `'last_updated'` property as the unique key. If arrays contain the key `_cache_key`, that one is used as cache key.

This may save yet another few processor cycles.

Cache the PHP data

Sometimes the most expensive parts of the operations are generating the data needed to populate the twig template. You can of course use WordPress's default [Transient API](#) to store this data.

You can also use some [syntactic sugar](#) to make the checking/saving/retrieving of transient data a bit easier:

home.php

```
$context = Timber::context();  
  
$context['main_stories'] = Timber\Helper::transient('main_stories', function () {  
    $posts = Timber::get_posts();  
  
    // As an example, do something expensive with these posts  
    $extra_teases = get_field('my_extra_teases', 'options');  
  
    foreach ($extra_teases as &$tease) {  
        $tease = Timber::get_post($tease);  
    }  
  
    $main_stories = [  
        'posts' => $posts,  
        'extra_teases' => $extra_teases,  
    ];
```

```
        return $main_stories;
    }, 600);

    Timber::render('home.twig', $context);
```

Here `main_stories` is a totally made-up variable. It could be called `foo`, `bar`, `elephant`, etc.

Measuring Performance

Some tools like Debug Bar may not properly measure performance because its data (as in, the actual HTML it's generating to tell you the timing, number of queries, etc.) is swept-up by the page's cache.

Timber provides some quick shortcuts to measure page timing. Here's an example of them in action:

single.php

```
// This generates a starting time
$start = Timber\Helper::start_timer();

$context = Timber::context([
    'whatever' => get_my_foo(),
]);

Timber::render('single.twig', $context, 600);

// This reports the time diff by passing the $start time
echo Timber\Helper::stop_timer($start);
```

Important notes

- Never use `{% apply spaceless %}` tags to minify your HTML output. These tags are only meant to control whitespace between HTML tags.

📄 #2

📄 📄 📄 2026-08-30 00:14:01 UTC

📄 📄 📄 2026-08-30 00:14:01 UTC