Context

- The Timber::context() function
  - Setting variables in the context
- Global context
  - Global context variables
  - Hook into the global context
  - Context cache
- Template contexts
  - Singular templates
    - Getting the current post
    - Using a custom post class
  - Archive templates
    - Use the default query
    - Write your own query
    - Change arguments for default query
    - Use a custom post class

The context is one of the most important concepts to understand in Timber. Think of the context as the set of variables that are passed to your Twig template.

In the following example, `$data` is an associative array of values. Each of the values will be available in the Twig template with the key as a variable name.

single.php

```
$data = [
    'message' => 'This can be any variable you want',
    'author' => 'Tom',
];

Timber::render('single.twig', $data);
```

single.twig

```
<h3>Message by {{ author }}</h3>
<p>{{ message }}</p>
```

The `Timber::context()` function

You don’t have to figure out all the variables you need in a template for yourself. Timber will provide you with a set of useful variables when you call `Timber::context()`.

single.php

```
$context = Timber::context();


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

Follow this guide to get an overview of what’s in the context, or use `var_dump( $context );` in PHP or `{{ dump() }}` in Twig to display the contents of the context in your browser.

Setting variables in the context

After you’ve called `Timber::context()`, you can add additional variables or overwrite existing variables in your context.

```
$context = Timber::context();


$context['today'] = wp_date('Ymd');


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

Another way to do this is to pass your custom data to the `Timber::context()` function itself:

```
$context = Timber::context([
    'today' => wp_date('Ymd'),
]);


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

Global context

The global context is the context that is always set when you load it through `Timber::context()`.

Global context variables

Among others, the following variables will be available:

- **site** – The `site` variable is a `Timber\Site` object which will make it easier for you to retrieve info about your WordPress site. If you're used to using `blog_info( 'sitename' )` in PHP, you can use `{{ site.name }}` in Twig instead.
- **theme** - The `theme` variable is a `Timber\Theme` object and contains info about your theme.
- **user** - The `user` variable will be a `Timber\User` object if a user/visitor is currently logged in and otherwise it will be `null`.

For a full list of variables, go have a look at the reference for `Timber::context()`.

Hook into the global context

In your theme, you probably have elements that you use on every page, like a navigation menu, or a postal address or phone number. You don't want to add these variables to the context every time you call `Timber::render()` or `Timber::compile()`. And you don't have to! You can use the `timber/context` filter to add your own data that will always be available.

Here's an example for how you could **add a navigation menu** to your context so that it becomes available in every template you use:

functions.php

```
// Example: Add a menu to the global context.
add_filter('timber/context', function ($context) {
    $context['menu'] = Timber::get_menu('primary-menu');

    return $context;
});
```

For menus to work, you will first need to register them.

Context cache

The global context will be cached. That's why you need to define your `timber/context` filter before using `Timber::context()` for the first time. Otherwise, the cache will be set before you could add your own data.

Having a cached global context can be useful if you need the context in other places. For example if you compile the template for a shortcode:

```
/**
 * Shortcode for address inside a WYSIWG field.
 */
add_shortcode('company_address', function () {
    return Timber::compile(
        'shortcode/company-address.twig',
        Timber::context_global()
    );
});
```

In this example, we've provided all the global context variables to **shortcode/company-address.twig** via `Timber::context_global()`. Whenever you only need the global context, you should use the `Timber::context_global()` function. You can call that function multiple times without losing performance.

Timber will not cache template contexts.

Template contexts

When WordPress decides which PHP template file it will display, it has already run database queries to fetch posts for archive templates or to set up the `$post` global for singular templates.

When you call `Timber::context()`, Timber will automatically populate your context with different variables like `post`, `posts`, `term`, `terms` or `author`, depending on which type of template file you're in.

Singular templates

The `post` variable will be available in singular templates (when `is_singular()` returns `true`), like posts or pages. It will contain a `Timber\Post` object of the currently displayed post.

single.php

```
$context = Timber::context();

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

By calling `Timber::get_post()` without any arguments, Timber will use the `$post` global for the current singular template.

Getting the current post

Now, Timber has already fetched the current post for your in `Timber::context()`. Here's how you could access it:

```
$context = Timber::context();

$post = $context['post'];

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

Using a custom post class

If you want to use your own post class, you can use the Post Class Map to register that class for your post type. `Timber::context()` will then automatically set the `post` variable using your class.

If you want to overwrite the existing `post` variable in the context, you can do that.

```
// Getting another post.
$post = Timber::get_post(12);
$post->setup();

// Get context with your post.
$context = Timber::context([
    'post' => $post,
]);
```

Or even shorter:

```
// Getting another post.
$post = Timber::get_post(12);

$context = Timber::context([
    'post' => $post->setup(),
```

```
]);
```

Be aware! Whenever you set up **a post in a singular template** (instead of relying on `Timber::context()` to do it for you), **you need to set up your post through** `$post->setup()`. The `setup()` function improves compatibility with third-party plugins.

Archive templates

The `posts` variable will be available in archive templates (when `is_archive()` returns `true`). In addition to that, it will also contain `post` or `term` variables for different archive types. Here's a small overview.

Archive	Condition	Context variables
Home	`is_home()`	`post` `posts`

Taxonomy Archive	`is_category()`
	`is_tag()`
	`is_tax()`
		`term`
		`posts`

Author Archive	`is_author()`	`author` `posts`
Search Archive	`is_search()`	`posts` `search_query`
All other archives	`is_archive()`	`posts`

The `posts` variable will contain an object that implements `Timber\PostCollectionInterace` with the posts that WordPress already fetched for your archive page.

Use the default query

archive.php

```
$context = Timber::context();


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

Write your own query

When you don't need the default query, you can pass in your own arguments to `Timber::get_posts()`.

archive.php

```
$context = Timber::context([
    'posts' => Timber::get_posts([
        'post_type' => 'book',
        'posts_per_page' => -1,
        'post_status' => 'publish',
    ]),
]);
```

Change arguments for default query

Sometimes you don’t want to use the default query, but build on the default query and only change a little thing. You can change arguments for the default query that WordPress will use to fetch posts by using the `merge_default` argument. For example, if you’d want to change the default query to *only show posts written by a specific group of authors*, you could pass in an `author__in` argument:

archive.php

```
$context = Timber::context([
    'posts' => Timber::get_posts(
        [
            'author__in' => [1, 6, 14],
        ],
        [
            'merge_default' => true,
        ]
    ),
]);

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

Timber will accept the parameters that can be found in WordPress's WP_Query class.

Use a custom post class

By default, `Timber::get_posts()` will contain `Timber\Post` objects. If you want to control what class will be used for the posts, you can use the Post Class Map.

□□□ #2
□ □□□ □□□ 2026-08-30 00:14:01 UTC
□ □□□ □□□ 2026-08-30 00:14:01 UTC