Menus

- Getting menus
  - Options
  - Menus need to be registered
- Extending menus
- Pages Menu
- Setting up a menu globally
  - Set up all menus globally
- Displaying the menu items
- The current menu item
  - Getting the current menu item outside the loop
  - Getting a current menu ancestor
  - Getting the current top-level item
- Menu item targets
- Tips

In Timber, we handle menus a little differently than in WordPress. We'd say it's a little smoother and more versatile than using WordPress's `wp_nav_menu()` function. And you never again need to rely on a crazy Walker Function.

Getting menus

To get a menu object in Timber, you use `Timber::get_menu()`. This function is similar to `wp_get_nav_menu_object()`. You have different options for what you pass in as an argument.

You can pass the **slug of the menu** you want to use:

```
$menu = Timber::get_menu('primary-navigation');
```

Or the **ID** number of the menu:

```
$menu = Timber::get_menu(3);
```

Or the proper **name** from the admin:

```
$menu = Timber::get_menu('Primary Navigation');
```

Or the **slug of the registered location**:

```
$menu = Timber::get_menu('primary');
```

What you get in return is a `Timber\Menu` object that holds a collection of `Timber\MenuItem` objects. If no menu can be found with the argument you provided, the function will return `null`.

In earlier versions of Timber, it was possible to pass in nothing. We've removed that functionality because it led to confusing cases where a menu built from your pages was returned. If you still want to get a menu from your existing pages, use `Timber::get_pages_menu()`.

Options

Optionally, you can send additional options to `Timber::get_menu()` in the second parameter.

```
$menu = Timber::get_menu('primary', [
    'depth' => 2,
]);
```

Currently, only `depth` is supported (see `wp_nav_menu()` for reference).

- **depth** – *(int)* How many levels of the hierarchy are to be included. 0 means all levels will be included. Default 0.

Menus need to be registered

Be aware that you first need to register your menu locations with `register_nav_menus()` before you call `Timber::get_menu()`.

```
/**
 * Register Menus
 */
add_action('after_setup_theme', function () {
    register_nav_menus([
```

```
        'primary' => 'Primary Menu',
        'secondary' => 'Secondary Menu',
        'footer' => 'Footer Menu',
    ]);
});
```

Extending menus

If you need additional functionality that the `Timber\Menu` and `Timber\MenuItem` classes don't provide or if you want to have cleaner Twig templates, you can extend the `Timber\Menu` or `Timber\MenuItem` class with your own classes:

```
class MenuPrimary extends \Timber\Menu
{
}
```

```
class MenuItemPrimary extends \Timber\MenuItem
{
}
```

To initiate your new `MenuPrimary` menu that will hold `MenuItemPrimary` objects, you also use `Timber::get_menu()`.

```
$menu = Timber::get_menu('primary');
```

In the same way that you can't instantiate post objects directly, you **can't** instantiate `Timber\Menu` or `Timber\MenuItem` objects or an object that extends this class with a constructor. Timber will use the Menu Class Map and the MenuItem Class Map to sort out which class it should use.

Pages Menu

Timber includes a function to create menu from your pages, without having to register menus first. This is a quick way to create a menu if you have a small website with only a couple of pages.

```
$menu = Timber::get_pages_menu();
```

This function will return an instance of `Timber\PagesMenu`, which is not quite the same as `Timber\Menu`, but it contains the same `Timber\MenuItem` objects as you know them.

If you want to extend a pages menu, you would do it like this:

```
class ExtendedPagesMenu extends \Timber\PagesMenu
{
}
```

There's a special PagesMenu Class Map which you can use to make Timber use your custom class.

Setting up a menu globally

Most of the time, you need the menu on every page. To achieve that, you can add your menu to the global context through the `timber/context` filter. This will make the menu available as an object in the context. The first thing to do is to initialize your menu using `Timber::get_menu()`.

functions.php

```
add_filter('timber/context', 'add_to_context');

/**
 * Global Timber context.
 *
 * @param array $context Global context variables.
 */
function add_to_context($context)
{
    // So here you are adding data to Timber's context object, i.e...
    $context['foo'] = 'I am some other typical value set in your functions.php file, unrelated
to the menu';

    // Now, in similar fashion, you add a Timber Menu and send it along to the context.
    $context['menu'] = Timber::get_menu('primary');

    return $context;
}
```

Now, when you call `Timber::context()`, your menu will already be set in the context. You don't need to initialize the menu in all your template files.

index.php

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

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

Set up all menus globally

Here’s a small snippet that you can use to automatically set up all your registered menus in the context.

functions.php

```
add_filter('timber/context', 'add_to_context');

/**
 * Global Timber context.
 *
 * @param array $context Global context variables.
 */
function add_to_context($context)
{
    // Set all nav menus in context.
    foreach (array_keys(get_nav_menu_locations()) as $location) {
        $menu = Timber::get_menu($location);
        $context[$location] = $menu;
    }

    return $context;
}
```

Displaying the menu items

In your Twig file, you can loop over the menu items like normal arrays with `{{ menu.items }}`. You’re in complete control of the markup. You can use `item.children` to check for and loop over child menu items.

index.twig

```
<nav>
    <ul class="nav-main">
        {% for item in menu.items %}
            <li class="nav-main-item {{ item.classes|join(' ') }}">
                <a
                    class="nav-main-link" href="{{ item.link }}"
                    {{ item.is_target_blank ? 'target="_blank"' }}
                >{{ item.title }}</a>

                {% if item.children %}
                    <ul class="nav-drop">
                        {% for child in item.children %}
                            <li class="nav-drop-item">
                                <a
                                    href="{{ child.link }}"
                                    {{ item.is_target_blank ? 'target="_blank"' }}
                                >{{ child.title }}</a>
                            </li>
                        {% endfor %}
                    </ul>
                {% endif %}
            </li>
        {% endfor %}
    </ul>
</nav>
```

The current menu item

When you need to check whether a menu item is the current menu item, you can use the `current` property. Here’s an example to display child menu items only if it’s the menu item is the currently visited page:

Twig

```
{% if item.current and item.children %}
    <ul class="nav-child">
        {% for child in item.children %}
            <li class="nav-child-item">
                <a
```

```
                    class="nav-child-link"
                    href="{{ child.link }}"
                >{{ child.title }}</a>
            </li>
        {% endfor %}
    </ul>
{% endif %}
```

Other available properties are

- `current_item_parent` for direct parents of a menu item and
- `current_item_ancestor` for when you have deeper nesting.

Getting the current menu item outside the loop

Say you want to display sibling menu items of the current page, for example in a sidebar. You can use the `current_item()` helper to achieve this:

Twig

```
<div class="sidebar">
    <a href="{{ menu.current_item.link }}">
        {{ menu.current_item.title }}
    </a>
    <ul>
        {% for child in menu.current_item.children %}
            <li>
                <a href="{{ child.link }}">{{ child.title }}</a>
            </li>
        {% endfor %}
    </ul>
</div>
```

Getting a current menu ancestor

You can limit the traversal depth of the tree when looking for the current item by passing a `$depth` parameter to `current_item`. Going off the previous example, say you wanted the root node of your sidebar to be the *second* level of the main menu tree. In that case, you could specify a depth of 2:

Twig

```
<div class="sidebar secondary-nav">
    <a href="{{ menu.current_item(2).link }}">
        {{ menu.current_item(2).title }}
    </a>
    <ul class="third-level-nav-items">
        {% for child in menu.current_item(2).children %}
            <li>
                <a href="{{ child.link }}">{{ child.title }}</a>
            </li>
        {% endfor %}
    </ul>
</div>
```

Getting the current top-level item

For getting the top-level (that is, level-1) item corresponding to the current post, you can call `current_top_level_item()`. This method takes no arguments and is just an alias for `current_item(1)`.

Menu item targets

To get the target for a menu item, you can use `item.target`:

```
<a href="{{ item.link }}" target="{{ item.target }}">
```

In the menu edit screen, WordPress offers a checkbox for each menu item that lets the administrator decide whether to open a menu item in a new tab. The `item.target` function will return `_blank` if that checkbox is ticked, and `_self` if it’s not ticked.

You might not need a value `_self` for the target attribute, because `_self` is the default value, which opens a link in the same tab/window. If you want to add `target="_blank"` only if needed, then you can use the conditional function `item.is_target_blank` :

```
<a
    href="{{ item.link }}"
    {{ item.is_target_blank ? 'target="_blank"' }}
>{{ item.title }}</a>
```

What about **external links**? If your site is `example.org`, then `google.com/whatever` is an external link. Whether it makes sense to open links in new tabs is not the topic to discuss here. If you still decide that you want that, you can check whether an item link is external with `item.is_external`:

```
<a
    href="{{ item.link }}"
    {{ item.is_external ? 'target="_blank"' }}"
>{{ item.title }}</a>
```

You could also use it in combination with `item.is_target_blank`:

```
<a
    href="{{ item.link }}"
    {{ item.is_target_blank or item.is_external ? 'target="_blank"' }}
>{{ item.title }}</a>
```

Tips

- Add items dynamically

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