WooCommerce

- Point of entry - main WooCommerce PHP file
- Archives
- Single Product
- Tease Product

Point of entry - main WooCommerce PHP file

The first step to get your WooCommerce project integrated with Timber is declaring WooCommerce support in your theme's **functions.php** file like so:

```
function theme_add_woocommerce_support()
{
    add_theme_support('woocommerce');
}

add_action('after_setup_theme', 'theme_add_woocommerce_support');
```

For more information about how you can enable or disable features and change settings through theme support, refer to the WooCommerce Theme Developer Handbook.

Once that's done you can start integrating WooCommerce into your theme by creating a file named **woocommerce.php** in the root of your theme. That will establish the context and data to be passed to your Twig files:

```
if (!class_exists('Timber')) {
    echo 'Timber not activated. Make sure you activate the plugin in <a href="/wp-
admin/plugins.php#timber">/wp-admin/plugins.php</a>';
```

```
    return;
}

$context = Timber::context();
$context['sidebar'] = Timber::get_widgets('shop-sidebar');

if (is_singular('product')) {
    $context['post'] = Timber::get_post();
    $product = wc_get_product($context['post']->ID);
    $context['product'] = $product;

    // Get related products
    $related_limit = wc_get_loop_prop('columns');
    $related_ids = wc_get_related_products($context['post']->id, $related_limit);
    $context['related_products'] = Timber::get_posts($related_ids);

    // Restore the context and loop back to the main query loop.
    wp_reset_postdata();

    Timber::render('views/woo/single-product.twig', $context);
} else {
    $posts = Timber::get_posts();
    $context['products'] = $posts;

    if (is_product_category()) {
        $queried_object = get_queried_object();
        $term_id = $queried_object->term_id;
        $context['category'] = get_term($term_id, 'product_cat');
        $context['title'] = single_term_title('', false);
    }

    Timber::render('views/woo/archive.twig', $context);
}
```

You will now need the two Twig files loaded from **woocommerce.php**: **views/woo/single-product.twig** and **views/woo/archive.twig**.

Archives

Create a Twig file according to the location asked by the above file, in this example that would be **views/woo/archive.twig**:

```
{% extends 'base.twig' %}

{% block content %}

    {% do action('woocommerce_before_main_content') %}

    <div class="before-shop-loop">
        {% do action('woocommerce_before_shop_loop') %}
    </div>

    <div class="loop">
        {% for post in products %}
            {% include ["partials/tease-product.twig"] %}
        {% endfor %}
    </div>

    {% do action('woocommerce_after_shop_loop') %}
    {% do action('woocommerce_after_main_content') %}

{% endblock  %}
```

You'll notice the inclusion of several of WooCommerce's default hooks, which you'll need to keep the integration seamless and allow any WooCommerce extension plugin to still work.

Next, we'll take care of the single product view.

Single Product

Create a Twig file according to the location asked by the above file, in this example that would be **views/woo/single-product.twig**:

```
{% extends "base.twig" %}

{% block content %}

    <div class="product">
        {% do action('woocommerce_before_single_product') %}
```

```
        <article itemscope itemtype="https://schema.org/Product" class="single-product-details
{{ post.class }}">

            <div class="entry-images">
                {% do action('woocommerce_before_single_product_summary') %}

                <img src="{{ post.thumbnail.src('shop_single') }}" />
            </div>

            <div class="summary entry-summary">
                {% do action('woocommerce_single_product_summary') %}
            </div>

            {% do action('woocommerce_after_single_product_summary') %}

            <meta itemprop="url" content="{{ post.link }}" />

        </article>

        {% for post in related_products %}
            {% include ["partials/tease-product.twig"] %}
        {% endfor %}

        {% do action('woocommerce_after_single_product') %}
    </div>

{% endblock  %}
```

Again we are keeping things simple by using WooCommerce's default hooks. If you need to override the output of any of those hooks, my advice would be to remove and add the relevant actions using PHP, keeping your upgrade path simple.

If you wanna use the same **tease-product.twig** output as your related products, you have to remove the default related-products from theme, add the following to your **functions.php** file:
`remove_action( 'woocommerce_after_single_product_summary', 'woocommerce_output_related_products', 20 );`

Finally, we'll need to create a teaser file for products in loops. Considering the code above that would be **views/partials/tease-product.twig**:

Tease Product

```
<article {{ fn('post_class', ['entry'] ) }}>

    {{ fn('timber_set_product', post) }}

    <div class="media">

        {% if showthumb %}
            <div class="media-figure {% if not post.thumbnail %}placeholder{% endif %}">
                <a href="{{ post.link }}">
                    {% if post.thumbnail %}
                        <img src="{{ post.thumbnail.src|resize(post_thumb_size[0],
post_thumb_size[1]) }}" />
                    {% else %}
                        <span class="thumb-placeholder"><i class="icon-camera"></i></span>
                    {% endif %}
                </a>
            </div>
        {% endif %}

        <div class="media-content">

            {% do action('woocommerce_before_shop_loop_item_title') %}

            {% if post.title %}
                <h3 class="entry-title"><a href="{{ post.link }}">{{ post.title }}</a></h3>
            {% else %}
                <h3 class="entry-title"><a href="{{ post.link }}">{{ fn('the_title')
}}</a></h3>
            {% endif %}

            {% do action( 'woocommerce_after_shop_loop_item_title' ) %}
            {% do action( 'woocommerce_after_shop_loop_item' ) %}

        </div>

    </div>
```

```
</article>
```

This should all sound familiar by now, except for one line:

```
{{ fn('timber_set_product', post) }}
```

For some reason, products in the loop don’t get the right context by default. This line will call the following function that you need to add somewhere in your **functions.php** file:

```
function timber_set_product($post)
{
    global $product;

    if (is_woocommerce()) {
        $product = wc_get_product($post->ID);
    }
}
```

Without this, some elements of the listed products would show the same information as the first product in the loop. If you see an error like `Warning: call_user_func_array() expects parameter 1 to be a valid callback, no array or string given`, this is your problem.

Note: Some users reported issues with the loop context even when using the `timber_set_product()` helper function. Turns out the default WooCommerce hooks interfere with the output of the aforementioned function.

One way to get around this is by building your own image calls, that means removing WooCommerce’s default hooks and declare your own template for the HTML to show the image:

```
{% if post.thumbnail %}
    <img src="{{ post.thumbnail.src|resize(shop_thumbnail_image_size) }}" />
{% endif %}
```

To remove the default image, add the following to your **functions.php** file:

```
remove_action('woocommerce_before_shop_loop_item_title',
'woocommerce_template_loop_product_thumbnail');
```

This comes with the added benefit that you’ll have total control over where your image is created in the markup.

□□□ #2
□ □□□ □□□ 2026-08-29 16:14:01 UTC
□ □□□ □□□ 2026-08-29 16:14:01 UTC