# Weight-Based Shipping Developer Reference

Developers · https://wpfactory.com/docs/weight-based-shipping-developers/

For developers, Weight-Based Shipping for WooCommerce registers the shipping method `weight_based_shipping`, stores each zone instance's settings in one WooCommerce option, and adds one filter, `woocommerce_shipping_weight_based_shipping_is_available`. Settings are edited per zone at **WPFactory > Weight-Based Shipping** (also reachable at **WooCommerce > Settings > Shipping**).

What the plugin registers, what it stores, and the hooks available for changing its behavior.

## Identifiers

These are stable across versions. Existing zone configurations depend on them, so they are not renamed even when the surrounding code is refactored.

| Thing | Value |
|---|---|
| Shipping method id | `weight_based_shipping` |
| Shipping method class | `WC_Weight_Based_Shipping_Method` (extends `WC_Shipping_Method`) |
| Main plugin class | `WC_Weight_Shipping_Plugin` |
| Global accessor | `wc_weight_shipping_plugin()` |
| Text domain | `wc-weight-shipping` |
| Plugin folder / main file | `woocommerce-weight-based-shipping/woocommerce-weight-based-shipping.php` |

Constants defined by the bootstrap:

| Constant | Value |
|---|---|
| `WC_WEIGHT_SHIPPING_VERSION` | The plugin version |

| Constant | Value |
| --- | --- |
| `WC_WEIGHT_SHIPPING_PLUGIN_DIR` | `plugin_dir_path()` of the main file |
| `WC_WEIGHT_SHIPPING_PLUGIN_URL` | `plugin_dir_url()` of the main file |
| `WC_WEIGHT_SHIPPING_PLUGIN_BASENAME` | `plugin_basename()` of the main file |

The method declares `shipping-zones`, `instance-settings` and `instance-settings-modal`, so it is zone-only and configured through the modal rather than a standalone settings page.

## Data storage

No custom tables and no custom post types. The shipping configuration lives entirely in WooCommerce's per-instance settings option:

```
woocommerce_weight_based_shipping_{instance_id}_settings
```

The `tiers` key inside it holds a plain indexed array, already sorted ascending by `min_weight`:

```
array(
    array( 'min_weight' => '0',  'max_weight' => '5', 'cost' => '10' ),
    array( 'min_weight' => '5',  'max_weight' => '10', 'cost' => '15' ),
    array( 'min_weight' => '10', 'max_weight' => '',   'cost' => '25' ),
)
```

All three values are strings passed through `wc_format_decimal()`. An empty `max_weight` means unlimited. Every other setting key is listed in the settings reference.

Reading a zone instance's tiers programmatically:

```
$settings = get_option( 'woocommerce_weight_based_shipping_' . $instance_id . '_settings', array()
);
$tiers    = isset( $settings['tiers'] ) ? $settings['tiers'] : array();
```

## Filters

**`woocommerce_shipping_weight_based_shipping_is_available`**

The one filter the plugin adds. It runs at the end of `is_available()`, after the enabled check and the maximum-weight check, so returning `true` will not resurrect a method that those have already rejected, but returning `false` can remove it for any reason of your own.

```
apply_filters( 'woocommerce_shipping_weight_based_shipping_is_available', true, $package, $this );
```

| Argument | Type | Description |
|---|---|---|
| `$available` | `bool` | Always `true` by the time the filter runs. |
| `$package` | `array` | The WooCommerce shipping package. |
| `$method` | `WC_Weight_Based_Shipping_Method` | The instance, for reading its settings. |

Hiding the method for logged-out customers:

```
add_filter(
    'woocommerce_shipping_weight_based_shipping_is_available',
    function ( $available, $package, $method ) {
        return is_user_logged_in() ? $available : false;
    },
    10,
    3
);
```

Hiding it in one particular zone instance:

```
add_filter(
    'woocommerce_shipping_weight_based_shipping_is_available',
    function ( $available, $package, $method ) {
        return ( 7 === (int) $method->instance_id ) ? false : $available;
    },
    10,
    3
);
```

### WooCommerce filters that also apply

The method returns an ordinary rate, so WooCommerce's own hooks work on it unchanged. The most useful is `woocommerce_package_rates`, which is the right place to adjust the cost after the fact:

```
add_filter( 'woocommerce_package_rates', function ( $rates, $package ) {
    foreach ( $rates as $rate ) {
        if ( 'weight_based_shipping' !== $rate->get_method_id() ) {
            continue;
        }
        // The weight the plugin calculated is on the rate.
        $weight = $rate->get_meta_data()['total_weight'] ?? 0;
        if ( $weight > 20 ) {
            $rate->set_cost( $rate->get_cost() + 5 );
        }
    }
    return $rates;
}, 10, 2 );
```

Rates are cached per package in the customer's session, so a filter like this only re-evaluates when the cart changes. Bump `WC_Cache_Helper::get_transient_version( 'shipping', true )` during development if you need to force it.

## Rate metadata

Every rate the method returns carries two pieces of metadata:

| Key | Value |
|---|---|
| `total_weight` | The package weight the rate was calculated from. |
| `weight_unit` | The store weight unit at the time of calculation. |

Because WooCommerce copies rate metadata onto the order's shipping line, the weight a customer was quoted on is recoverable from the order afterwards:

```
foreach ( $order->get_items( 'shipping' ) as $item ) {
    if ( 'weight_based_shipping' !== $item->get_method_id() ) {
        continue;
```

```
    }
    $weight = $item->get_meta( 'total_weight' );
    $unit   = $item->get_meta( 'weight_unit' );
}
```

This is the cleanest way to reconcile what was charged against what the warehouse actually weighed.

## How weight is collected

`calculate_package_weight()` walks `$package['contents']`, and for each line:

1. reads `$product->get_weight()`;
2. if that is empty and the product `is_type( 'variation' )`, loads the parent with `wc_get_product( $product->get_parent_id() )` and reads its weight;
3. multiplies by `$item['quantity']` and adds to the running total.

Product type is never checked beyond the variation fallback, so any product type that implements `get_weight()` is counted. There is no filter on the weight total itself. To adjust it, filter the cart contents or adjust the resulting rate through `woocommerce_package_rates`.

## The tier field format (and why it looks unusual)

The tier rows post as three parallel `[]` arrays rather than nested-array input names:

```
{field_key}_min[]
{field_key}_max[]
{field_key}_cost[]
```

where `{field_key}` is what `get_field_key( 'tiers' )` returns for the instance. `validate_weight_tiers_field()` reassembles them server-side by index.

This is deliberate. WooCommerce's shipping-zone settings modal serializes the form with `backbone-modal.js`'s `getFormData()`, which understands the simple `[]` suffix and then re-posts everything wrapped under a `data` key. Nested bracket names (`tiers[0][min]`) are mangled in that path and never reach the server, which was the cause of the tier-saving bug fixed in 1.3.2. If you extend the tier table, keep the flat naming.

The admin JavaScript only adds, removes and validates rows. Saving does not depend on it.

## What the plugin does not register

- No REST API routes.
- No shortcodes.
- No cron events or scheduled actions.
- No database tables and no activation/upgrade migration.
- No front-end scripts or styles. The only assets are the inline style and script inside the tier field, admin-side.
- No order-side hooks, which is why HPOS compatibility is a declaration rather than an implementation.

## Admin integrations

Admin-only, loaded behind `is_admin()` from `vendor/autoload.php`. The cross-selling and admin-menu integrations are each guarded with `class_exists()`; the key manager is hooked to `init` by function name.

| Library | What it adds |
|---|---|
| `wpfactory-key-manager` | Licensing and update checks. |
| `wpfactory-cross-selling` | The Recommendations page and the plugin-row action link. |
| `wpfactory-admin-menu` | The **WPFactory > Weight-Based Shipping** entry, registered as a submenu pointing at `admin.php?page=wc-settings&tab=shipping` because this plugin owns no WooCommerce settings tab of its own. |

The bundled `vendor/` folder is required in the admin: `vendor/autoload.php` is loaded with `require_once`, so do not remove it from the plugin package.

## Where to go next

- Settings reference
- How the shipping cost is calculated
- FAQ and troubleshooting



**WEIGHT-BASED SHIPPING FOR WOOCOMMERCE**

See every feature and the pricing on the Weight-Based Shipping for WooCommerce plugin page.

**View plugin →**

Need a hand? Our team answers every ticket on the Weight-Based Shipping support forum.

Related articles

- How to Offer Economy and Express Weight-Based Shipping Options
- How to Offer Free Shipping Over an Order Amount with Weight-Based Rates
- How to Set Different Weight-Based Shipping Rates per Zone or Country
- How to Set Up Weight Tiers in Weight-Based Shipping
- Install Weight-Based Shipping and Add It to a Zone