WooCommerce for Developers

Igor Benic

WooCommerce for Developers

Extend WooCommerce sites with code

Igor Benić

This book is for sale at http://leanpub.com/woodev

This version was published on 2018-05-28

Contents

About the Book

Hi there! Thank you for purchasing this book. Not only you have supported the work of a developer but you have invested in your career!

This book will teach you how to extend WooCommerce and understand how WooCommerce is functioning. You will learn how to develop for WooCoomerce using already defined actions and filters (hooks) that are defined by WooCommerce but also how to implement your own hooks within your plugins.

The whole book will present a specific project that will cover various specific topics. We could learn each topic separately but defining a specific project will require us to focus more on the details. By using this strategy you will be able to learn how to think and develop for specific purposes.

The specific topics that will be discussed and developed here are hard to find on the internet so be sure to study them good because they could come in handy from time to time.

Code in this book will be written in PHP and JavaScript mostly but some HTML and CSS could be seen in some parts.

All the code that you will see in this book can be used outside of it. I am giving you the permission to use them as you want (for non-commercial and commercial projects).

This book was not written by a member of WooCommerce or WooThemes team but only by me (Igor Benić) who is using WooCommerce in WordPress development.

How to read and use the code

Most of chapters (articles, tutorials) here are articles with code examples.

They are written in a way that you, as a reader, can easily understand every part of it. We are always starting from nothing and then by the end of the article you will have a usable code that you can easily edit to your own needs.

Code examples are written in separate boxes from text such as:

```
1  This is a line
```

but mostly the code examples will be consisted of much more lines such as:

```
<?php
This is a code example
$var = "variable in a code example";
```

Since a lot of these examples use the existing code and many of those code lines will be wider than this book's page, you will see sometimes a symbol '\' that indicates the next line to be the part of the line before.

When you see that symbol, ignore it (delete it without adding space) and continue to insert the code along the same line.

Here is an example of a long variable value that should be written in one line:

```
<?php
$long_variable = "This is a variable with a very long value to see how the same \
line is extended in more lines in this book.";
```

In the most articles, we will create a solution through the whole article so bare in mind that some of the code will come after a previous one.

Some code will not be a new code, but actually a redefined or refactored version of a previous code in the same article.

If you apply, copy or write the code in the article into another file, please be aware that only one opening *php* tag is needed. The opening tag will be used again, only if there was a need to close the previous one, for example: to write some regular HTML.

To show you what I mean let's use an example of this two code examples:

Code Example 1

```
<?php
$variableA = "Variable A";
```

Code Example 2

```
<?php
$variableB = "Variable B";
```

If we want to place them in one file, we would use only the first opening **php** tag like this:

```php
<?php
$variableA = "Variable A";
$variableB = "Variable B";
```

WooCommerce 3.x

I have decided to write only for up-to-date WooCommerce version. If you ever need to add a part to support old WooCommerce 2.6.x code or even older, than you can use this code to wrap the features:

```
if( version_compare( WC()->version, ‘3.0.0’, '<' ) ) {
     // backward compatibility here.
}
```

The WooDev Project

WooDev is just a name combined from the title of this book “WooCommerce for Developers”. WooDev will be the main project for which we will create different children projects such as:

- Shipping Method
- Restrictions for the shipping method
- General restriction on checkout
- Product Type
- Product Type connected to other WordPress content
- Product Type connected to an external service

Requirements

To follow this book you should have WordPress installed and WooCommerce activated. Be sure to code on your local machine or development environment since everything in here should be also tested on a staging server before using it on your production server.

The book will be refreshed when a refactoring is needed. You should have the latest versions of WordPress and WooCommerce installed.

Custom Shipping Method

Creating WooCommerce Shipping methods can be really fun. But to have fun you first need to know what your shipping method can do or can't do.

This is really important before entering any code. I want to be sure that you understand what WooCommerce Shipping Class can do so we will go through all the Shipping Class code.

The basic abstract class **WC_Shipping_Method** can be found in *woocommerce/includes/abstracts/abstract-wc-shipping-method.php*

Shipping API

Since you can extend or modify each class as you want, you can even modify some methods in the Shipping Class.

We will not go into modification of some previously defined methods in the abstract class, but if you would like to see how something was modified you can go into the folder *includes/shipping* inside **woocommerce** and look at each of those methods how they were defined.

There are still some methods that should be defined when creating your custom Shipping Method. Before we go into that, we should learn about the important attributes of our shipping class:

- **$id** - A required unique identifier for the shipping class
- **$method_title** - Title that will show in the admin area
- **$title** - Title that will be shown on the cart or checkout page
- **$method_description** - Used to describe our shipping method in the admin area
- **$availablity** - Indicator if the shipping method is available or not
- **$countries** - Array of countries that this shipping method is shipping to or not. Depending on *$availability*
- **$tax_status** - if set to *taxable*, the tax will be charged is possible
- **$minimum_fee** - a fee that will be charged when using this method when there are not other fees set,
- **$enabled** - indicator if this shipping method is enabled or not
- **$instance_id** - we can have more than one shipping method from the same class
- **$instance_form_fields** - fields that we use for settings
- **$instance_settings** - instance settings from database
- **$supports** - what this shipping method supports (settings, shipping-zones, instance-settings and/or instance-settings-modal)

What are those supports settings?

- **settings** - backward compatibility for old versions of WooCommerce
- **shipping-zones** - functionality for zones + instances
- **instance-settings** - instance settings used instead of settings
- **instance-settings-modal** - settings are opened in the modal and not on separate page

Now that you know which attributes are the most important ones and which you can easily set, we should also learn about some of the methods you should also set if needed:

- **__construct** - the constructor method will set the id and some other important attribues
- **init** - used to initialise the shipping fields and get the values for that fields
- **init_form_fields** - used to define all the form fields for our shipping method
- **calculate_shipping** - this, besides the constructor method, has to be set to register or shipping method cost or costs. Use the method **add_rate** to add the rates inside *calculate_shipping* method.

Countries and Availability

As described above, if there is an array of countries the shipping method can become available or not for those countries.

This is easily set by using the attribute **$availability**. We are setting that attribute when defining our own custom Shipping Method class.

This attribute can be left as it is without setting it in our own custom class or we can define it as follows:

- **specific** - Ships only to set countries
- **including** - Ships only to set countries
- **excluding** - Ships to all other countries than the set ones

The array of countries set with their ISO Codes. To find out about the ISO Codes of the countries you have to set use a site such as https://countrycode.org/[1].

[1] countrycode.org

Defining the Shipping Method

It is very important to define your shipping method. By defining your shipping method, I mean to define how our shipping method will calculate the cost, how much weight can it ship or which is the maximum dimension of our packages.

What about the countries and the availability of our shipping method? We will create a real world shipping method. I will use a shipping company in my own country **TISAK**.

TISAK operates in zones. Each country has their own zone and each zone has its own price. There are also weight and dimension limitations so we need to be aware of that also.

TISAK ships packages so each package is defined by their maximum dimensions and weight. We will not allow our customer to choose which package to use. We will instead automatically set which package to use based on the weight and dimension since that will be the package we will have to use when shipping with TISAK.

Since the manual for TISAK is written on Croatian I will not show it here. I can only say that TISAK does ship to around 123 countries + Croatia.

The cost of shipping is differently defined when shipping inside Croatia or when shipping to other countries. We will have them inside the countries array. There are also different price tags for each zone and each package.

Packages which TISAK uses are:

- Small
- Medium
- Large

When shipping inside Croatia prices for each package are:

- Small - 15kn (~ $2)
- Medium - 20kn (~ $3)
- Large - 25kn (~ $4)

When shipping to other countries prices for each package are also defined by zones:

- Zone 0
  - Small - 95kn (~ $15)
  - Medium - 105 (~ $16)
  - Large - 135kn (~ $21)
- Zone 1
  - Small - 220kn (~ $34)

    - Medium - 250kn (∼ $39)
    - Large - 275 (∼ $43)
- Zone 2
    - Small - 260kn (∼ $40)
    - Medium - 300kn (∼ $45)
    - Large - 360kn (∼ $55)
- Zone 3
    - Small - 470kn (∼ $72)
    - Medium - 550kn (∼ $85)
    - Large - 790kn (∼ $122)

Since we will automatically assign which package will be used when calculating the shipping cost we need to know the dimensions (in cm) of each package (length x width x height):

- Small - 20x20x15
- Medium - 30x20x20
- Large - 40x30x15

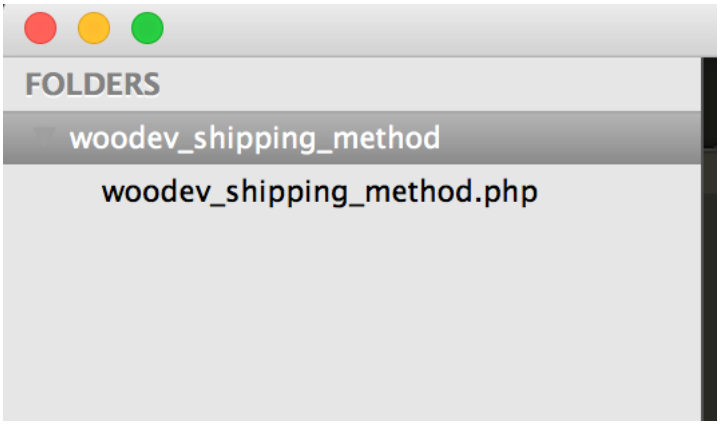
I have used the unit *cm* since that is how TISAK has defined their package dimensions. You can use a different one as long as it is allowed in WooCommerce.

The last part that we should define is the weight. TISAK does not define the cost of their shipping by weight but they do have a limit of maximum 10kg on any package. If our cart items surpass this limit in weight, then our shipping method will not be available.

Now that we have defined our shipping method, we can start coding! Prepare your developing environment and let's start coding.

Shipping Class

We will start first off with a simple WordPress plugin definition. Create your own plugin folder and give it a name to your liking. I will call it **woodev_shipping_method**. Inside that folder create a file with the same name and the *php* extension.



Folder and File for Custom Shipping Method

Let's enter the plugins' definition inside our newly created file:

```
<?php
/*
Plugin Name: WooCommerce For Developers Custom Shipping
Plugin URI: https://leanpub.com/woodev
Description: A Custom Shipping Method done in the "WooCommerce for Developers" e\
Book
Version: 1.0
Author: Igor BeniÄ‡
Author URI: http://ibenic.com
textdomain: woodev_shipping
*/
```

Go to your WordPress admin area inside the menu **Plugins** and activate your newly create plugin. By activating it, all our changes can be seen instantly on our website.

Make sure WooCommerce is active

The headline kind of tells you what is going to be done here, right? We will check if WooCommerce is active and then proceed with our functionality.

There is no point in adding our Shipping Method to WooCommerce if WooCommerce is not installed and active. So let's add this chunk of code next:

```
/**
 * Check if WooCommerce is active
 */
if ( in_array( 'woocommerce/woocommerce.php',
apply_filters( 'active_plugins', get_option( 'active_plugins' ) ) ) ) {

        // our code will go here because we are sure WooCommerce is active

}
```

Basically, we get all the active plugins and then check if WooCommerce is there by using *plugin_-folder/plugin_file.php* string that is saved for every activated plugin.

The Base Class & Shipping Settings

We will define a function that will hold the definition of our custom shipping method and hook it to a WooCommerce action **woocommerce_shipping_init** which is used to get all registered WooCommerce Shipping Methods.

```
function woodev_tisak_shipping() {
  if ( ! class_exists( 'WOODEV_TISAK_Shipping' ) ) {
    class WOODEV_TISAK_Shipping extends WC_Shipping_Method {

    }
  }
}
add_action( 'woocommerce_shipping_init', 'woodev_tisak_shipping' );
```

In our function *woodev_tisak_shipping* we have defined a new class **WOODEV_TISAK_Shipping** that extends the abtract shipping class.

Now we need to add the constructor method inside our new class:

```
// ...
class WOODEV_TISAK_Shipping extends WC_Shipping_Method {

/**
 * Constructor for your shipping class
 *
 * @access public
 * @return void
 */
 public function __construct() {
   $this->id        = 'woodev_tisak_shipping';
   $this->method_title        = __( 'TISAK Shipping', 'woodev_shipping' );
   $this->method_description        = __( 'TISAK Shipping Settings', 'woodev_shipping' \
);
   $this->title        = __("TISAK","woodev_shipping");
   $this->init();
   $this->enabled        = true;
 }
// ...
```

So here we have defined the unique **id** of our shipping method, title that will be displayed in the admin area, description that will be displayed in the admin area and the title that will be displayed on our cart or checkout pages.

Method *init* will be used to get all the settings and settings' fields for this shipping method. The last attribute we have defined is **enabled** which enables our shipping method to be used for shipping costs.

Let's define now our method *init* so that we can use our settings:

```
// ...
/**
 * Init your settings
 *
 * @access public
 * @return void
 */
function init() {
  $this->init_form_fields();
  $this->init_settings();
  // Save settings in admin if you have any defined
  add_action( 'woocommerce_update_options_shipping_' . $this->id,
    array( $this, 'process_admin_options' ) );
```

```
14  }
15  // ...
```

In this method we are calling another method *init_form_fields* that is used to set our shipping settings' fields. We are also calling the method *init_settings* to get all our settings that we can use when calculating the shipping cost.

Finally we are hooking the method *process_admin_options* so that all our fields values are saved to database in the admin area.

So, how to define our fields for this shipping method?

Shipping Settings' Fields

We will construct an array of field definitions and set it to our attribute **#form_fields**. Let's define our form fields:

```
1   // ...
2   /**
3    * Settings Fields
4    * @return void
5    */
6   function init_form_fields() {
7     $this->form_fields = array(
8      'enable' => array(
9         'title' => __( 'Enable', 'woodev_shipping' ),
10        'type' => 'checkbox',
11        'description' => __( 'Enable this shipping.', 'woodev_shipping' ),
12        'default' => 'no'
13        ),
14    );
15
16  }
17  // ...
```

We have set only one field. The name (id) of our field is **enable**. Everything else is self explanatory so I will not go into that.

Now that we have our field set, we can enable our shipping method within the admin area. To reflect that setting, we need to modify our contructor method so that our attribute **enabled** is getting set from our settings:

```
// ...
class WOODEV_TISAK_Shipping extends WC_Shipping_Method {

/**
 * Constructor for your shipping class
 *
 * @access public
 * @return void
 */
 public function __construct() {
   $this->id        = 'woodev_tisak_shipping';
   $this->method_title        = __( 'TISAK Shipping', 'woodev_shipping' );
   $this->method_description        = __( 'TISAK Shipping Settings', 'woodev_shipping' \
);
   $this->title        = __("TISAK","woodev_shipping");
   $this->init();
   $this->enabled        = $this->settings["enable"];
 }
// ...
```

OK, now we have defined our shipping settings and our base class. We still have to define our method to calculate the shipping cost and set all the available countries.

Let's go with the second requirement.

Shipping Countries

We will now create our own custom function that will return an associative array where keys will be the countries ISO codes and values will be the corresponding zones. Place this function below our class definition.

This is a long list:

```
/**
 * Array with all country codes where this shipping method is shipping
 * @return array country codes with zones
*/

function woodev_shipping_countries(){
    return array(
        'BA' => 0,
        'XK' => 0,
        'ME' => 0,
```

```
11          'RS' => 0,
12          'SI' => 0,
13          'AT' => 1,
14          'BE' => 1,
15          'BG' => 1,
16          'CZ' => 1,
17          'DK' => 1,
18          'FR' => 1,
19          'DE' => 1,
20          'HU' => 1,
21          'IT' => 1,
22          'NL' => 1,
23          'PL' => 1,
24          'RO' => 1,
25          'SK' => 1,
26          'GB' => 1,
27          'EE' => 2,
28          'FI' => 2,
29          'GR' => 2,
30          'IE' => 2,
31          'LV' => 2,
32          'LT' => 2,
33          'LU' => 2,
34          'PT' => 2,
35          'ES' => 2,
36          'SE' => 2,
37          'AF' => 3,
38          'AL' => 3,
39          'DZ' => 3,
40          'AS' => 3,
41          'AD' => 3,
42          'AO' => 3,
43          'AI' => 3,
44          'AG' => 3,
45          'AR' => 3,
46          'AM' => 3,
47          'AW' => 3,
48          'AU' => 3,
49          'AZ' => 3,
50          'BS' => 3,
51          'BH' => 3,
52          'BD' => 3,
```

```
53          'BB' => 3,
54          'BY' => 3,
55          'BZ' => 3,
56          'BJ' => 3,
57          'BM' => 3,
58          'BT' => 3,
59          'BO' => 3,
60          'BQ' => 3,
61          'BW' => 3,
62          'BR' => 3,
63          'VG' => 3,
64          'BN' => 3,
65          'BF' => 3,
66          'BI' => 3,
67          'KH' => 3,
68          'CM' => 3,
69          'IC' => 3,
70          'CV' => 3,
71          'KY' => 3,
72          'CF' => 3,
73          'TD' => 3,
74          'CL' => 3,
75          'CN' => 3,
76          'CO' => 3,
77          'KM' => 3,
78          'CG' => 3,
79          'CD' => 3,
80          'CK' => 3,
81          'CR' => 3,
82          'CI' => 3,
83          'CW' => 3,
84          'CY' => 3,
85          'DJ' => 3,
86          'DM' => 3,
87          'DO' => 3,
88          'TL' => 3,
89          'EC' => 3,
90          'EG' => 3,
91          'SV' => 3,
92          'GQ' => 3,
93          'ER' => 3,
94          'ET' => 3,
```

```
95          'FO' => 3,
96          'FJ' => 3,
97          'PF' => 3,
98          'GA' => 3,
99          'GM' => 3,
100         'GE' => 3,
101         'GH' => 3,
102         'GI' => 3,
103         'GL' => 3,
104         'GD' => 3,
105         'GP' => 3,
106         'GU' => 3,
107         'GT' => 3,
108         'GN' => 3,
109         'GW' => 3,
110         'GY' => 3,
111         'HT' => 3,
112         'HN' => 3,
113         'HK' => 3,
114         'IS' => 3,
115         'IN' => 3,
116         'ID' => 3,
117         'IQ' => 3,
118         'IL' => 3,
119         'JM' => 3,
120         'JP' => 3,
121         'JO' => 3,
122         'KZ' => 3,
123         'KE' => 3,
124         'KI' => 3,
125         'KR' => 3,
126         'FM' => 3,
127         'KW' => 3,
128         'KG' => 3,
129         'LA' => 3,
130         'LB' => 3,
131         'LS' => 3,
132         'LR' => 3,
133         'LY' => 3,
134         'LI' => 3,
135         'MO' => 3,
136         'MK' => 3,
```

```
137             'MG' => 3,
138             'MW' => 3,
139             'MY' => 3,
140             'MV' => 3,
141             'ML' => 3,
142             'MT' => 3,
143             'MH' => 3,
144             'MQ' => 3,
145             'MR' => 3,
146             'MU' => 3,
147             'YT' => 3,
148             'MX' => 3,
149             'MD' => 3,
150             'MC' => 3,
151             'MN' => 3,
152             'MS' => 3,
153             'MA' => 3,
154             'MZ' => 3,
155             'MM' => 3,
156             'NA' => 3,
157             'NP' => 3,
158             'KN' => 3,
159             'NC' => 3,
160             'NZ' => 3,
161             'NI' => 3,
162             'NE' => 3,
163             'NG' => 3,
164             'MP' => 3,
165             'NO' => 3,
166             'OM' => 3,
167             'PK' => 3,
168             'PW' => 3,
169             'PA' => 3,
170             'PG' => 3,
171             'PY' => 3,
172             'PE' => 3,
173             'PH' => 3,
174             'PR' => 3,
175             'QA' => 3,
176             'RE' => 3,
177             'RU' => 3,
178             'RW' => 3,
```

```
179             'WS' => 3,
180             'SM' => 3,
181             'SA' => 3,
182             'SN' => 3,
183             'SC' => 3,
184             'SL' => 3,
185             'SG' => 3,
186             'SB' => 3,
187             'ZA' => 3,
188             'LK' => 3,
189             'BL' => 3,
190             'LC' => 3,
191             'SX' => 3,
192             'MF' => 3,
193             'VC' => 3,
194             'SR' => 3,
195             'SZ' => 3,
196             'CH' => 3,
197             'TW' => 3,
198             'TJ' => 3,
199             'TZ' => 3,
200             'TH' => 3,
201             'TG' => 3,
202             'TO' => 3,
203             'TT' => 3,
204             'TN' => 3,
205             'TR' => 3,
206             'TM' => 3,
207             'TC' => 3,
208             'TV' => 3,
209             'UG' => 3,
210             'UA' => 3,
211             'AE' => 3,
212             'UY' => 3,
213             'UZ' => 3,
214             'VU' => 3,
215             'VE' => 3,
216             'VN' => 3,
217             'VG' => 3,
218             'WF' => 3,
219             'YE' => 3,
220             'ZM' => 3,
```

```
221         'ZW' => 3,
222         );
223 }
```

Now that we have our function for countries, we can assign the country codes to the attribute **$countries** and set the attribute **$availability** to *including*. We will do that in our method *init*:

```
1  function init() {
2          $this->init_form_fields();
3          $this->init_settings();
4          $country_codes = woodev_shipping_countries();
5          $this->countries = array_keys( $country_codes );
6          $this->countries[] = 'HR';
7          $this->availability = 'including';
8
9      // Save settings in admin if you have any defined
10         add_action( 'woocommerce_update_options_shipping_' . $this->id,
11                 array( $this, 'process_admin_options' ) );
12 }
```

First, we get the array of countries with their zones. After that, we get only the keys in that array which are actually only the country codes. That array of keys will be then set to the attribute **$countries**.

Availability is then set to *including* so that this shipping method will be available only for those countries we have set.

Getting the Right Package

Let's define a method that will be used to get the right package from our dimensions, maximum length, width or height. This method will return false if we could not get any package from provided parameters.

```php
/**
 * Get the package for TISAK
 * @param  number $dimension     Dimension volume (length x width x height)
 * @param  number $maximumLength Maximum length from all items in our cart
 * @param  number $maximumWidth  Maximum width from all items in our cart
 * @param  number $maximumHeight Maximum height from all items in our cart
 * @return mixed                 Returns false if there is no package
 *                                                                    that can be selected
 */
public function getTisakPackage(
        $dimension,
        $maximumLength,
        $maximumWidth,
        $maximumHeight ) {

                $packageS = 20 * 20 * 15;
                $packageM = 30 * 20 * 20;
                $packageL = 40 * 30 * 15;

                if( $maximumLength > 40 ){
                    return false;
                }

                if( $maximumWidth > 30 ){
                    return false;
                }

                if( $maximumHeight > 20 ) {
                    return false;
                }

                if( $dimension <= $packageS ) {

                  return 's';

                } elseif ( $dimension <= $packageM ) {

                  return 'm';

                } elseif ( $dimension <= $packageL ) {

                  return 'l';
```

```
43
44                  }
45
46                  return false;
47  }
```

In this method we are passing the mentioned parameters and define the maximum dimensions for each package (**$packageS**, **$packageM** and **$packageL**). If any maximum parameter is higher than the maximum possible (from any package) then we do not have a shippable package.

Getting the Price for Package

We have to create a method or methods to get the right price for the selected package based on the country where we need to ship.

Since each package has its own price for Croatia and a different one for every zone we will need to define two different methods for national and international shipping.

Shipping value for Croatia is easily calculated for each package since there are no zones. Here is the method for getting the price for each package when shipping in Croatia.

```
1   public function croatia_shipping_value( $package ){
2     switch ( $package ) {
3       case 's':
4         return 15;
5         break;
6       case 'm':
7         return 20;
8         break;
9       default:
10        return 25;
11        break;
12    }
13  }
```

By receiving the package we are returning the price for each package size.

To calculate the shipping price for international shipping we will have to develop helper methods that will be use to decouple our code into smaller parts and thus make it more maintainable and readable.

The first helper method will be for getting the price by zones:

```php
/**
* Returns the array with prices for a zone
* @param  number $zone zone number
* @return array       prices for packages
*/
public function get_zone_prices( $zone ) {

        $zonePrice = array(
            0 => array(
                's' => 95,
                'm' => 105,
                'l' => 135),
            1 => array(
                's' => 220,
                'm' => 250,
                'l' => 275),
            2 => array(
                's' => 260,
                'm' => 300,
                'l' => 360),
            3 => array(
                's' => 470,
                'm' => 550,
                'l' => 790),
            );
        return $zonePrice[ $zone ];
}
```

In this method we are receiving a zone as a number. We have defined an array with prices for each package for every zone. We are returning only one set of package prizes by using the zone number as the index of the array.

So now we have the method to retrieve all package prizes for a particular zone. We also have all countries with their zones assigned in the method **shipping_countries()**.

That are actually two helper methods that can be combined to get the right prize for the country and the package. Let's now create our method to calculate the shipping price when shipping internationally.

```php
/**
* Returns the prices for the provided country and package
* @param  string $country ISO country code
* @param  string $package
* @return number          Price for the package and country
*/
public function international_shipping_value( $country, $package ){
        $countries = woodev_shipping_countries();

        if( ! isset( $countries[ $country ] ) ) {
            return false;
        }

        $countryZone = (int) $countries[ $country ];

        $packagesFromZone = $this->get_zone_prices( $countryZone );

        $price = $packagesFromZone[ strtolower( $package ) ];

        return $price;
}
```

In this method we will pass two parameters: country code and package size. We will get all the countries from our method **shipping_countries()** and check if the passed country code is in that array of countries.

Afterwards, if the country is supported by our shipping method, we will get the zone that is assigned to that country by passing the country code to the array of countries. Since the country code is the **key** in the array and the zone is the **value** array, we will receive the **value** and that is the zone we require.

Once that is done, we are getting all the package prices for that specific zone using the method **get_zone_prices**. Once our prices with package sizes are returned for the passed zone, we are getting the specific prize for the package size we have passed to this method. The last thing to do is to return the price.

At this point we have our method to calculate the cost of the shipping for any country to which **TISAK** ships. We have our method to get the right package by dimensions and weight. We now need to combine all of those methods in the method **calculate_shipping** which WooCommerce uses to get the cost of that shipping.

Calculating the Shipping Cost

The first step is to define some variables which will be used to calculate the shipping cost.

```
/**
 * calculate_shipping function.
 *
 * @access public
 * @param mixed $package
 * @return void
 */
public function calculate_shipping( $package ) {

    $cost = 0;
    $weight = 0;
    $currency = get_woocommerce_currency();
    $maximumLength = 0;
    $maximumHeight = 0;
    $maximumWidth = 0;

    if( $currency != 'HRK' ){
      return false;
    }

    $dimensions = 0;

            ...
```

Variable **$weight** will be used to contain the total weight of the package.

We are getting the currency used in WooCommerce by using the function *get_woocommerce_currency()*. I am here checking if the currency is **Croatian Kuna** and I return false it is not. So if WooCoommerce uses another currency this shipping will never be displayed as available at the checkout page.

This does not have to be in your case. Or even in this case. Since all the prices here are calculated and set as they would be in **Croatian Kuna**, we could have a *flag* that is true if the currency is not **Croatian Kuna**. We will then use a conversion function or API to convert the total shipping cost into your currency.

If you are following this article by copying the code, I suggest you to not use that part where I return false if it is not HRK. Or you can set your WooCommerce currency to HRK to see the shipping method.

In this method a parameter **$package** is passed. This parameter holds all the items we have in our cart and all our shipping definitions on the checkout page. We need to see the dimensions of all those items in the package and calculate the dimensions.

```
...
foreach ( $package['contents'] as $item_id => $values )
{
  $_product = $values['data'];
  $weight = $weight + $_product->get_weight() * $values['quantity'];
  $width = floatval( wc_get_dimension( $_product->get_width(), 'cm' ) );
  if( $maximumWidth < $width ) {
      $maximumWidth = $width;
  }

  $height = floatval( wc_get_dimension( $_product->get_height(), 'cm' ) );
  if( $maximumHeight < $height ) {
      $maximumHeight = $height;
  }

  $length = floatval( wc_get_dimension( $_product->get_length(), 'cm' ) );
  if( $maximumLength < $length ) {
      $maximumLength = $length;
  }

  $dimensions = $dimensions + (( $length * $values['quantity']) * $width * $heig\
ht );
}
...
```

On the first line are getting the value the product object for each item. Then we are calculating the weight which we add to the total weight in our variable **$weight**.

After that we are getting the width of this product and we are converting it in the unit **cm** since all our methods and definitions are using **cm** for dimensions.

We are also assigning that width to the **$maximumWidth** if that width is greater then the current maximum width.

The same logic is used for both height and length. The last part of the foreach loop is calculating the dimensions. Dimension is calculated in Length * Height * Width. Since we also can have more than one of those items, we have to use the parameter quantity also.

When the dimension for that item is calculated we are adding it up to the total dimensions which is our variable **$dimensions**.

We have our total weight and our total dimensions. Let's see if we can ship our package with **TISAK**:

```
...
$weight = wc_get_weight( $weight, 'kg');

if( $weight > 10 ){
return false;
}

$tisak_package = $this->getTisakPackage( $dimensions, $maximumLength, $maximumWi\
dth, $maximumHeight );

if( $tisak_package == false ) {
   return false;
}
...
```

Since our maximum weight which can be shipped is 10 **kg** we have to set the weight to **kg** unit. Once that is done we check if the weight is under **10kg**. If not, we return false and this shipping method is not displayed.

We are also getting the package by passing the total dimensions variable, maximum length, width and height. If we do not receive a package value but we receive **false**. Then it means that the dimensions we have passed are exceeding all the packages.

Let's now define how to calculate the cost if everything until that point went well:

```
...
if( $package['destination']['country'] == 'HR' ) {

        $cost = $this->croatia_shipping_value( $tisak_package );

}else{

        $cost = $this->international_shipping_value( $package['destination']['co\
untry'], $tisak_package );
}

if( $cost == false ) {
        return false;
}
...
```

We check the destination for that package. If the country code is **HR** then it means that this package will be shipped inside Croatia. If that is the case, we are using our method **croatia_shipping_value** to get the cost.

If the package is going to be shipped internationally, then we will use our other method **international_shipping_value**.

Once that is done and if the variable **$cost** is false, then we return false and the shipping method will not be displayed.

The last part of this method is to add the rate:

```
...
$rate = array(
    'id' => $this->id,
    'label' => $this->title,
    'cost' => $cost,
    //'calc_tax' => 'per_item'
);

// Register the rate
$this->add_rate( $rate );
}
```

Adding our Shipping Class to all other Shipping Methods

To add the shipping method to other shipping method (register it), we need to use the filter *woocommerce_shipping_methods*. That filters passes an array of all registered Shipping method.

By adding our shipping id as a **key** inside that array while the value is the name of our Shipping class, we will register it.

```
/**
 * Adding our Shipping class to methods
 * @param  array $methods
 * @return array
 */
function woodev_add_shipping_methods( $methods ) {
        $methods['woodev_tisak_shipping'] = 'WOODEV_TISAK_Shipping';
        return $methods;
}
add_filter( 'woocommerce_shipping_methods', 'woodev_add_shipping_methods' );
```

That's it! You now have a shipping method that you can use in your own WooCommerce project. Let's learn how to add or remove shipping zones programmatically.