[TUTORIALS](https://greyhawkdigital.com/category/tutorials/)

**[WooCommerce: Third Party API Calls At Checkout](https://greyhawkdigital.com/woocommerce-third-party-api-calls-at-checkout/" \o "WooCommerce: Third Party API Calls At Checkout)**

I’ve been doing a lot more WooCommerce work as of late. And with good reason – WooCommerce (as of the end of 2016) powers nearly [42% of all eCommerce stores](https://wptavern.com/woocommerce-powers-42-of-all-online-stores) on the internet. One of the biggest requests I have when working with WooCommerce is how to integrate a third party vendor into a checkout screen. An example would be a gift card merchant. The retail store may not have the capability to fulfill their own gift cards, and so a third party merchant is used.  That’s well and good, but how do we get WooCommerce and the third party merchant to talk to each other?

As long as the third party vendor has even a half-way decent API, we can tie this into WooCommerce.

Prerequisites

The biggest prerequisite is that the third party vendor have an API that allows *at least* one way communication.  Preferredly, we’d like to have a payload AND response sent back, but sometimes that’s just not the case.  API documentation will differ on a per-vendor basis, but having an understanding of how the API responds and accepts payloads is a must here.

We are also working under a few assumptions:

* We want to only interface with the API upon completion of a payment.
* We want to store / display information from the API
* We want the API information to be sent out with WooCommerce’ email.

Sending / Receiving from the API

I’ve added a function below. We’ll walk through it once you’ve had a chance to read it.

add\_action( 'woocommerce\_payment\_complete', 'my\_api\_call');

function my\_api\_call( $order\_id ){

// Order Setup Via WooCommerce

$order = new WC\_Order( $order\_id );

// Iterate Through Items

$items = $order->get\_items();

foreach ( $items as $item ) {

// Store Product ID

$product\_id = $item['product\_id'];

$product = new WC\_Product($item['product\_id']);

// Check for "API" Category and Run

if ( has\_term( 'api', 'product\_cat', $product\_id ) ) {

$name = $order->billing\_first\_name;

$surname = $order->billing\_last\_name;

$email = $order->billing\_email;

$projectsku = $product->get\_sku();

$apikey = "KEY\_GOES\_HERE";

// API Callout to URL

$url = '##API URL##';

$body = array(

"Project" => $projectsku,

"Name" => $name,

"Surname" => $surname,

"Email" => $email,

"KEY" => $apikey

);

$response = wp\_remote\_post( $url,

array(

'headers' => array('Content-Type' => 'application/json; charset=utf-8'),

'method' => 'POST',

'timeout' => 75,

'body' => json\_encode($body),

)

);

$vars = json\_decode($response['body'],true);

// API Response Stored as Post Meta

update\_post\_meta( $order\_id, 'meta\_message\_'.$projectsku, $vars['message'] );

update\_post\_meta( $order\_id, 'meta\_link\_'.$projectsku, $vars['link']);

update\_post\_meta( $order\_id, 'did-this-run','yes'); // just there as a checker variable for me

}

}

}

The first thing we do is to create an action for the woocommerce\_payment\_complete hook. This hook will only fire once the payment from whatever payment method the customer chooses comes back as successful.  Any pending or “pending payment” orders will not fire this action.

We want to then create a new $order via the WC\_Order class, and loop through the items.

For this particular code, I didn’t want -every- item to fire the API since there are other types of products in their store. So, we then check for any products in the ‘API’ category – via the slug – and only run the script further when that condition is met.

If the condition is met, we start to build the payload – the information we want to send to our third-party. In this case, we’re sending first name, last name, email, SKU (which I used as a unique product identifier), and API Key.  Most APIs are restricted in access, and an API Key may be given in order to validate with the third party server.  In this example, the SKU matches with a Project ID in the third party system, and lets me access that particular item via a variable shared in both data sets.

The $url – our API endpoint – is then set up.  We can then start to build the $body based on whatever documentation given to us by the third party. In this scenario, we’ve set up the $body as an array of the various variables we’ve collected.

The $response variable actually sends off the data via the wp\_remote\_post function, which has two parts: the URL, and the payload.  I’ve set some various headers and options above to let WordPress know that our payload should be sent as a JSON request. This, in turn, is received and then the response is sent back as JSON as well.  The $vars variable accepts that request and decodes the JSON into a format that’s readable by our system. In this case, it’s a PHP Variable with key/value pairs.

Finally, I want to store some of the response into the order meta. This will allow us to access those variables in other places – customer/admin emails, and the order screen.

(You may notice above that I also call the SKU of the product. I like doing this, and using it as a variable in the custom meta, so that if multiple unique items are purchased we can store this information – and display it – per product.)

Order Confirmation

In my theme folder, I’ve created a subfolder called ‘woocommerce’. Within that are two more subfolders: emails and order. These two folders – and the included files – are what lets me override the default WooCommerce templates and display some of our newly acquired data.

For our Order confirmation screen, we want to copy the file from ‘/wp-content/plugins/woocommerce/templates/order/order-details-item.php’ into ‘/wp-content/themes/my-theme/woocommerce/order/order-details-item.php’. Right now it’s an exact duplicate. But we want to add a bit of code in.

I like to add this in right after the

on line 49:

<?php if ( has\_term( 'api', 'product\_cat', $product->id ) ) { >

<?php $varproduct = new WC\_Product($item['product\_id']); ?>

<?php $sku = $varproduct->get\_sku();?>

<tr>

<td colspan="2" class="assessment\_link">

<?php

$message = get\_post\_meta( $order->id, 'meta\_message\_'.$sku, TRUE);

$link = get\_post\_meta( $order->id, 'meta\_link\_'.$sku, TRUE);

?>

<p><?php echo $message; ?>. <a target="\_blank" href="<?php echo $link; ?>"><?php echo $link; ?></a></p>

 </td>

</tr>

<?php } ?>

As before, we’re checking to see if the product in question belongs to the API category. If so, we run the next bit of code: creating a $product variable, grabbing the SKU, then calling the post\_meta we stored during the API call. Finally, we’re echoing that message out.

WooCommerce Emails.

There’s two emails I’m concerned with, as a store manager: the admin [new order] email, and the customer [order complete] email.  Both have a separate email template:

* Admin [New Order]: copy ‘/wp-content/plugins/woocommerce/templates/emails/admin-new-order.php’ into ‘/wp-content/themes/my-theme/woocommerce/emails/admin-new-order.php’
* Customer [Order Complete]: copy ‘/wp-content/plugins/woocommerce/templates/emails/customer-completed-order.php’ into ‘/wp-content/themes/my-theme/woocommerce/emails/customer-completed-order.php’

Once again, we’re calling post meta. This time, I like to add it under the do\_action (‘woocommerce\_email\_order\_details’) function:

<?php $items = $order->get\_items();

foreach ( $items as $item ) {

$product\_id = $item['product\_id'];

$product = new WC\_Product($item['product\_id']);

$projectsku = $product->get\_sku();

if ( has\_term( 'api', 'product\_cat', $product\_id ) ) {

echo '<p>';

echo get\_post\_meta($order->id, 'meta\_message\_'.$projectsku, true);

echo '</p><p>';

echo get\_post\_meta($order->id, 'meta\_link\_'.$projectsku, true);

echo '</p>';

}

} ?>

We’re checking to see if the ‘API’ category is attached to an item, grabbing the SKU, then pulling the custom order meta information we need.

Once again – API documentation varies from vendor to vendor, so your mileage may vary.  The nice thing about using WordPress’s built in function? If the response comes back with anything other than a 200 response, the function will fail; meaning that you won’t sell a gift card – for example – if you don’t receive the card number and PIN code back.