Plugin Structure Development

Contribution Section

This is a section for contributors. If you thinking of helping develop the plugin this is where you should look.

How it is built

The first version was built in a halfway careless way; the purpose was to reach out for a PSR-4 variant, but due to the lack of knowledge, that never happened. It was neither never entirely released as the time ran out for it. The current version that is located in Bitbucket is however PSR-4 controlled. The purpose of this is to save as much performance and memory usage as possible, so the decision is put on just only load the parts of the plugin that is necessary to run with at the moment.

What is different compared with the inhouse version?

Everything.

- The codebase. You'd be scared if you look at v2.x
- With filters, actions and detailed logging for problem tracking.
- Data loss protection when bridging orders via Resurs Bank. Instead of having a big load of parameters in the url, we're happy with one single base64-encoded variable than can be decoded on its way back to the checkout without loosing original charset or content.

Sidenotes and details

- The plugin, when creating an order-api URLs, is using Woocommerce-approved components to generate a proper url based on which permalink setup you run with.

Special features

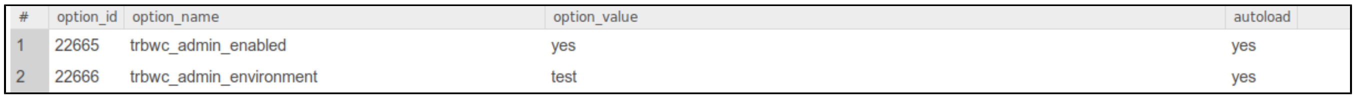
Each order in the order view will save its current API status

Example: Your current credentials are "userTest". This user belongs to the test environment. You do a few test orders with this username and then change to production with "userLive". If you're going back to the old test orders, they will still be viewable but with the old credentials that belonged to the specific order. The metadata that takes care of this part of the plugin will create a unique keyset and encrypt each credential set with this and is not plain text.

This feature can be disabled from the advanced menu.

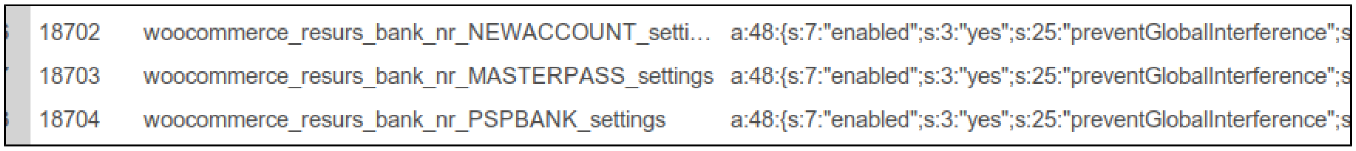
Saving configuration

The admin functions are going into a more native state than before. Instead of bulked configuration rows, all settings are stored in wp_options per keyed row. See image below. This makes the configuration sections in the plugin less sensitive when it comes to serialized arrays. Everything are saved with a unique prefix that indicates that the data belongs to the plugin. The admininstration configuration are now also built with the WC_Admin_Settings API (output_fields method): https://docs.woocommerce.com/wc-apidocs/source-class-WC_Admin_Settings.html#191-671, which increases WooCommerce-version 3.4.0 as the least requirement.



#	option_id	option_name	option_value	autoload
1	22665	trbwc_admin_enabled	yes	yes
2	22666	trbwc_admin_environment	test	yes

The old formatting:



18702	woocommerce_resurs_bank_nr_NEWACCOUNT_setti…	a:48:{s:7:"enabled";s:3:"yes";s:25:"preventGlobalInterference";s
18703	woocommerce_resurs_bank_nr_MASTERPASS_settings	a:48:{s:7:"enabled";s:3:"yes";s:25:"preventGlobalInterference";s
18704	woocommerce_resurs_bank_nr_PSPBANK_settings	a:48:{s:7:"enabled";s:3:"yes";s:25:"preventGlobalInterference";s

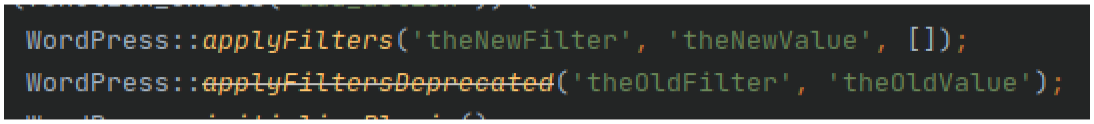
Filter and action execution

The plugin has its own way to handle filters. There's a centralized place for where the magics happen and where the filters gets their proper names. It's not actually necessary, as it always runs the apply_filters method. See below:

Static filter calls

```
use ResursBank\WordPress;
WordPress::applyFilters('theNewFilter', 'theNewValue', []);
WordPress::applyFiltersDeprecated('theOldFilter', 'theOldValue');
```

The above two calls translates the filters to **rbwc_theNewFilter** and **resurs_bank_theOldFilter**. Running an IDE, this will cause this kind of look while using the filters, so the only purpose is to make it a bit clearer if we use new filters or old filters.



WordPress::applyFilters('theNewFilter', 'theNewValue', []);

WordPress::~~applyFiltersDeprecated~~('theOldFilter', 'theOldValue');

Filters are executed like the one above but the constant RESURSBANK_SNAKECASE_FILTERS, in the inital area of the plugin coverts them to snakecase for the moment.

Coding standards, casings, etc (or: should I use snake_case or camelCase usages?)

You can read more about this in the Actions and filters section since this is just a sumup.

In the initiation of the plugin there was a constant added: RESURSBANK_SNAKECASE_FILTERS. The plan was to use camelCases even in the filter /actions. However, many plugins are using snake_case calls so the constant was put there as a permanent setting. When you use the internal **WordPress::applyFilters** and **WordPress::doAction** most of the work are automated. So internally, we hook up with camelCased filters an actions, like this: ***WordPress::doAction('doThisAction')***. When sending this command to the filters and actions, it is in the same time converted to snakecase formatting.

The same thing occurs reversely on AJAX calls. This is something you can read more about in the Scripts and AJAX section. However, very shortly, when snake_case-based AJAX calls arrives, they are translated to camelCase requests in the primary Ajax-call handler ResursBank\Module\PluginApi. If the call above (doThisAction) was an ajax call, the action would be named resursbank_do_this_action. When handled internally it is retranslated to PluginAPI\doThisAction().

Other principles

- Always add since-docblocks in the code, so we always can refer from which version the feature was added. Best practice is to add it when we know which version we're working on, or before the major release if it is long-time pullrequests.
- We follow PSR-schemes and inspections like phpcs, mess detector and beautifyers. "NotCamelCaps"-inspections from WooCommerce are excluded.