Wild Apricot Press (WAP) - Developer's Guide

Wild Apricot Press is a WordPress plugin, written predominantly in PHP for the main functionality, with two CSS files for formatting custom content. This document describes how WAP was designed from a developer's standpoint so that the plugin may be easily modified. Each file is described with greater detail below:

plugin.php

plugin.php is the main file of WAP, in which the title, description, version (among other parameters) are defined. This file creates an instance of each of the classes used in WAP, including *Activator, DataEncryption*, and *MySettingsPage*. This file does not contain much functionality for WAP, but rather creates the classes that do contain the majority of the functionality.

uninstall.php

Like *plugin.php*, this is another standard file required by WordPress. When the plugin is deleted from the WordPress site, the *uninstall.php* file is run to clean up any variables or other data that the plugin may have been using (but would no longer require at this point since the plugin is deleted).

In *uninstall.php,* we can see that all of the wp_options table entries are deleted from the user's database. Depending on if the user selected this setting, the Wild Apricot synced users are deleted from the WordPress user database as well. Learn more about this setting in *MySettingsPage.php*.

Activator.php

This file contains the code associated with the activation of the plugin. That is, this code runs once when the plugin is "activated" in WordPress, which is usually done immediately after installation.

In *Activator.php*, the add-on plugins are checked for and activated, and the Wild Apricot login page is created if valid Wild Apricot credentials have already been entered. Refer to the *activate()* function in *Activator.php* for more description.

Addon.php

This class manages the add-on plugins for WAP.

One prominent function includes checking the license keys against the Integromat scenario to validate that the user has connected their WordPress site to only their registered Wild Apricot site (not another Wild Apricot site). Please see the *validate_license_key()* function.

DataEncryption.php

The *DataEncryption* class is responsible for encrypting and decrypting sensitive data. The code is adapted from this article: https://felix-arntz.me/blog/storing-confidential-data-in-wordpress/

To encrypt or decrypt values, we can do it like this:

TO ENCRYPT:

```
$dataEncryption = new DataEncryption();
$encrypted_value = $dataEncryption->encrypt($value_to_encrypt);
```

TO DECRYPT:

```
$dataEncryption = new DataEncryption();
$decrypted_value = $dataEncryption->decrypt($value_to_decrypt);
```

Deactivator.php

The *Deactivator* class is similar to the *Activator* class, however *Deactivator* is run when the plugin is "deactivated". Most notably, the Wild Apricot login page is hidden and removed from the WordPress website's menu so that Wild Apricot functionality is disabled. Also, the CRON jobs for refreshing data are unset. See more about the CRON jobs under the "WAP's Cron Jobs" section.

MySettingsPage.php

This class manages the settings pages for the WAP plugin; both the user interface and processing the data input on the settings pages. The settings page was set up with the techniques described in Chapter 3 of the "Professional WordPress Development" book.

WAIntegration.php

WAIntegration contains the "meat and potatoes" of the plugin → most of the functionality with access control is done here.

WAWPApi.php

This class manages the API calls to the Wild Apricot API. Each instance of the *WAWPApi* class requires the user's access token and Wild Apricot account ID. (You can see this in the constructor of the class). After passing in the access token and account ID, the plugin can access data from Wild Apricot, including membership levels and more.

css/wawp-styles-admin.css

Contains the CSS styling for the menus on the admin page (i.e. the WAP Settings).

css/wawp-styles-shortcode.css

Contains the CSS styling for front-end WAP interfaces; most notably, this file contains the CSS for the shortcode that renders the WAP login page.

Custom WordPress Hooks

The WAP plugin uses several custom WordPress hooks for controlling its operation; the custom hooks are defined below. The highlighted hooks are hooks for custom cron jobs, which are described on the next page.

Hook Name	Hook Function
wawp_cron_refresh_user_hook	● Runs the Cron Job that refreshes and resyncs the users' Wild Apricot data with their WordPress profiles. ● Runs every 24 hours
wawp_cron_refresh_license_check	● Runs the Cron Job that checks if the license key(s) are still valid ● Runs every 24 hours ● This hook is also triggered when the admin user visits the "Settings" tab in the **Wild Apricot Press** to ensure that a hacker is not manually entering new license keys to switch the Wild Apricot site
wawp_cron_refresh_memberships_hook	● Runs the Cron Job that refreshes and resyncs the membership levels and groups from Wild Apricot to WordPress ● Runs every 24 hours
wawp_wal_credentials_obtained	● Runs when the Wild Apricot credentials and license key(s) have been entered and verified that they are valid ● Inserts the WAP login page and the "Log In"/"Log Out" button to the menu ● Essentially kickstarts the WAP functionality to the actual front-end of the website so that the user's website may use the WAP functionality
wawp_wal_set_login_private	● Runs when the WAP functionality should be removed from the website; this may stem from: ○ User enters new Wild Apricot credentials or license key(s) that are invalid ○ The WAP plugin is deactivated
wawp_create_select_all_checkboxes	● Used to run jQuery that checks all checkboxes in the Wild Apricot Membership checklist on each post/page

WAP's Cron Jobs

WAP has 3 custom cron jobs, primarily for refreshing user data. The table below shows each cron job, the hook that activates it, the function that is run for each instance, and how often the cron job runs.

Cron Job	Hook	Function	Occurrence
Refreshes user data from Wild Apricot to each user's WordPress metadata	wawp_cron_refresh_user_hook	**refresh_user_wa_info()** in *WAIntegration.php*	24 hours
Checks that the saved license key(s) still apply to the corresponding Wild Apricot URL	wawp_cron_refresh_license_check	**check_updated_credentials()** in *WAIntegration.php*	24 hours
Refreshes the membership groups and levels from Wild Apricot to WordPress	wawp_cron_refresh_memberships_hook	**cron_update_wa_memberships()** in *MySettingsPage.php*	24 hours

How WAP works (from a developer's standpoint)

From Initial Installation

1. User downloads the WAP plugin (from the NewPath website or the WordPress plugin store, etc.) as *Wild-Apricot-Press.zip*
2. User uploads plugin to their WordPress site
3. Plugin is activated
    a. Activation code is run → located in *Activation.php*
4. **Wild Apricot Press** menu and settings pages are created and added to the WordPress dashboard
    a. The code for the settings pages is located in *MySettingsPage.php*
5. Admin notices are shown to remind the user to enter their Wild Apricot credentials and license key(s).
6. User enters their Wild Apricot credentials (API Key, Client ID, Client Secret) under **Wild Apricot Press > Authorization**
    a. Inputs are sanitized
    b. Credentials are checked with the Wild Apricot API
        i. If valid, then a green message saying "Valid credentials" is displayed, and the admin notice to input Wild Apricot credentials is removed
        ii. Else, a red message indicating invalid credentials is shown
7. User selects which menu(s) they would like the "Log In"/"Log Out" button to appear on; this is also under **Wild Apricot Press > Authorization**
8. User enters their license key(s) under **Wild Apricot Press > Licensing**
    a. There is one license for each plugin (i.e. WAP has 1 license key, and each add-on has its own license key)
    b. Inputs are sanitized
    c. License keys are checked with the Integromat scenario to see if they are correctly registered with the current WordPress site and Wild Apricot site
        i. If valid, then the WAP plugin is fully activated to the WordPress site, and the custom WordPress hook **"wawp_wal_credentials_obtained"** is run. User is presented with a success message.
        ii. If invalid, a message is shown to inform the user of their invalid license(s)
9. Following a successful step **#8** (that is, both valid Wild Apricot credentials and valid license(s) have been entered), then the "Log In" button is added to the specified menu.
10. If it does not already exist, then the WAP login page is programmatically created. If this page already exists, then it is made public again. The "Log In" button is linked to this page.

Upon Deactivation / Invalid Credentials are Detected

In the case that the WAP plugin is deactivated, whether it be manually through the WordPress admin or due to invalid Wild Apricot credentials or license(s) being entered, the WAP functionality on the front-end of the website is disabled. That is, the users on the WordPress website are no longer able to log into their Wild Apricot accounts on WordPress, and the access restriction is no longer functional.

Upon Re-Activation

If the user has previously deactivated the plugin and then activates the plugin again, the following steps run:

1. Checks if Wild Apricot credentials and license key(s) are valid
    a. If valid, then the WAP functionality is activated again
        i. “wawp_wal_credentials_obtained” hook is run
        ii. Sets up cron events again

Debugging Tips

Unlike other programming languages such as JavaScript or C++, it is sometimes difficult to print out or log variables in PHP, particularly in the WordPress environment. However, there are still several useful techniques for printing out or logging variables, as described below.

Method #1: error_log()

By using the built-in *error_log()* function, you can log variables to the “debug.log” file located in the root of your WordPress installation under the “wp-content” folder. First, ensure that you have turned on debugging for your WordPress installation; to do so, you must add the following code to the “wp-config.php” file in the root of your WordPress installation, as described here: https://wordpress.org/support/article/debugging-in-wordpress/

```
define( 'WP_DEBUG', true );
define( 'WP_DEBUG_LOG', true );
define( 'WP_DEBUG_DISPLAY', true);
```

You can then use the *error_log()* message like this:

```
error_log('This message will be logged in debug.log');
```

Please keep in mind that any errors you get (e.g. undefined variables, uncaught exceptions, etc.) will be logged automatically to the “debug.log” file.

Method #2: my_log_file()

You can also use a custom function called *my_log_file()*, in which you can specify a local file to log messages to. This is not a built-in function, so you would have to include the *my_log_file()* function to every file that you are planning on using. The code for *my_log_file()* is displayed below:

NOTE: Please make sure that you change the $error_dir path to a local path on your computer where you would like to log the values (highlighted below).

```
static function my_log_file( $msg, $name = '' )
      {
            // Print the name of the calling function if $name is left
empty
            $trace=debug_backtrace();
            $name = ( '' == $name ) ? $trace[1]['function'] : $name;

            $error_dir = '/Applications/MAMP/logs/php_error.log';
            $msg = print_r( $msg, true );
```

```
            $log = $name . "  |  " . $msg . "\n";
            error_log( $log, 3, $error_dir );
      }
```

You can then call the *my_log_file()* function like so:

```
my_log_file('Logging this message to my log file!');
my_log_file($variable);
```