Amilia > WordPress API (A2WP)

**TABLE OF CONTENTS**

|  |  |
| --- | --- |
| [Purpose](#Purpose) | p. 2 |
| [Functionality Overview](#Functionality) | p. 2-3 |
| [Authorization](#Authorization) | p. 3-4 |
| [Endpoints and URLs](#Endpoints) | p. 5 |
| [Proxies](#Proxies) |  |
| [Adding/Editing HTML Templates](#Templates) | p. |
| [Adding Custom Functions](#CustomFunctions) | p. |

**PURPOSE**

A2WP’s main goal is to fetch data from SBVPA’s Amilia account (<https://app.amilia.com/Account/en/sbvpa>), format the results, and push the product to SBVPA’s WordPress site. Currently, A2WP accomplishes two functions: it creates new activity posts for each activity added to Amilia, and it updates activity info when someone loads its respective webpage.

**FUNCTIONALITY OVERVIEW**

A2WP is split into two files: a2wp.js and call-a2wp.js. The first file defines the class, and the second file makes the objects. You should hopefully only need to ever work with call-a2wp.js when building out new functionality, but just in case – here’s a brief description of each function within the A2WP class.

**checkRun**

checkRun holds an array “check” with two bools: one determines if the current parent path matches the target parent path, and the other determines if the current child matches the target child.

Take this example URL:

https://sbvpastg.wpenginepowered.com/things-to-do/activity-5927158/

The child would be the last slug in the url (/activity-5927158/) and the parent would be the pathname sans the child (/things-to-do/). The current and target parents are directly compared, while the current and target children are not – the script only needs to know if the child exists and whether or not the script *expects* one to exist.

Both bools in the check array must be true for the script to progress.

**fetchTemplate**

fetchTemplate retrieves text from the HTML template file the user wants their A2WP object to manipulate.

**fetchData**

fetchData grabs data from whatever URL specified. The method will be passed by call() because, while GET is usually used, the GetAmilia proxy requires POST.

**postData**

postData pushes data to the specified url and endpoint. All pushed data will be stored in the bodyData parameter.

**call**

call runs all of the above functions. If checkRun and fetchTemplate return true, call fetches the Amilia and WP data and then passes it into a loop where each custom function runs. This loop generates JSON with all the new data needed to create WP posts, and call then iterates through the JSON and runs postData for each item.

**AUTHORIZATION**

A2WP needs authorization for both Amilia and WordPress.

Amilia’s authorization header and bearer token automatically generate through the URL A2WP currently uses. The GetAmilia proxy uses a WP application password.

The script connects to WordPress by passing the X-WP-Nonce header and grabbing a nonce value that WordPress creates within it’s functions.php file. A2WP can read the value because, when initializing amilia\_to\_wordpress(), it localizes a CDATA array (apiData) with the nonce inside.

Here is the path to functions.php (in WP File Manager): wp-content/themes/sbvpa/functions.php

WP File Manager

A screenshot of a computer

AI-generated content may be incorrect.



Image of functions.php script

A screenshot of a computer program

AI-generated content may be incorrect.

Example of CDATA as seen in “view page source”

A computer screen shot of a computer code

AI-generated content may be incorrect.

**ENDPOINTS AND URLS**

Amilia URL: <https://amilia-img-proxy.azurewebsites.net/api/GetAmilia>

Endpoints:

GET / POST 🡪 activities

WordPress REST URL: <https://sbvpastg.wpenginepowered.com/wp-json/wp/v2/>

Endpoints:

GET 🡪 activities

POST 🡪 activities?slug=activities-{id}

POST 🡪 media

GetImg proxy URL: <https://amilia-img-proxy.azurewebsites.net/api/GetImg>

**PROXIES**

Add later

**ADDING/EDITING HTML TEMPLATES**

HTML templates are important to the script because they are the markup that will go directly into the WP editor box when creating a new post.

A2WP already has one template in the WP File Manager (wp-content/themes/sbvpa/html/). Here’s a snippet of it.

A screen shot of a computer program

AI-generated content may be incorrect.

A2WP’s DOM parser reads the ids so it knows where to insert what information. The ids are not for styling purposes – **please do not remove them**. You can change the format, tags, placement, etc. of all the markup (you can add more classes and ids, too), but the appropriate id must be inside the appropriate tag.

For instance, if you want the schedule summary to be a <p> at the top of the page instead of a list element in the sidebar, just remember to include id=“amilia-wp-activity-schedule-summary” in <p> tag.

You can also create a new HTML template in the same location as activity-template.html, create a new updateDOM function in the A2WP class, and then reference both when you call your object.

**ADDING CUSTOM FUNCTIONS**

Custom functions are declared and then assigned to an A2WP object. Each function runs in the order they are added. The purpose here is to manipulate Amilia and WP data in order to post something to WP.

To edit or create your own object, you can download the script from WP File Manager here: wp-content/themes/sbvpa/js/call-a2wp.js

Scroll down to the bottom of the script until you reach “Call your objects here.” Here, write a new instance of A2WP and pass the following arguments:

* amilia
  + The endpoint and args used to GET Amilia data.
* wp
  + The endpoint and args used to GET or POST WP data.
* targetPath
  + A variable that tells the script where to run so it doesn’t run when someone loads a completely unrelated page on SBVPA.
  + A target path may look like this: "/things-to-do/"
  + If you want the script to run only when a slug exists after the parent, the target path will look like this: "/things-to-do/{path}"
  + If the slug is optional, the target path will look like this: "/things-to-do/{path?}"
* templatePath
  + The path to your template, which ideally exists in WP File Manager. However, if it doesn’t, you don’t need to prepend apiData to your path file.
* categories
  + JSON that links specific variables to category ids.

The new A2WP instantiation should look something like this.

A screen shot of a computer code

AI-generated content may be incorrect.

The addFunc calls here are taking custom functions and passing them into the object. They will run in the order passed, so in the example above, they go **getId 🡪 updateActDOM 🡪 assignCats**.

Each custom function takes a parameter “input” (which is a JSON), adds to input (optional), and then returns input so it can be passed to the next function.

Define your functions under “Make your custom functions here.”

Finally, run your new A2WP object with call().