Rest Routes Pro

Dec 30, 2022

Contents

Rest Routes Pro is a WordPress plugin that lets you extend the WordPress REST API. You can build complexes custom endpoints for native WordPress objects (posts, taxonomies, users, custom fields) as well as for any custom database table.

The beauty of the plugin is that you don't need to touch a single line of PHP code for that. It comes out with a well-designed interface for building your custom endpoints.

CHAPTER 1

Who is that For

Front-end Developers

If you are a WordPress front-end developer or a mobile developer that uses WordPress as a back-end service you would take a lot from Rest Routes. You can build your front-end applications using any framework of your choice (React, Vue, etc.) and then use Rest Routes to design the custom endpoints that they will consume without the need of touching PHP code for it.

Integration with third-party APIs

This plugin is recommended for you that is developing WordPress applications that need to communicate with external servers. You can easily build custom routes that will both expose data to the third-party server and populate data into your WordPress application database.

WordPress Headless Applications

If you are simply using WordPress as a back-end application, which means that you are not using anything related to the front-end of WordPress, such as Themes, then this plugin is fully recommended for you. You can easily build custom routes that will let you expose your back-end data and also populate existing database tables.

CHAPTER 2

Custom Endpoints

You can easily find all the custom routes that you previously created through Rest Routes Pro as well as add a new one through the WordPress admin menu **Rest Routes Pro > Routes**.

2.1 Basic Settings

This is the first section that you will find once editing or creating a new route. There you will find two text fields:

Namespace

Namespaces are the first part of the URL for the endpoint. They should be used as a vendor/package prefix to prevent clashes between custom routes. Namespaces allow for two plugins to add a route of the same name, with different functionality.

Namespaces in general should follow the pattern of vendor/v1, where the vendor is typically your plugin or theme slug, and v1 represents the first version of the API. If you ever need to break compatibility with new endpoints, you can then bump this to v2.

Route name

This is the second part of the custom endpoint and the name of your route that comes out just after the namespace. So given the route below:

wp-json/acme/v1/some-nice-route

The namespace is *acme/v1* and the route name *some-nice-route*.

2.2 Visibility

First of all, you must understand that for every single route you can have many endpoints. So, for instance, for the route above *wp-json/acme/v1/some-nice-route* you can have one endpoint for displaying some posts using the HTTP GET method and another endpoint for creating posts using the HTTP POST method.

Knowing that you can whether switch the visibility of your entire route (both endpoints) or only switch the visibility of some particular endpoint.

For doing that, you must use the green switcher for enabling or disabling.

Once you do that, the selected endpoints or the entire route won't be registered on the list of available WP REST endpoints anymore, until you enable them again.

2.3 Methods

This is the first left box that you will find once editing or creating a new route. There is where you can define which HTTP method you are going to use. You can have up to 4 endpoints on each route and each endpoint should be using a different method. This means that you can't have the same HTTP method for two or more custom endpoints.

In order to follow the recommended practices when working with REST APIs, remember to always use the right method for your endpoints:

- GET: for displaying resources
- POST: for creating resources
- PUT: for editing resources
- DELETE for deleting resources

This is a required field, you can't have one endpoint without a method

2.4 The Source Field

When creating your custom routes, you will notice that several fields in many places contain a field called **Source**. This is where you can give a **Fixed Value**, a **Custom Parameter** (see *custom parameters*), or a **Dynamic** field.

These fields can be found in many places, such as:

- Default filters (e.g.: post id, post type)
- Custom fields filters
- Taxonomies filters
- Custom table filters
- Limit and offset
- and much more

These fields are the ones responsible for customizing your endpoint and that can be by adding some filter or maybe limiting the number of posts displayed through the endpoint.

If you select the **Fixed Value** option, you will need to provide some value for your field and this value will be always used whenever this endpoint will be called.

If you select the **Custom Parameter**, then it will prepare the endpoint for receiving the information from the end-user. See the *custom parameters* section for understanding better how Custom Parameters work.

If you select **Dynamic** then it will display another select field where you can choose some dynamic data for the field. Currently, it works with date and current user data.

If you select **Request body** then the field will parse the whole body of the request and use it in the field.

2.5 Multiple-values Field

Those fields accept one or more values. If you defined the *source field* of it as **Fixed Value** you can separate the values using a comma. When you define it as *custom parameter* you can pass array arguments in the request, something like: **your-url.dev/arg[]=1&arg[]=2&arg[]=3**

2.6 Custom Parameters

This is a very important part of the plugin. This is where you can define custom parameters that can be used by end-users to interact with your custom endpoints.

In order to use that, first of all, you must add it using the **Custom Parameters** section. There are a couple of settings you can define for each parameter:

- Name: the way to identify your parameter and also the name that end-users will use
- Type: you can force a type for your parameter
- Required: marking this option will deny requests that are not passing this parameter
- Default: you can put any default value for your parameter here to be used in case of missing it

After defining the custom parameter it is time to use that somewhere and that you already learned in the *source field* section.

2.7 Conditional Display

This feature will allow you to conditionally display some fields in the output of several endpoint types, such as Posts Display, Taxonomies Display, Users Display, and Custom Table Display.

As soon as you add some field to the output of an endpoint, you will see a section called “Conditional Display”. The next step is to add at least one *custom parameter*. As the last step you will have to fill the 3 fields accordingly:

- Conditional Parameter: this is the parameter that will control whether a field should be displayed in the output or not.
- Compare: the comparison type
- Conditional value: the value that should be present in the custom parameter in order to show the output field.

E.g.: Imagine that you have a custom parameter in the URL called “show_prices”. Then you want to show the price field in the output only if the value of “show_prices” is equal to “yes”. Then this is what you have to do:

- Add a custom parameter called “show_prices”
- Add an output field, the price field you want to conditionally show
- In the output field area, select “show_prices” in the “Conditional Parameter” field, then in “Compare” select “Equal to” and in “Value” type “yes”

2.8 Endpoint Privacy

Your custom routes can be protected or public. If don’t want to protect your custom route, then you can simply ignore this section.

If you want to protect that, then you can choose some capability in the **Endpoint Privacy** section.

For creating custom routes that only administrators can access, you could choose the *manage_options* capability. This would make the request fail if the logged user has no capability of *manage_options* (non-administrators).

By default, Rest Routes comes with a Basic Authentication mechanism, so you can use a WP username and password for doing authenticated requests.

2.9 Endpoint Type

This is a key part of the plugin, where you will define the purpose of your endpoint. There are distinct options available for every Endpoint Type. So, as soon as you switch the type, the right options will be displayed to you.

2.9.1 Posts

You will see in this section endpoint types responsible for creating endpoints for posts as well as associated custom fields and taxonomies.

Display Posts

This should be used for outputting posts as well as associated custom fields and taxonomies.

Note: We recommend using the GET method for this endpoint type in order to follow the best REST practices.

Once you select this Endpoint Type you will find several options that will let you completely customize your endpoint. Those options will let you refine the results that this endpoint will output:

Default Fields Filter

Here you can add many filters for different default post fields.

- **Status**: the status of the post (published, draft, private, etc)
- **Type**: the post type of the post (post, page, product, etc)
- **Title**: the exact title of the post
- **ID**: the id of the post. It accepts *multiple values*
- **ID not**: the id of the post you don't want to return. It accepts *multiple values*
- **Page name**: the name of the page. It accepts *multiple values*
- **Author ID**: the id of the post's author. It accepts *multiple values*
- **Author ID not in**: the id of the post's author you don't want to return. It accepts *multiple values*
- **Author name**: the post's author name.
- **Parent ID**: the post's parent ID. It accepts *multiple values*
- **Parent ID not in**: the post's parent id you don't want to return. It accepts *multiple values*
- **Post search**: the keywords passed here will be used to look for posts by the post title or post content (default WordPress search mechanism)

Once you choose a filter you will see a field called *source field*, you should choose the right option accordingly to your needs.

Query Groups

See the *query groups* section for more information about this one.

Custom Fields Filter

See the *custom fields* section for more information about this one.

Taxonomies Filter

This one is used for adding filters for multiple taxonomies, in case you want to display posts based on one or more terms. Same as the custom fields section here you can add as many filters as you want for the taxonomies and each filter contains a set of fields:

- **Source** (see *source field*)
- **Taxonomy**: the taxonomy that you want to add the filter for
- **Field type**: the term field that you want to add the filter for, possible values are: Term ID, Name, and Slug
- **Query Group**: this field appears only when there is a query group already defined. For more details about this one please check the *query groups* section

Ordering

See the *ordering* section for more information about this one.

Limit and Offset (Pagination)

See the *limit and offset* section for more information about this one.

Output

In this section, you are able to choose which fields you will want to output through the endpoint. By default, all default fields are outputted. Below you will find the complete list of fields that you can expose:

- Attached audios
- Attached images
- Attached videos
- Comment count
- Comment status
- Custom field
- Custom field (Toolset Types)
- Featured image
- GUID
- Menu order
- Parent post: Attached audios
- Parent post: Attached images
- Parent post: Attached videos
- Parent post: Comment count
- Parent post: Comment status
- Parent post: Custom field
- Parent post: Featured image
- Parent post: GUID

- Parent post: Menu order
- Parent post: Permalink
- Parent post: Ping status
- Parent post: Pinged
- Parent post: Post ID
- Parent post: Post author
- Parent post: Post content
- Parent post: Post content filtered
- Parent post: Post date
- Parent post: Post date GMT
- Parent post: Post excerpt
- Parent post: Post format
- Parent post: Post mime type
- Parent post: Post modified
- Parent post: Post modified GMT
- Parent post: Post name
- Parent post: Post parent
- Parent post: Post password
- Parent post: Post status
- Parent post: Post title
- Parent post: Post type
- Parent post: Taxonomy
- Parent post: To ping
- Permalink
- Ping status
- Pinged
- Post ID
- Post author
- Post content
- Post content filtered
- Post date
- Post date GMT
- Post excerpt
- Post format
- Post mime type
- Post modified

- Post modified GMT
- Post name
- Post parent
- Post password
- Post status
- Post title
- Post type
- Taxonomy
- To ping

Edit Posts

This endpoint type can be used to edit some posts as well as associated custom fields and taxonomy terms.

Note: We recommend you to choose the **Editable** method which can be POST, PUT, or PATCH in order to follow the best REST practices.

Default Fields

The very first thing you should do is to define how the endpoint will find the ID of the post to be edited. For this, you have a default field that contains only the *source field*.

See below the complete list of fields that can be edited through this endpoint type:

- Title
- Content
- Excerpt
- Date
- Password
- Parent
- Menu order
- Status
- Type
- Author

Once you choose a filter you will see a field called *source field*, you should choose the right option accordingly to your needs and this will inform the endpoint how it will populate the post fields.

Custom Fields

This section lets you update associated custom fields. If the custom field is not already associated with the post then a new custom field is added and connected.

For each custom field, you will have to fill two fields, the *source field* and "Custom field name". This is required in order to inform the endpoint how to populate the custom field when editing the post.

Notice that you can add as many custom fields as you need.

Taxonomies

This section lets you update the associated taxonomy terms exactly like in the Custom Fields section.

There is an extra option that lets you choose whether you want to append the term to already associated terms or simply disconnect other terms and let only the new one associated with the post.

For each taxonomy, you will have to fill a few fields, the *source field*, the "Taxonomy" which is the taxonomy type, and "Field type" which is the field used to match the taxonomy term and associate it. This is required in order to inform the endpoint on how to populate the taxonomy term when editing the post.

Create Posts

This endpoint type should be used to create new posts as well as associate custom fields and taxonomy terms.

Note: We recommend using the **Creatable** method which is POST in order to follow the best REST practices.

Default Fields

See below the list of fields that can be filled when creating a new post through the endpoint:

- Title
- Content
- Excerpt
- Date
- Password
- Parent
- Menu order
- Status
- Type
- Author

For each default field, you will have to select the *source field* accordingly to the way you want to populate the field of the new post.

Custom Fields

When adding a new post through your custom endpoint you will also be able to associate custom fields to it.

For each custom field, you will have to fill two fields, the *source field* and "Custom field name". This is required in order to inform the endpoint how to populate the custom field when creating the new post.

Taxonomies

When creating a new post you will also be able to associate taxonomy terms or create a new one and associate it to the newly created post.

For each taxonomy, you will have to fill a few fields, the *source field*, the "Taxonomy" which is the taxonomy type, and "Field type" which is the field used to match the taxonomy term and associate it. This is required in order to inform the endpoint on how to populate the taxonomy term when creating the new post.

2.9.2 Taxonomies

In this section, you will see the endpoint types responsible for creating endpoints for taxonomy terms.

Display Taxonomies

This endpoint type should be used whenever you need to display taxonomy terms as well as associated term meta fields.

Note: We recommend using the GET method for this endpoint type in order to follow the best REST practices.

Query Groups

See the *query groups* section for more information about this one.

Custom Fields Filter

See the *custom fields* section for more information about this one.

Ordering

See the *ordering* section for more information about this one.

Limit and Offset (Pagination)

See the *limit and offset* section for more information about this one.

Endpoint Output

In this section, you can define which term fields you will want to output through your endpoint. By default, all the term fields will be outputted.

See below the list of available fields:

- Term ID
- Name
- Slug
- Term group
- Taxonomy ID
- Description
- Count
- Custom field: **this is a special field, if you choose this one you will need to also fill a new field called "Custom field name"**

2.9.3 Users

Now it's time to learn about the endpoint types responsible for handling actions on users as well as connected user meta fields.

Display Users

This endpoint type should be used whenever you need to output information about users.

Note: We recommend using the GET method for this endpoint type in order to follow the best REST practices.

Default Fields Filter

Here is where you can add filters for default user fields, this will let you refine the results of your endpoint.

- User ID in: It accepts *multiple values*
- User login
- User nice name
- Roles: It accepts *multiple values*
- User email
- User URL
- User registered
- User status
- User display name
- Roles in: It accepts *multiple values*
- Blog ID
- Has published posts
- User ID not in: It accepts *multiple values*

Once you choose a filter you will see a field called *source field*, you should choose the right option accordingly to your needs.

Query Groups

See the *query groups* section for more information about this one.

Custom Fields Filter

See the *custom fields* section for more information about this one.

Ordering

See the *ordering* section for more information about this one.

Limit and Offset (Pagination)

See the *limit and offset* section for more information about this one.

Endpoint Output

Here is where you define which user fields should be outputted, by default all the user fields will be outputted. See below the list of fields available:

- User ID
- User login
- User nice name
- User role

- User email
- User URL
- User registered
- User status
- User display name
- Custom field: **this is a special field, if you choose this one you will need to also fill a new field called "Custom field name"**

Create Users

This endpoint type is the one that should be used for creating new users.

Note: We recommend you to choose the **Creatable** method which POST in order to follow the best REST practices.

Default Fields

This section is where you say how the default user fields should be populated when adding new users through the endpoint. The available fields are:

- User login
- User nice name
- User role
- User email
- User URL
- User status
- User display name
- User password

Once you choose a user field you will see a field called *source field*, you should choose the right option accordingly to your needs and this will tell the endpoint how to retrieve the data for the user fields when creating new users.

Custom Fields

When adding a new user through your custom endpoint you will also be able to associate custom fields to it.

For each custom field, you will have to fill two fields, the source field and "Custom field name". This is required in order to inform the endpoint on how to populate the custom field when creating the new user.

Edit Users

This endpoint type can be used to edit some users as well as associated custom fields.

Note: We recommend you to choose the **Editable** method which can be POST, PUT, or PATCH in order to follow the best REST practices.

Default Fields

The very first thing you should do is to define how the endpoint will find the ID of the user to be edited. For this, you have a default field that contains only the *source field*.

See below the complete list of fields that can be edited through this endpoint type:

- User nice name
- User role
- User email
- User URL
- User status
- User display name

Once you choose a filter you will see a field called *source field*, you should choose the right option accordingly to your needs and this will inform the endpoint how it will populate the user fields.

Custom Fields

This section lets you update associated custom fields. If the custom field is not already associated with the user then a new custom field is added and connected.

For each custom field, you will have to fill two fields, the *source field* and "Custom field name". This is required in order to inform the endpoint how to populate the custom field when editing the user.

Notice that you can add as many custom fields as you need.

2.9.4 Custom Tables

The WordPress database structure is very powerful, however, sometimes we still need to create custom tables may be because of performance or for filling a very particular need.

Rest Routes is fully compatible with custom tables, which means that you can create custom endpoints for doing anything with custom tables.

Display Items

This is the endpoint type that can be used whenever you need to display items from custom tables.

Note: We recommend you choose the **Readable** method which is GET in order to follow the best REST practices.

Table Selection

This is a required section, where you should choose which table you will want to output data.

Filter Columns

In this section, you can add filters for the table columns as well as choose the relation type (AND | OR). You can add as many filters as you need and that will refine the results that your endpoint will output.

Ordering

See the *ordering* section for more information about this one.

Limit and Offset (Pagination)

See the *limit and offset* section for more information about this one.

Endpoint Output

Here is where you can define which columns of the table should be displayed in the output. By default, all the columns will be displayed.

Create Items

This endpoint type should be used whenever you need to create items on custom tables.

Note: We recommend choosing the **Creatable** method which is POST in order to follow the best REST practices.

Table Selection

This is a required section, where you should choose which table you will want to create data.

Columns to Populate

Here you should define how you will populate the columns of the custom table.

Once you choose a column you will see a field called *source field*, you should choose the right option accordingly to your needs and this will inform the endpoint how it will populate the custom table field.

Also, we've recently introduced a new field here called "Type to store". This field should be used whenever you want to store the data in the database in a particular format, possible values are: raw, JSON, and serialized.

Edit Items

Table Selection

This is a required section, where you should choose which table you will want to edit data.

Note: We recommend you to choose the **Editable** method which can be POST, PUT, or PATCH in order to follow the best REST practices.

> **Warning:** This is a dangerous endpoint type! You must be sure of what you are doing. This endpoint will let you update both single and a range of entries from any kind of database table, even default WordPress ones. So, pay attention specially to the Filters section and always make a database backup.

Columns to Edit

In this section, you will tell the endpoint which columns should be updated and how.

Once you choose a column you will see a field called *source field*, you should choose the right option accordingly to your needs and this will inform the endpoint how it will populate the custom table fields.

Also, we've recently introduced a new field here called "Type to store". This field should be used whenever you want to store the data in the database in a particular format, possible values are: raw, JSON, and serialized.

Filters

This section is where you should adjust the range of affected custom table entries. You can add as many filters as you need as well as adjust the relation type (AND | OR).

Once you choose a column you will see a field called *source field*, you should choose the right option accordingly to your needs and this will inform the endpoint how it will populate the custom table field.

Delete Items

This endpoint type should be used for deleting entries from custom tables.

Note: We recommend you to choose the **Deletable** method which DELETE in order to follow the best REST practices.

Warning: This is a dangerous endpoint type! You must be sure of what you are doing. This endpoint will let you delete both single and a range of entries from any kind of database table, even default WordPress ones. So, pay attention especially to the Filters section, and always make a database backup.

Table Selection

This is a required section, where you should choose which table you will want to edit data.

Filters

This section is the one that will limit the range of affected entries.

Once you choose a column you will see a field called *source field*, you should choose the right option accordingly to your needs and this will inform the endpoint how it will retrieve the custom table column data.

2.9.5 REST API Call

Use this action if you want to perform a REST call through your own endpoint. This can be very handy when you need to notify a third-party service when a certain action is done.

A few options are available for this endpoint type in the Basic information tab:

- Method: the HTTP request method you want to perform
- URL: the URL you want to call
- Basic authentication: provide the user and password for the basic authentication in case the request is protected

When you go to the "Request parameters" tab, you will be able to add as many parameters as you want in order to compose the URL. Of course, custom parameters will be accepted here and will be carried along from the original request done on your custom endpoint.

In the Request body, you can compose the body of the request as you wish.

In Request headers you can add as many headers as you want, also supporting custom parameters that can be carried from the original request into the new request you will perform.

As an output for the endpoint you have two options:

- Normal field: where you can add as many fields as you wish in order to be rendered as the output of your request after the extra request is done.
- Request response: the response of the extra request will be displayed in the output.

2.9.6 Custom SQL Query

Warning: This is a dangerous endpoint type! You must be sure of what you are doing. This endpoint will let you update both single and a range of entries from any kind of database table, even default WordPress ones. So, pay attention to the query you are writing, you may turn your site completely unusable when using the wrong queries.

Given the disclaimer above, this is a very advanced and useful endpoint. It lets you build complex custom SQL queries for both update, delete, and display table entries.

The custom SQL query that you will save here, will be performed from the custom endpoint and then rendered as output.

Note: You can also use placeholders in order to replace them with the value passed through custom URL parameters. All you need to do is to wrap them with %%. Your query would look like "SELECT * FROM wp_posts WHERE ID=%post_id%", where post_id is the custom URL parameter you previously configured for your endpoint and is now passing when calling the endpoint.

2.10 Query Groups

Query groups are options that will appear when you are dealing with filters for Custom Fields and Taxonomies. This is a way of dealing with complex queries, so you can break the filter in two or more groups.

When working with **WP_Query** the *meta_query* clauses can be nested in order to construct complex queries. For example, for showing products where **color=orange** OR **color=red&size=small** translates to the following in code:

```
$args = array( 'post_type' => 'product', 'meta_query' => array( 'relation' => 'OR', array( 'key' => 'color', 'value' => 'orange', 'compare' => '=', ), array( 'relation' => 'AND', array( 'key' => 'color', 'value' => 'red', 'compare' => '=', ), array( 'key' => 'size', 'value' => 'small', 'compare' => '=', ), ), ), );

$query = new WP_Query( $args );
```

To achieve that with Rest Routes, you should:

- In the **Query Groups** section add a new group choosing the related field as **AND**
- Set the **Main relation type** field under the Custom Fields section to **OR**
- Add a custom field filter for **color**
- Add a new custom field filter for **color** and choose the group you already created
- Add a new custom field filter for **size** and also choose the same group as above

2.11 Custom Fields Filter

Here is where you can add filters for your posts based on values of associated custom fields. You can add as many filters of this kind as you want and each one has a set of fields:

- **Source** (see *source field*)
- **Custom field name**: the exact custom field key stored in the database

- **Compare**: the comparison type for the custom field
- **Type**: the type that the query should treat this field
- **Query Group**: this field appears only when there is a query group already defined. For more details about this one please check the *query groups* section

2.12 Ordering

This one should be used to define an order for the items that you are outputting which can be posts, terms, users, custom table items, etc. You have two groups of fields that you should fill:

- **Order by**

- Source: (see *source field*) - Order by: this is the field that you want to order your results by. It can be default fields or even a custom field - **Order**:

- Source: (see *source field*) - Order: here you can define the direction of the ordering which can be **ASC** or **DESC**

2.13 Limit and Offset

This section is where you define how many items you want to display through the endpoint and also how many items you want to skip. Both groups of fields contains only one field which is the *source field*.

You are also able to add pagination on your site through this section. For this, you will have to:

- Define a way to find the current page number (Page Number), it can be either a fixed number (not very useful in this case) or a custom URL parameter, you can even use the button "Add custom parameter" to automatically add a new custom URL parameter called "page" for you. Then, all you have to do is to connect this new custom parameter with your in the Source select field an then select the right parameter in the dropdown that will appear below the Source one.
- Define how many items per page you will want to display.

After the settings above are done, you will be able to call the endpoint with wp-json/your-endpoint/?page=1, wp-json/your-endpoint/?page=2 and so on.

CHAPTER 3

Third-party Compatibility

Currently, Rest Routes is compatible with the following plugins:

- ACF
- Toolset Types

3.1 ACF

3.1.1 Display Custom Fields

If you have custom fields from ACF you are able to output them through Rest Routes custom endpoints.

This functionality works also with the repeating custom fields, in this case, all the levels/rows of the repeating field will be displayed. You have the choice to optionally display one level deep based on the sub-field key if you want, in this case, you must use the character "|" in the Custom Field Name.

E.g. you have a repeating field called "types" and then a sub-field called "name", you could display all the names by filling the Custom Field Name with "types|name".

3.1.2 Edit Custom Fields

Also, you can edit custom ACF fields, including repeating custom fields. In this case, the same rule as above applies here.

You should use the "|" character in order to define the repeating field + sub-field. For E.g. you want to update the field "name" in a repeating field called "types". You must fill the "Custom Field Name" field with the value "types|name".

3.1.3 Custom Fields in Filters

There is the possibility to filter posts, users, or terms by some ACF custom fields. The same pattern of using the character "|" applies here.

So, you can create an endpoint and add a filter by custom parameter based on an ACF custom field, it can be a repeating field. As in the sections above, you could use “types|name” in order to filter fields where the sub-field “name” of repeating field “Type” is “anything”.

The “types|name” should be placed in the Custom Field Name after adding one item in the Custom Fields filter of some Display endpoint.

CHAPTER 4

Hooks

Like any other good WordPress plugin, we also offer a set of actions and filters that will let you interact with our plugin.

4.1 Actions

4.1.1 Posts

do_action('rest_routes_before_create_posts_callback', $data, $endpoint)

Called right before the creation of posts. Parameters:

- data: the *WP_REST_Request* object
- endpoint: the endpoint object

do_action('rest_routes_after_create_posts_callback', $data, $endpoint, $post_id)

Called right after the creation of posts. Parameters:

- data: the *WP_REST_Request* object
- endpoint: the endpoint object

do_action('rest_routes_before_edit_posts', $data, $endpoint);

Called right before editing a post. Parameters:

- data: the *WP_REST_Request* object
- endpoint: the endpoint object

do_action('rest_routes_after_edit_posts', $data, $endpoint, $post_id);

Called right after editing a post. Parameters:

- data: the *WP_REST_Request* object
- endpoint: the endpoint object

- id: the id of the edited post

4.1.2 Taxonomies

4.1.3 Users

do_action('rest_routes_before_create_users', $data, $endpoint);

Called right before the creation of users. Parameters:

- data: the *WP_REST_Request* object
- endpoint: the endpoint object

do_action('rest_routes_after_create_users', $data, $endpoint, $user_id)

Called right after the creation of users. Parameters:

- data: the *WP_REST_Request* object
- endpoint: the endpoint object

do_action('rest_routes_before_edit_user', $data, $endpoint)

Called right before editing the user. Parameters:

- data: the *WP_REST_Request* object
- endpoint: the endpoint object

do_action('rest_routes_after_edit_user', $data, $endpoint, $user_id)

Called right after editing the user. Parameters:

- data: the *WP_REST_Request* object
- endpoint: the endpoint object
- id: the id of the edited user

4.1.4 Custom Tables

do_action('rest_routes_before_create_custom_table', $data, $endpoint)

Called right before the creation of items on the custom table. Parameters:

- data: the *WP_REST_Request* object
- endpoint: the endpoint object

do_action('rest_routes_after_create_custom_table', $data, $endpoint, $wpdb_new->insert_id)

Called right after the creation of items on the custom table. Parameters:

- data: the *WP_REST_Request* object
- endpoint: the endpoint object
- id: the id of the newly created item

do_action('rest_routes_before_delete_custom_table', $data, $endpoint);

Called right before deleting custom table items. Parameters:

- data: the *WP_REST_Request* object

- endpoint: the endpoint object

do_action('rest_routes_after_delete_custom_table', $data, $endpoint)

Called right after deleting custom table items. Parameters:

- data: the *WP_REST_Request* object
- endpoint: the endpoint object

do_action('rest_routes_before_edit_custom_table_callback', $data, $endpoint)

Called right before editing custom table items. Parameters:

- data: the *WP_REST_Request* object
- endpoint: the endpoint object

do_action('rest_routes_after_edit_custom_table_callback', $data, $endpoint)

Called right after editing custom table items. Parameters:

- data: the *WP_REST_Request* object
- endpoint: the endpoint object

4.1.5 REST API Call

do_action('rest_routes_rest_api_response_body', $body, $endpoint, $request)

Called after the REST API request is performed, providing the response body.

- body: the body of the response
- endpoint: the endpoint object
- request: the *WP_REST_Request* object

do_action('rest_routes_before_call_rest_api_call_callback', $data, $endpoint)

Called before the REST API request is performed.

- data: the *WP_REST_Request* object
- endpoint: the endpoint object

do_action('rest_routes_after_call_rest_api_call_callback', $data, $endpoint)

Called after the REST API request is performed.

- data: the *WP_REST_Request* object
- endpoint: the endpoint object

4.1.6 Custom SQL Query

do_action('rest_routes_before_call_custom_sql_query_callback', $data, $endpoint)

Called right before the custom SQL query is executed.

- data: the *WP_REST_Request* object
- endpoint: the endpoint object

do_action('rest_routes_after_call_custom_sql_query_callback', $data, $endpoint)

Called right after the custom SQL query is executed.

- data: the *WP_REST_Request* object
- endpoint: the endpoint object

4.2 Filters

4.2.1 Posts

apply_filters('rest_routes_output_custom_field', $customFieldValue, $outputField, $elementId)

It lets you filter the value of custom fields that will be outputted through **Display Posts** endpoint type. We use it internally into our compatibility classes for ACF and Toolset Types in order to render the parsed custom field accordingly to the third-party plugin format.

apply_filters('rest_routes_should_process_tax_on_post_creation', true, $endpoint, $data, $id)

It lets you decide if taxonomies should be processed post-creation. Parameters:

- choice: the filtered boolean value which by default is *true*
- endpoint: the endpoint object
- data: the *WP_REST_Request* object
- id: the id of the newly created post

apply_filters('rest_routes_should_process_cf_on_post_creation', true, $endpoint, $data, $id)

It lets you decide if custom fields should be processed on post creation. Parameters:

- choice: the filtered boolean value which by default is *true*
- endpoint: the endpoint object
- data: the *WP_REST_Request* object
- id: the id of the newly created post

apply_filters('rest_routes_create_posts_result', $result, $data, $endpoint, $id)

It lets you filter the result of the endpoint for the creation of posts. Parameters:

- result: the variable that can be filtered which by default is a message containing the id of the newly created post
- data: the *WP_REST_Request* object
- endpoint: the endpoint object
- id: the id of the newly created item

apply_filters('rest_routes_display_posts_results', $result, $data, $endpoint)

It lets you filter the result of the endpoint for displaying posts. Parameters:

- result: the variable that can be filtered which by default is an array of objects
- data: the *WP_REST_Request* object
- endpoint: the endpoint object

apply_filters('rest_routes_should_process_tax_on_post_edit', true, $endpoint, $data, $id)

It lets you decide if taxonomies should be processed when editing a post. Parameters:

- choice: the filtered boolean value which by default is *true*

- endpoint: the endpoint object
- data: the *WP_REST_Request* object
- id: the id of the newly created post

apply_filters('rest_routes_should_process_cf_on_post_edit', true, $endpoint, $data, $id)

It lets you decide if custom fields should be processed when editing a post. Parameters:

- choice: the filtered boolean value which by default is *true*
- endpoint: the endpoint object
- data: the *WP_REST_Request* object
- id: the id of the newly created post

apply_filters('rest_routes_edit_post_result', $result, $data, $endpoint, $id)

It lets you filter the result of the endpoint for editing a post. Parameters:

- result: the variable that can be filtered which by default is an array containing the id of the edited post
- data: the *WP_REST_Request* object
- endpoint: the endpoint object
- id: the id of the edited post

apply_filters('rest_routes_output_parent_field_prefix', 'parent_', $outputField, $post)

It lets you define the prefix of the parent fields in the output of your endpoints. By default, it is "parent_".

apply_filters('rest_routes_should_process_tax_cf_on_post_creation', true, $endpoint, $request_data, $postId)

Filter added in the Create Posts endpoint and says whether term meta fields should be updated on post creation, parameters:

- result: boolean which says whether term meta fields should be updated, default is *true*
- endpoint: the endpoint object
- data: the *WP_REST_Request* object
- id: the id of the post

4.2.2 Taxonomies

apply_filters('rest_routes_display_taxonomies_results', $result, $data, $endpoint)

It lets you filter the result of the endpoint for displaying taxonomy terms. Parameters:

- result: the variable that can be filtered which by default is an array of objects
- data: the *WP_REST_Request* object
- endpoint: the endpoint object

4.2.3 Users

apply_filters(rest_routes_output_custom_field', $customFieldValue, $outputField, $elementId)

It lets you filter the value of custom fields that will be outputted through **Display Users** endpoint type. We use it internally into our compatibility classes for ACF and Toolset Types in order to render the parsed custom field accordingly to the third-party plugin format.

apply_filters('rest_routes_should_process_cf_on_user_create', true, $endpoint, $data, $id)

It lets you decide if custom fields should be processed on user creation. Parameters:

- choice: the filtered boolean value which by default is *true*
- endpoint: the endpoint object
- data: the *WP_REST_Request* object
- id: the id of the newly created user

apply_filters('rest_routes_create_users_result', $result, $data, $endpoint, $id)

It lets you filter the result of the endpoint for the creation of users. Parameters:

- result: the variable that can be filtered which by default is an array containing the id of the newly created item
- data: the *WP_REST_Request* object
- endpoint: the endpoint object
- id: the id of the newly created item

apply_filters('rest_routes_display_users_results', $result, $data, $endpoint)

It lets you filter the result of the endpoint for displaying users. Parameters:

- result: the variable that can be filtered which by default is an array of objects
- data: the *WP_REST_Request* object
- endpoint: the endpoint object

apply_filters('rest_routes_edit_post_result', $result, $data, $endpoint, $id)

It lets you filter the result of the endpoint for editing a user. Parameters:

- result: the variable that can be filtered which by default is an array containing the id of the edited user
- data: the *WP_REST_Request* object
- endpoint: the endpoint object
- id: the id of the edited user

4.2.4 Custom Tables

apply_filters('rest_routes_create_custom_table_result', $result, $data, $endpoint, $id)

It lets you filter the result of the endpoint for the creation of items in a custom table. Parameters:

- result: the variable that can be filtered which by default is an array containing the id of the newly created item
- data: the *WP_REST_Request* object
- endpoint: the endpoint object
- id: the id of the newly created item

apply_filters('rest_routes_display_custom_table_results', $result, $data, $endpoint)

It lets you filter the result of the endpoint for displaying items from custom tables. Parameters:

- result: the variable that can be filtered which by default is an array of objects
- data: the *WP_REST_Request* object
- endpoint: the endpoint object

apply_filters('rest_routes_custom_table_db_credentials', array $connection)

It lets you change the default database used on your custom endpoints for custom tables. By default, it connects to the WordPress database. You can change it by returning an array containing the connection credentials to the desired database: 'username', 'password', 'host', 'database'.

4.2.5 REST API Call

apply_filters('rest_routes_call_rest_api_call_result, $result, $endpoint, $data)

It lets you filter the response of the endpoint:

- result: the result of the response
- endpoint: the endpoint object
- data: the *WP_REST_Request* object

4.2.6 Custom SQL Query

apply_filters('rest_routes_custom_sql_query', $query, $data, $endpoint)

It lets you filter the SQL query that will be performed by the endpoint:

- query: the query saved in the endpoint
- endpoint: the endpoint object
- data: the *WP_REST_Request* object

apply_filters('rest_routes_call_custom_sql_query_result_error, $result, $endpoint, $data)

It lets you filter the result of the request when some error happened:

- result: the result of the query
- endpoint: the endpoint object
- data: the *WP_REST_Request* object

apply_filters('rest_routes_call_custom_sql_query_result_success, $result, $endpoint, $data)

It lets you filter the result of the request when everything went well:

- result: the result of the query
- endpoint: the endpoint object
- data: the *WP_REST_Request* object

4.2.7 Third-Party

apply_filters('rest_routes_disable_wp_rest_route_cache', bool $value)

It lets you decide whether WP Rest Cache will stop caching your custom endpoints built with Rest Routes. By default, it is set to false.

CHAPTER 5

FAQ

5.1 Does this plugin offer REST API authentication?

Yes, Rest Routes comes out with a default Basic Authentication mechanism, so you can perform authenticated requests using WP username and password.

5.2 Can I have more than one endpoint for my route?

Yes. You can have one endpoint for each HTTP verb. So, it means that you can create one route, e.g. acme/products and create two endpoints, one GET for displaying products and one POST for creating a new product. When editing your route you should find a button in the button "Add new endpoint", this is the button that you should click on.

5.3 Can I use dynamic data, such as current user ID, email, etc?

Yes, when adding any filter, you can select "Dynamic" in the Source field. There, you will be able to use current user fields as well as current date fields. Don't forget to use some authenticator plugin if you want to use current user data, such as https://wordpress.org/plugins/jwt-auth/

5.4 Does this plugin works with custom tables or only with default WordPress tables?

Yes, it works with custom tables. You can create endpoints for displaying, creating, editing, and deleting data from any database table.

5.5 Can I work with non-default databases?

Yes, with this *custom table database filter* you can change the database used for the custom table endpoints.