Backbone JavaScript Client

- Using
  - Default values
  - Available methods
  - Accepted options
  - Localizing the API Schema
  - Waiting for the client to load
  - Model examples:
  - Collection examples:
  - Working With Revisions

The REST API includes a JavaScript/Backbone client library.

The library provides an interface for the WP REST API by providing Backbone Models and Collections for all endpoints exposed through the API Schema.

Using

Activate the WP-API plugin. Enqueue the script directly:

```
wp_enqueue_script( 'wp-api' );
```

or as a dependency for your script:

```
wp_enqueue_script( 'my_script', 'path/to/my/script', array( 'wp-api' ) );
```

The library parses the root endpoint (the 'Schema') and creates matching Backbone models and collections. You will now have two root objects available to you: `wp.api.models` and `wp.api.collections`.

The models and collections include:

Models:

- Category
- Comment
- Media
- Page
- PageMeta
- PageRevision
- Post
- PostMeta
- PostRevision
- Schema
- Status
- Tag
- Taxonomy
- Type
- User

Collections:

- Categories
- Comments
- Media
- PageMeta
- PageRevisions
- Pages
- Posts
- Statuses
- Tags
- Taxonomies
- Types
- Users

You can use these endpoints as-is to read, update, create and delete items using standard Backbone methods (fetch, sync, save & destroy for models, sync for collections). You can also extend these objects to make them your own, and build your views on top of them.

Default values

Each model and collection includes a reference to its default values, for example:

`wp.api.models.Post.prototype.args`

- author: null
- comment_status: null
- content: null
- date: null

- date_gmt: null
- excerpt: null
- featured_media: null
- format: null
- modified: null
- modified_gmt: null
- password: null
- ping_status: null
- slug: null
- status: null
- sticky: null
- title: null

Available methods

Each model and collection contains a list of methods the corresponding endpoint supports. For example, models created from `wp.api.models.Post` have a methods array of:

```
["GET", "POST", "PUT", "PATCH", "DELETE"]
```

Accepted options

Each model and collection contains a list of options the corresponding endpoint accepts (note that options are passed as the second parameter when creating models or collections), for example:

`wp.api.collections.Posts.prototype.options`

- author
- context
- filter
- order
- orderby
- page
- per_page
- search
- status

Localizing the API Schema

The client will accept and use a localized schema as part of the `wpApiSettings` object. The Schema is currently not passed by default; instead the client makes an ajax request to the API to load the Schema, then caches it in the browser's session storage (if available). Activating the client-js plugin

with `SCRIPT_DEBUG` enabled uses a localized Schema. Check the client-js example or this branch which attempts to only localize the schema once per client.

Waiting for the client to load

Client startup is asynchronous. If the api schema is localized, the client can start immediately; if not the client makes an ajax request to load the schema. The client exposes a load promise for provide a reliable wait to wait for client to be ready:

```
wp.api.loadPromise.done( function() {
//... use the client here
} )
```

Model examples:

To create a post and edit its categories, make sure you are logged in, then:

```
// Create a new post
var post = new wp.api.models.Post( { title: 'This is a test post' } );
post.save();

// Load an existing post
var post = new wp.api.models.Post( { id: 1 } );
post.fetch();

// Get a collection of the post's categories (returns a promise)
// Uses _embedded data if available, in which case promise resolves immediately.
post.getCategories().done( function( postCategories ) {
  // ... do something with the categories.
  // The new post has an single Category: Uncategorized
  console.log( postCategories[0].name );
  // response -> "Uncategorized"
} );

// Get a posts author User model.
post.getAuthorUser().done( function( user ){
  // ... do something with user
  console.log( user.get( "name" ) );
```

```
} );

// Get a posts featured image Media model.
post.getFeaturedMedia().done( function( image ){
  // ... do something with image
  console.log( image );
} );

// Set the post categories.
post.setCategories( [ "apples", "oranges" ] );

// Get all the categories
var allCategories = new wp.api.collections.Categories()
allCategories.fetch();

var appleCategory = allCategories.findWhere( { slug: "apples" } );

// Add the category to the postCategories collection we previously fetched.
appleCategory.set( "parent_post", post.get( "id" ) );

// Use the POST method so Backbone will not PUT it even though it has an id.
postCategories.create( appleCategory.toJSON(), { type: "POST" } );

// Remove the Uncategorized category
postCategories.at( 0 ).destroy();

// Check the results - re-fetch
postCategories = post.getCategories();

postCategories.at( 0 ).get( "name" );
// response -> "apples"
```

Collection examples:

To get the last 10 posts:

```
var postsCollection = new wp.api.collections.Posts();
postsCollection.fetch();
```

To get the last 25 posts:

```
postsCollection.fetch( { data: { per_page: 25 } } );
```

Use filter to change the order & orderby options:

```
postsCollection.fetch( { data: { 'filter': { 'orderby': 'title', 'order': 'ASC' } } } );
```

All collections support pagination automatically, and you can get the next page of results using `more`:

```
postsCollection.more();
```

To get page 5 of a collection:

```
posts.fetch( { data: { page: 5 } } );
```

Check if the collection has any more posts:

```
posts.hasMore();
```

Working With Revisions

You can access post or page revisions using the PostRevisions or PageRevisions collections or through the Post or Page collection.

For example, to get a collection of all revisions of post ID 1:

```
var revisions = new wp.api.collections.PostRevisions({}, { parent: 1 });
```

Revision collections can also be accessed via their parent's collection. This example makes 2 HTTP requests instead of one, but now the original post and its revisions are available:

```
var post = new wp.api.models.Post( { id: 1 } );
post.fetch();
post.getRevisions().done( function( revisions ){
  console.log( revisions );
});
```

If you add custom endpoints to the API they will also become available as models/collections. For example, you will get new models and collections when you add REST API support to your custom post type. To access custom endpoints created, use the item's JSON endpoint name as CamelCase; for example if the item is found at the JSON path of /wp-json/wp/v2/my-custom-post, it can be

accessed via the api at wp.api.collections.MyCustomPost. Note: Because the schema is stored in the user's session cache to avoid re-fetching, you may need to open a new tab to get a new read of the Schema.

```
// Extend wp.api.models.Post and wp.api.collections.Posts to load a custom post type
const CustomPost = wp.api.models.Post.extend( {
  urlRoot: wpApiSettings.root + 'wp/v2/custom_post_slug',
  defaults: {
    type: 'custom_post_slug',
  },
} );
const CustomPosts = wp.api.collections.Posts.extend( {
  url: wpApiSettings.root + 'wp/v2/custom_post_slug',
  model: BLProduct,
} );
const someCustomPosts = new CustomPosts();
someCustomPosts.fetch().then( ( posts ) => {
  // do something with the custom posts
} );
```

□□□ #2
□ □□□ □□□ 2025-06-04 15:42:29 UTC
□ □□□ □□□ 2025-06-04 15:42:29 UTC