

Coding the Future: A Developer's Guide to Registering WordPress Abilities

Uros - Development Team Lead, Thrive Themes (Awesome Motive)



What the Abilities API is

A core registry of discrete units of functionality with a machine-readable contract.

Each ability declares a name, description, input schema, output schema, category, permission callback and execute callback.

- Feature plugin / Composer package: wordpress/abilities-api
- WordPress 6.9 (Dec 2025): PHP registration, registry, execution, REST routes
- WordPress 7.0: JavaScript client (@wordpress/abilities)
- WordPress 7.1: exposure flags refined (meta.public)

Design goal: a site can answer "what can you do?" to other plugins, scripts and AI agents.

Core concepts

Six words for the rest of the talk.

- **Ability** - namespace/ability-name, an instance of WP_Ability
- **Category** - every ability belongs to exactly one (WP_Ability_Category)
- **Registry** - WP_Abilities_Registry singleton, plus one for categories
- **Callback** - what runs on WP_Ability::execute()
- **Schemas** - JSON Schema draft-4 subset, the REST API validator
- **Permission callback** - can the current user run this, with this input?

The functions and classes you will touch

Your cheat sheet.

- Registration: `wp_register_ability()`,
`wp_register_ability_category()`
- Lookup: `wp_get_ability()`, `wp_get_abilities()`,
`wp_has_ability()`
- Removal: `wp_unregister_ability()`
- Hooks: `wp_abilities_api_categories_init`,
`wp_abilities_api_init`

Plugin -> `wp_register_ability()` -> Registry -> PHP,
REST, JS, MCP. Nothing on the right side needs code
from you.

```
// WP_Ability
$ability->get_name();
$ability->get_label();
$ability->get_description();
$ability->get_category();
$ability->get_input_schema();
$ability->get_output_schema();
$ability->get_meta();

$ability->check_permissions( $input );
$ability->execute( $input );
```

Before vs after

Same feature: hand-wired AJAX + REST vs one registered ability.

```
// BEFORE: one feature, wired up twice
add_action( 'wp_ajax_wcg Galicia_summary',
    function () {
        check_ajax_referer( 'wcg Galicia' );
        current_user_can( 'edit_posts' ) || wp_die();
        $id = absint( $_POST['post_id'] ?? 0 );
        wp_send_json( wcg Galicia_get_post_summary( $id ) );
    } );

add_action( 'rest_api_init', function () {
    register_rest_route( 'wcg Galicia/v1', '/summary',
        [ /* args, permission, callback again */ ] );
} );

// Two permission checks, input cleaned by hand,
// no schema, invisible to AI agents
```

```
// AFTER: register once, use everywhere
add_action( 'wp_abilities_api_init', function () {
    wp_register_ability(
        'wcg Galicia/get-post-summary', [
            'label' => __( 'Get post summary', 'wcg Galicia' ),
            'description' => __( 'Returns title, ...' ),
            'category' => 'wcg Galicia-content',
            'input_schema' => [ /* post_id */ ],
            'output_schema' => [ /* title, excerpt */ ],
            'permission_callback' => fn( $in ) =>
                current_user_can( 'read_post', $in['post_id'] ),
            'execute_callback' =>
                'wcg Galicia_get_post_summary',
            'meta' => [ 'show_in_rest' => true ],
        ] );
} );

// One permission check, validated input,
// works in PHP, REST, JS and MCP
```

Step 1: register a category

Categories go on

wp_abilities_api_categories_init - that hook only.

- Slug: lowercase, dashes only
- Label and description are translatable strings
- The category must exist before any ability that references it

WordPress fires the categories hook first for exactly this reason.

```
add_action(
    'wp_abilities_api_categories_init',
    function () {
        wp_register_ability_category(
            'wcgalicia-content',
            [
                'label' => __( 'WordCamp Content', 'wcgalicia' ),
                'description' => __(
                    'Abilities for WordCamp content.', 'wcgalicia'
                ),
            ]
        );
    }
);
```

Step 2: register the ability

Hook: `wp_abilities_api_init`. Guard with `function_exists()` below 6.9.

- Returns `WP_Ability` or null - null means bad name, missing category or duplicate. It won't throw.
- The description is not a comment: an AI agent reads it to decide whether your ability is the right tool.
- Write it like documentation: what it does, when to use it, what it returns.

```
add_action( 'wp_abilities_api_init', function () {  
    wp_register_ability( 'wcgalicia/get-post-summary', [  
        'label' => __( 'Get post summary', 'wcgalicia' ),  
        'description' => __(  
            'Returns title, excerpt and word count of a post.',  
            'wcgalicia'  
        ),  
        'category' => 'wcgalicia-content',  
        'input_schema' => [ /* ... */ ],  
        'output_schema' => [ /* ... */ ],  
        'permission_callback' => fn( $input ) =>  
            current_user_can( 'read_post', $input['post_id'] ),  
        'execute_callback' => 'wcgalicia_get_post_summary',  
        'meta' => [ /* ... */ ],  
    ] );  
} );
```

Anatomy of the arguments

Required: `label`, `description`, `category`, `execute_callback`, `permission_callback`.

- **input_schema** - optional only when there is no input; if you accept input, define it
- **output_schema** - optional, but mandatory in practice
- **permission_callback** - receives the input; return true, false or `WP_Error`
- **meta** - annotations (readonly, destructive, idempotent) and exposure flags
- **ability_class** - your own `WP_Ability` subclass for custom behaviour (rare)

The execute callback

One argument in, one result out. Transport-agnostic by design.

- \$input arrives already validated and sanitized against input_schema
- The return value is validated against output_schema - a mismatch is a WP_Error
- Return WP_Error yourself for domain failures
- No echo, no wp_die, no \$_POST, no nonce handling

The same function serves a PHP call, a REST request, a JS call and an MCP tool call.

```
function wcg Galicia_get_post_summary( array $input ) {
    $post = get_post( $input['post_id'] );

    if ( ! $post ) {
        return new WP_Error(
            'not_found',
            __( 'Post not found.', 'wcgalicia' ),
            [ 'status' => 404 ]
        );
    }

    return [
        'title'      => $post->post_title,
        'excerpt'    => get_the_excerpt( $post ),
        'word_count' => str_word_count(
            wp_strip_all_tags( $post->post_content )
        ),
    ];
}
```

Naming and namespacing

namespace/ability-name - lowercase, dashes, no underscores.

- Namespace = your plugin slug; stay away from core's namespaces
- Underscores are rejected at registration
- The name is a public API: other plugins and agents will hardcode it
- Verb-first names read well when an agent chooses between tools:

get-, create-, update-, delete-, send-

Input and output schemas

JSON Schema draft-4 subset - the same validator the REST API uses.

- Under the hood: `rest_validate_value_from_schema` / `rest_sanitise_value_from_schema`
- Useful: `type`, `properties`, `required`, `enum`, `minimum`, `format`, `default`, `additionalProperties`
- Describe every property - free documentation for agents
- The output schema is a contract: agents chain your output into another ability's input

```
'input_schema' => [  
  'type'      => 'object',  
  'properties' => [  
    'post_id' => [  
      'type'      => 'integer',  
      'minimum'   => 1,  
      'description' => 'ID of the post to summarize.',  
    ],  
    'length' => [  
      'type'      => 'string',  
      'enum'      => [ 'short', 'long' ],  
      'default'   => 'short',  
    ],  
  ],  
  'required'      => [ 'post_id' ],  
  'additionalProperties' => false,  
],
```

Permission policies

The permission callback receives the input, so checks can be object-level.

- Enforced on every surface: `execute()`, REST /run, MCP - one check, written once
- Return bool, or `WP_Error` for a message and HTTP status
- REST still requires an authenticated request (cookie + nonce, application passwords); the ability check is on top
- Anti-pattern: `__return_true` on anything that writes

```
// Object-level, not role-level
'permission_callback' => function ( $input ) {
    if ( ! current_user_can(
        'edit_post', $input['post_id']
    ) ) {
        return new WP_Error(
            'forbidden',
            __( 'You cannot edit this post.', 'wccgalicia' ),
            [ 'status' => 403 ]
        );
    }
    return true;
},

// Not this
'permission_callback' => '__return_true',
```

Meta: annotations and exposure

Abilities are private by default - not in REST, not in MCP, until you opt in.

- REST /run picks its method from annotations:
readonly -> GET, destructive + idempotent -> DELETE, else POST
- Agents read annotations to decide how careful to be: confirm destructive, retry idempotent
- 6.9 / 7.0: show_in_rest must be explicit; 7.1: meta.public seeds per-channel defaults

```
'meta' => [  
  'annotations' => [  
    'readonly'    => true,  
    'destructive' => false,  
    'idempotent'  => true,  
  ],  
  'show_in_rest' => true,  
  'public'       => true,    // 7.1+  
  'mcp'         => [  
    'public' => true,  
    'type'   => 'tool',  
  ],  
],
```

Consuming from PHP

The registry is the interface between plugins.

- Look up by name, check permissions, execute
- No includes, no knowledge of the other plugin's class names
- `wp_get_abilities()` returns the full list for building your own UI

```
$ability = wp_get_ability(  
    'wccgalicia/get-post-summary'  
);  
  
$input = [ 'post_id' => 42 ];  
  
if ( $ability &&  
    true === $ability->check_permissions( $input ) )  
{  
  
    $result = $ability->execute( $input );  
  
    if ( ! is_wp_error( $result ) ) {  
        echo $result['word_count'];  
    }  
}
```

Consuming from REST

Zero `register_rest_route()` calls. Three endpoints.

- List: `/abilities` - discovery, requires the read capability
- Get: `/abilities/{name}` - full schema and annotations
- Run: `/abilities/{name}/run` - HTTP method decided by annotations
- Base path: `/wp-json/wp-abilities/v1` - `{name}` is namespace/ability-name

Demo: curl the list endpoint and find your schema in the response.

```
GET /wp-json/wp-abilities/v1/abilities
```

```
GET /wp-json/wp-abilities/v1/abilities  
/wcgalicia/get-post-summary
```

```
GET /wp-json/wp-abilities/v1/abilities  
/wcgalicia/get-post-summary/run  
?input[post_id]=42  
&input[length]=short
```

```
# readonly ability -> GET,  
# input goes in the query string
```

Consuming from JavaScript

@wordpress/abilities - WordPress 7.0 and up.

- Server abilities are discovered over REST and executed from JS with the same name and rules
- Client-only abilities can be registered in JS for editor-side actions
- Data store: core/abilities
- Verify exact export names against your version before copying

```
import {
  executeAbility,
  getAbilities,
} from '@wordpress/abilities';

const summary = await executeAbility(
  'wccgalicia/get-post-summary',
  { post_id: 42 }
);

console.log( summary.word_count );
```

Consuming from AI agents: MCP Adapter

WordPress/mcp-adapter turns public abilities into MCP tools, resources and prompts.

- Default server: `/wp-json/mcp/mcp-adapter-default-server`
- Meta-tools: `discover-abilities`, `get-ability-info`, `execute-ability`
- WP-CLI stdio transport for local development with Claude Desktop or Cursor
- Custom servers with a curated list of abilities
- Permission callback and schemas are enforced end to end - no side door

Real world: WooCommerce ships canonical product and order abilities with `show_in_rest` and `mcp.public` enabled.

Gotchas

Fast round.

- Wrong hook (init, rest_api_init) - the ability silently never exists
- Category registered after the ability - null return, notice only with WP_DEBUG
- Underscores in names - rejected
- Output schema is strict - test your outputs
- Vague descriptions - agents misuse or ignore your ability
- Abilities that change data but have no annotations - agents can't tell safe from dangerous
- Composer polyfill on 6.9+ - watch for double loading
- Registration runs every request - keep it cheap, no DB queries in the hook

Takeaways

Start with two or three read-only abilities from your plugin.

Write descriptions like documentation. Add annotations.

Opt in to REST and MCP deliberately, not by default.



Uros

Development Team Lead, Thrive Themes (Awesome Motive)

developer.wordpress.org/apis/abilities-api

github.com/WordPress/abilities-api

github.com/WordPress/mcp-adapter

Thank you - questions?