@wordpress/abilities

- Table of Contents
- Installation
  - As a WordPress Script
- Usage
  - Using with React and WordPress Data
- API Reference
  - Functions
    - getAbilities(args: AbilitiesQueryArgs = {}): Ability[]
    - getAbility(name: string): Ability | undefined
    - getAbilityCategories(): AbilityCategory[]
    - getAbilityCategory(slug: string): AbilityCategory | undefined
    - registerAbility(ability: Ability): void
    - registerAbilityCategory(slug: string, args: AbilityCategoryArgs): void
    - unregisterAbilityCategory(slug: string): void
    - unregisterAbility(name: string): void
    - executeAbility(name: string, input?: Record<string, any>): Promise<any>
  - Store Selectors
- Contributing to this package

Client library for the WordPress Abilities API, providing a standardized way to discover and execute WordPress capabilities.

Table of Contents

- Installation
- Usage
- API Reference

- Development and Testing

Installation

The client is currently available as a part of the Composer package.

As a WordPress Script

When the Abilities API is installed, the client is automatically registered and enqueue in the admin.

Usage

```
// In your WordPress plugin or theme JavaScript
const { getAbilities, getAbility, executeAbility } = wp.abilities;
// or import { getAbilities, getAbility, executeAbility } from '@wordpress/abilities';
depending on your setup

// Get all abilities
const abilities = getAbilities();

// Get a specific ability
const ability = getAbility( 'my-plugin/my-ability' );

// Execute an ability
const result = await executeAbility( 'my-plugin/my-ability', {
    param1: 'value1',
    param2: 'value2',
} );
```

Using with React and WordPress Data

The client includes a data store that integrates with `@wordpress/data` for use in React components:

```
import { useSelect } from '@wordpress/data';
import { store as abilitiesStore } from '@wordpress/abilities';

function MyComponent() {
    const abilities = useSelect(
        ( select ) => select( abilitiesStore ).getAbilities(),
        []
    );

    const specificAbility = useSelect(
        ( select ) =>
            select( abilitiesStore ).getAbility( 'my-plugin/my-ability' ),
        []
    );

    return (
        <div>
            <h2>All Abilities</h2>
            <ul>
                { abilities.map( ( ability ) => (
                    <li key={ ability.name }>
                        <strong>{ ability.label }</strong>:{ ' ' }
                        { ability.description }
                    </li>
                ) ) }
            </ul>
        </div>
    );
}
```

API Reference

Functions

`getAbilities( args: AbilitiesQueryArgs = {} ): Ability[]`

Returns all registered abilities. Optionally filter by category slug.

```
// Get all abilities
const abilities = getAbilities();
console.log( `Found ${ abilities.length } abilities` );

// Get abilities in a specific category
const dataAbilities = getAbilities( { category: 'data-retrieval' } );
console.log( `Found ${ dataAbilities.length } data retrieval abilities` );
```

`getAbility( name: string ): Ability | undefined`

Returns a specific ability by name, or undefined if not found.

```
const ability = getAbility( 'my-plugin/create-post' );
if ( ability ) {
    console.log( `Found ability: ${ ability.label }` );
}
```

`getAbilityCategories(): AbilityCategory[]`

Returns all registered ability categories. Categories are used to organize abilities into logical groups.

```
const categories = getAbilityCategories();
console.log( `Found ${ categories.length } categories` );

categories.forEach( ( category ) => {
    console.log( `${ category.label }: ${ category.description }` );
} );
```

`getAbilityCategory( slug: string ): AbilityCategory | undefined`

Returns a specific ability category by slug, or undefined if not found.

```
const category = getAbilityCategory( 'data-retrieval' );
if ( category ) {
    console.log( `Found category: ${ category.label }` );
    console.log( `Description: ${ category.description }` );
}
```

`registerAbility( ability: Ability ): void`

Registers a client-side ability. Client abilities are executed locally in the browser and must include a callback function and a valid category.

```
import { registerAbility } from '@wordpress/abilities';

registerAbility( {
    name: 'my-plugin/navigate',
    label: 'Navigate to URL',
    description: 'Navigates to a URL within WordPress admin',
    category: 'navigation',
    input_schema: {
        type: 'object',
        properties: {
            url: { type: 'string' },
        },
        required: [ 'url' ],
    },
    callback: async ( { url } ) => {
        window.location.href = url;
        return { success: true };
    },
} );
```

`registerAbilityCategory( slug: string, args: AbilityCategoryArgs ): void`

Registers a client-side ability category. This is useful when registering client-side abilities that introduce new categories not defined by the server.

```
import { registerAbilityCategory } from '@wordpress/abilities';

// Register a new category
registerAbilityCategory( 'block-editor', {
    label: 'Block Editor',
    description: 'Abilities for interacting with the WordPress block editor',
} );

// Register a category with optional metadata
registerAbilityCategory( 'custom-category', {
    label: 'Custom Category',
```

```
    description: 'A category for custom abilities',
    meta: {
        color: '#ff0000',
    },
} );

// Then register abilities using the new category
registerAbility( {
    name: 'my-plugin/insert-block',
    label: 'Insert Block',
    description: 'Inserts a block into the editor',
    category: 'block-editor', // Uses the client-registered category
    callback: async ( { blockType } ) => {
        // Implementation
        return { success: true };
    },
} );
```

`unregisterAbilityCategory( slug: string ): void`

Unregisters an ability category from the store.

```
import { unregisterAbilityCategory } from '@wordpress/abilities';

unregisterAbilityCategory( 'block-editor' );
```

`unregisterAbility( name: string ): void`

Unregisters a client-side ability from the store.

```
import { unregisterAbility } from '@wordpress/abilities';

unregisterAbility( 'my-plugin/navigate' );
```

`executeAbility( name: string, input?: Record<string, any> ): Promise<any>`

Executes an ability with optional input parameters.

```
// Execute an ability
const result = await executeAbility( 'my-plugin/create-item', {
```

```
    title: 'New Item',
    content: 'Item content',
} );
```

Store Selectors

When using with `@wordpress/data`:

- `getAbilities( args: AbilitiesQueryArgs = {} )` – Returns all abilities from the store, optionally filtered by query arguments
- `getAbility( name: string )` – Returns a specific ability from the store
- `getAbilityCategories()` – Returns all categories from the store
- `getAbilityCategory( slug: string )` – Returns a specific category from the store

```
import { useSelect } from '@wordpress/data';
import { store as abilitiesStore } from '@wordpress/abilities';

function MyComponent() {
    // Get all abilities
    const allAbilities = useSelect(
        ( select ) => select( abilitiesStore ).getAbilities(),
        []
    );

    // Get all categories
    const categories = useSelect(
        ( select ) => select( abilitiesStore ).getAbilityCategories(),
        []
    );

    // Get abilities in a specific category
    const dataAbilities = useSelect(
        ( select ) =>
            select( abilitiesStore ).getAbilities( {
                category: 'data-retrieval',
            } ),
        []
    );
```

```
    // Get a specific category
    const dataCategory = useSelect(
        ( select ) =>
            select( abilitiesStore ).getAbilityCategory( 'data-retrieval' ),
        []
    );

    return (
        <div>
            <h2>All Abilities ({ allAbilities.length })</h2>
            <h2>Categories ({ categories.length })</h2>
            <ul>
                { categories.map( ( category ) => (
                    <li key={ category.slug }>
                        <strong>{ category.label }</strong>:{ ' ' }
                        { category.description }
                    </li>
                ) ) }
            </ul>
            <h2>{ dataCategory?.label } Abilities</h2>
            <ul>
                { dataAbilities.map( ( ability ) => (
                    <li key={ ability.name }>{ ability.label }</li>
                ) ) }
            </ul>
        </div>
    );
}
```

Contributing to this package

This is an individual package that's part of the Gutenberg project. The project is organized as a monorepo. It's made up of multiple self-contained software packages, each with a specific purpose. The packages in this monorepo are published to npm and used by WordPress as well as other software projects.

To find out more about contributing to this package or Gutenberg as a whole, please read the project's main contributor guide.

□□□ #2
□ □□□ □□□ 2026-10-31 18:22:12 UTC
□ □□□ □□□ 2026-10-31 18:22:14 UTC