@wordpress/hooks

- Installation
  - Usage
    - The global instance
  - API Usage
- Parameters
  - hookName
  - namespace
  - Events on action/filter add or remove
  - The all hook
- Contributing to this package

A lightweight & efficient EventManager for JavaScript.

Installation

Install the module

```
npm install @wordpress/hooks --save
```

*This package assumes that your code will run in an **ES2015+** environment. If you're using an environment that has limited or no support for such language features and APIs, you should include the polyfill shipped in `@wordpress/babel-preset-default` in your code.*

Usage

In your JavaScript project, use hooks as follows:

```
import { createHooks } from '@wordpress/hooks';
```

```
myObject.hooks = createHooks();
myObject.hooks.addAction(); //etc...
```

The global instance

In the above example, we are creating a custom instance of the `Hooks` object and registering hooks there. The package also creates a default global instance that's accessible through the `defaultHooks` named exports, and its methods are also separately exported one-by-one.

In the WordPress context, that enables API functions to be called via the global `wp.hooks` object, like `wp.hooks.addAction()`, etc.

One notable difference between the JS and PHP hooks API is that in the JS version, `addAction()` and `addFilter()` also need to include a namespace as the second argument. Namespace uniquely identifies a callback in the form `vendor/plugin/function`.

API Usage

- `createHooks()`
- `addAction( 'hookName', 'namespace', callback, priority )`
- `addFilter( 'hookName', 'namespace', callback, priority )`
- `removeAction( 'hookName', 'namespace' )`
- `removeFilter( 'hookName', 'namespace' )`
- `removeAllActions( 'hookName' )`
- `removeAllFilters( 'hookName' )`
- `doAction( 'hookName', arg1, arg2, moreArgs, finalArg )`
- `doActionAsync( 'hookName', arg1, arg2, moreArgs, finalArg )`
- `applyFilters( 'hookName', content, arg1, arg2, moreArgs, finalArg )`
- `applyFiltersAsync( 'hookName', content, arg1, arg2, moreArgs, finalArg )`
- `doingAction( 'hookName' )`
- `doingFilter( 'hookName' )`
- `didAction( 'hookName' )`
- `didFilter( 'hookName' )`
- `hasAction( 'hookName', 'namespace' )`
- `hasFilter( 'hookName', 'namespace' )`
- `actions`
- `filters`
- `defaultHooks`

Parameters

hookName

Should be a non empty string containing only numbers, letters, dashes, periods and underscores. Also, the hook name cannot begin with `__`.

- Type: `String`
- Required: Yes

namespace

Should be a non empty string containing only numbers, letters, dashes, periods, underscores and slashes. It should take the form `vendor/plugin/function`.

- Type: `String`
- Required: Yes

Events on action/filter add or remove

Whenever an action or filter is added or removed, a matching `hookAdded` or `hookRemoved` action is triggered.

- `hookAdded` action is triggered when `addFilter()` or `addAction()` method is called, passing values for `hookName`, `functionName`, `callback` and `priority`.
- `hookRemoved` action is triggered when `removeFilter()` or `removeAction()` method is called, passing values for `hookName` and `functionName`.

The `all` hook

In non-minified builds developers can register a filter or action that will be called on *all* hooks, for example: `addAction( 'all', 'namespace', callbackFunction );`. Useful for debugging, the code supporting the `all` hook is stripped from the production code for performance reasons.

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 18:19:12 UTC
□ □□□ □□□ 2026-10-31 18:19:13 UTC