Usage

- Installation
  - Usage as a Plugin
  - Usage as a Library
    - Step 1. Add the Repository as a Remote
    - Step 2. Add the Repo as a Subtree
    - Step 3. Update the Subtree
  - Loading Action Scheduler
  - Load Order
  - Usage in Themes
- Scheduling Recurring Actions on Activation and Ensuring They Remain Scheduled
  - Best Practice: Schedule on Activation
  - Ensuring Recurring Actions Remain Scheduled
  - Compatibility Notes:
  - Passing arguments

Using Action Scheduler requires:

1. installing the library
2. scheduling an action

Installation

There are two ways to install Action Scheduler:

1. regular WordPress plugin; or
2. a library within your plugin's codebase.

Note that Action Scheduler follows an L-2 dependency version policy. That is, the library requires at least the "latest minus two" version of WordPress and the PHP minimum version requirement of that WordPress version.

Usage as a Plugin

Action Scheduler includes the necessary file headers to be used as a standard WordPress plugin.

To install it as a plugin:

1. Download the .zip archive of the latest stable release
2. Go to the **Plugins > Add New > Upload** administration screen on your WordPress site
3. Select the archive file you just downloaded
4. Click **Install Now**
5. Click **Activate**

Or clone the Git repository into your site's `wp-content/plugins` folder.

Using Action Scheduler as a plugin can be handy for developing against newer versions, rather than having to update the subtree in your codebase. **When installed as a plugin, Action Scheduler does not provide any user interfaces for scheduling actions**. The only way to interact with Action Scheduler is via code.

Usage as a Library

To use Action Scheduler as a library:

1. include the Action Scheduler codebase
2. load the library by including the `action-scheduler.php` file

Using a subtree in your plugin, theme or site's Git repository to include Action Scheduler is the recommended method. Composer can also be used.

To include Action Scheduler as a git subtree:

Step 1. Add the Repository as a Remote

```
git remote add -f subtree-action-scheduler https://github.com/woocommerce/action-scheduler.git
```

Adding the subtree as a remote allows us to refer to it in short from via the name `subtree-action-scheduler`, instead of the full GitHub URL.

Step 2. Add the Repo as a Subtree

```
git subtree add --prefix libraries/action-scheduler subtree-action-scheduler trunk --squash
```

This will add the `trunk` branch of Action Scheduler to your repository in the folder `libraries/action-scheduler`.

You can change the `--prefix` to change where the code is included. Or change the `trunk` branch to a tag, like `2.1.0` to include only a stable version.

Step 3. Update the Subtree

To update Action Scheduler to a new version, use the commands:

```
git fetch subtree-action-scheduler trunk
git subtree pull --prefix libraries/action-scheduler subtree-action-scheduler trunk --squash
```

Loading Action Scheduler

Regardless of how it is installed, to load Action Scheduler, you only need to include the `action-scheduler.php` file, e.g.

```
<?php
require_once( plugin_dir_path( __FILE__ ) . '/libraries/action-scheduler/action-scheduler.php'
);
```

There is no need to call any functions or do else to initialize Action Scheduler.

When the `action-scheduler.php` file is included, Action Scheduler will register the version in that file and then load the most recent version of itself on the site. It will also load the most recent version of all API functions.

Load Order

Action Scheduler will register its version on `'plugins_loaded'` with priority `0` - after all other plugin codebases has been loaded. Therefore **the `action-scheduler.php` file must be included before `'plugins_loaded'` priority `0`**.

It is recommended to load it *when the file including it is included*. However, if you need to load it on a hook, then the hook must occur before `'plugins_loaded'`, or you can use `'plugins_loaded'` with negative priority, like `-10`.

Action Scheduler will later initialize itself on `'init'` with priority `1`. Action Scheduler APIs should not be used until after `'init'` with priority `1`. As described in API Function Availability, you can also use the `'action_scheduler_init'` action hook for this purpose.

Usage in Themes

When using Action Scheduler in themes, it's important to note that if Action Scheduler has been registered by a plugin, then the latest version registered by a plugin will be used, rather than the version included in the theme. This is because of the version dependency handling code using `'plugins_loaded'` since version 1.0.

Scheduling Recurring Actions on Activation and Ensuring They Remain Scheduled

When developing a plugin or extension that relies on recurring actions, it is essential to schedule those actions when the plugin is first activated or updated.

Best Practice: Schedule on Activation

Plugins should always schedule any important recurring actions during activation or upgrade routines. This ensures the action is registered as soon as the plugin is enabled.

Example: Scheduling on Activation

```
/**
 * Schedule recurring action on plugin activation.
 */
function my_plugin_activate() {
    if ( ! as_has_scheduled_action( 'my_plugin_recurring_action' ) ) {
        as_schedule_recurring_action(
            time(),
            HOUR_IN_SECONDS,
            'my_plugin_recurring_action'
        );
    }
}
register_activation_hook( __FILE__, 'my_plugin_activate' );
```

Ensuring Recurring Actions Remain Scheduled

After initial scheduling, there may be rare cases where a recurring action is lost (e.g., due to a database failure or repeated action failures). The `action_scheduler_ensure_recurring_actions` hook provides a reliable opportunity to check and reschedule the action if necessary.

Example: Using the Ensure Hook

```
/**
 * Ensure recurring action remains scheduled.
 */
add_action( 'action_scheduler_ensure_recurring_actions', function() {
    if ( ! as_has_scheduled_action( 'my_plugin_recurring_action' ) ) {
        as_schedule_recurring_action(
            time(),
            HOUR_IN_SECONDS,
            'my_plugin_recurring_action'
        );
    }
});
```

Compatibility Notes:

When using Action Scheduler, be aware of version-specific features and functions. If your plugin may run with older versions of Action Scheduler because it is relying on ActionScheduler to be included separately from your plugin where you cannot be sure of the version being loaded, use the following guidance to ensure compatibility.

New in 3.9.3: `action_scheduler_ensure_recurring_actions` Hook and `as_supports()`

- The `action_scheduler_ensure_recurring_actions` hook and the `as_supports()` API method were introduced in Action Scheduler 3.9.3.
- To use this hook, first check that the current Action Scheduler version supports it.

Example: Ensuring a Recurring Action Remains Scheduled in a Backward Compatible Way

```
/**
 * Schedules the plugin's recurring action if it is not already scheduled.
```

```
 */
function my_plugin_schedule_my_recurring_action() {
	if ( ! as_has_scheduled_action( 'my_plugin_recurring_action' ) ) {
		as_schedule_recurring_action(
			time(),
			HOUR_IN_SECONDS,
			'my_plugin_recurring_action'
		);
	}
}

/**
 * Add the hooks needed for my_plugin_schedule_my_recurring_action() so it can make sure our hook stays scheduled.
 */
add_action( 'init', function () {
	if ( function_exists( 'as_supports' ) && as_supports( 'ensure_recurring_actions_hook' ) ) {
		// Preferred: runs periodically in the background.
		add_action( 'action_scheduler_ensure_recurring_actions', 'my_plugin_schedule_my_recurring_action' );
	} elseif ( is_admin() ) {
		// Fallback: runs on every admin request.
		my_plugin_schedule_my_recurring_action();
	}
} );
```

Note on `as_has_scheduled_action()` (Added in 3.3.0)

- The `as_has_scheduled_action()` function was introduced in Action Scheduler 3.3.0.
- If you need to support even older versions, check if the `as_has_scheduled_action` function exists and fallback to `as_next_scheduled_action()` instead.

For more details on all available API functions, their parameters, and when they were added, refer to the API Reference.

Passing arguments

It is possible to pass arguments to your callbacks. When you initially supply the arguments via a call to `as_schedule_single_action()` or one of its sister functions, they should be in an array. However, your callback function will receive each array item as an individual parameter. Here's an example:

```
// You must specify the number of arguments to be accepted (in this case, 2).
add_action( 'purchase_notification', 'send_purchase_notification', 10, 2 );

// When scheduling the action, provide the arguments as an array.
as_schedule_single_action( time(), 'purchase_notification', array(
    'bob@foo.bar',
    'Learning Action Scheduler (e-book)',
) );

// Your callback should accept the appropriate number of parameters (again, in this case, 2).
function send_purchase_notification( $customer_email, $purchased_item ) {
    wp_mail(
        $customer_email,
        'Thank you!',
        "You purchased $purchased_item successfully."
    );
}
```

The above is a pretty simple illustration, and you would of course need to make changes if you wish to do anything more complex, such as accept a variable number of arguments. However, it hopefully illustrates the basic principles involved.

□□□ #1
□ □□□ □□□ 2026-08-29 22:03:34 UTC
□ □□□ □□□ 2026-08-29 22:03:34 UTC