

Plugin API

Languages: [English](#) • [Español](#) • [日本語](#) • [Português do Brasil](#) • [ไทย](#) • [中文\(简体\)](#) • [\(Add your language\)](#)

Introduction

This page documents the API (Application Programming Interface) hooks available to WordPress plugin developers, and how to use them.

This article assumes you have already read [Writing a Plugin](#), which gives an overview (and many details) of how to develop a plugin. This article is specifically about the API of "Hooks", also known as "Filters" and "Actions", that WordPress uses to set your plugin in motion. These hooks may also be used in themes, as [described here](#).

Note: This information applies to WordPress [Versions 1.2](#) and higher. Before Version 1.2, modifications were called "hacks" and involved editing the source code of WordPress itself.

Hooks, Actions and Filters

Hooks are provided by WordPress to allow your plugin to 'hook into' the rest of WordPress; that is, to call functions in your plugin at specific times, and thereby set your plugin in motion. There are two kinds of hooks:

1. **Actions:** Actions are the hooks that the WordPress core launches at specific points during execution, or when specific events occur. Your plugin can specify that one or more of its PHP functions are executed at these points, using the Action API.

2. **Filters:** Filters are the hooks that WordPress launches to modify text of various types before adding it to the database or sending it to the browser screen. Your plugin can specify that one or more of its PHP functions is executed to modify specific types of text at these times, using the Filter API.

You can sometimes accomplish the same goal with either an action or a filter. For example, if you want your plugin to change the text of a post, you might add an action function to `publish_post` (so the post is modified as it is saved to the database), or a filter function to `the_content` (so the post is modified as it is displayed in the browser screen).

For a thorough listing of all action and filter hooks in WP see Adam Brown's [WordPress Hooks Database](#).

Function Reference

Filter Functions	Actions Functions
<div><ul style="list-style-type: none"><code>has_filter()</code><code>add_filter()</code><code>apply_filters()</code><code>current_filter()</code><code>merge_filters()</code><code>remove_filter()</code><code>remove_all_filters()</code></div>	<div><ul style="list-style-type: none"><code>has_action()</code><code>add_action()</code><code>do_action()</code><code>do_action_ref_array()</code><code>did_action()</code><code>remove_action()</code><code>remove_all_actions()</code></div>
Activation/Deactivation Functions	
<div><ul style="list-style-type: none"><code>register_activation_hook()</code></div>	<div><ul style="list-style-type: none"><code>register_deactivation_hook()</code></div>

Home Page

WordPress Lessons

Getting Started

Working with WordPress

Design and Layout

Advanced Topics

Troubleshooting

Developer Docs

About WordPress

Codex Resources

Community portal

Current events

Recent changes

Random page

Help

Actions

Actions are triggered by specific events that take place in WordPress, such as publishing a post, changing themes, or displaying a page of the admin panel. Your plugin can respond to the event by executing a PHP function, which might do one or more of the following:

- Modify database data
- Send an email message
- Modify what is displayed in the browser screen (admin or end-user)

The basic steps to making this happen (described in more detail below) are:

1. Create the PHP function that should execute when the event occurs, in your plugin file.
2. Hook to the action in WordPress, by calling `add_action()`
3. Put your PHP function in a plugin file, and activate it.

Create an Action Function

The first step in creating an action in your plugin is to create a PHP function with the action functionality of your plugin, and put it in your plugin file (your plugin file must go into the `wp-content/plugins` directory). For example, if you want your friends to get an email message whenever you create a new post, you might define the following function:

```
function email_friends($post_ID) {
    $friends = 'bob@example.org,susie@example.org';
    mail($friends, "sally's blog updated",
        'I just put something on my blog: http://blog.example.com');
    return $post_ID;
}
```

For most actions, your function should accept a single parameter (usually the post or comment ID, depending on the action). Some actions take more than one parameter – check the documentation for the action (if available) or the WordPress source code for more information. Besides the one parameter, you can also access the [global variables of WordPress](#), and call other WordPress functions (or functions in your plugin file).

Any text output by the function (e.g. with `print`) will appear in the page source at the location where the action was invoked.

NOTE: Keep in mind that other plugins or the WordPress core may already be using the function name you have thought of. See the next section, [Avoiding Function Name Collisions](#) for more information.

Avoiding Function Name Collisions

It is possible that someone has created a plugin with a function named the same as one in your plugin!

This is a problem because PHP does not allow multiple functions with the same name. If two plugins provide function with the same name, or a plugin provides a function with a name the same as a WordPress function, the blog could cease to function. There are two ways to avoid this problem.

The first solution is to prefix every function in your plugin with a unique set of characters. If your name is John Q. Public, you might declare your functions as `function jqp_output() {...}`. The likelihood that someone with the same initials does the same thing with their plugin is possible, but low.

The second - and possibly easier - solution is to enclose your plugin functions in a class and call the class methods statically. This sounds more complicated than it is.

Consider this class, which expands on the examples provided above:

```
class emailer {
    function send($post_ID) {
        $friends = 'bob@example.org,susie@example.org';
        mail($friends,"sally's blog updated",'I just put something on my blog: http://blog.example.com');
        return $post_ID;
    }
}

add_action('publish_post', array('emailer', 'send'));
```

This class, called *emailer* has a method *send* that implements the plugin functionality.

The `add_action()` function outside of the class adds the action to WordPress that tells it to call the *send* method when a post is published. The array used in the second parameter tells the plugin system to call the static method of the class 'emailer' named 'send'.

The function *send* is protected from the global namespace by the class declaration. It is not possible to call `send()` directly, and so any other function named *send* will not collide with this one. If you did want to call `send()`, you would need to use a scope resolution operator, like this: `emailer::send()`

The above example is for static methods. If you have an instance of a class then that won't work. To call a method of an instance you need to pass the instance as a variable. Consider the above example modified to take this into account:

```
class emailer {
    function send($post_ID) {
        $friends = 'bob@example.org,susie@example.org';
        mail($friends,"sally's blog updated",'I just put something on my blog: http://blog.example.com");
        return $post_ID;
    }
}
$myEmailClass = new emailer();
add_action('publish_post', array($myEmailClass, 'send'));
```

Classes are a complicated subject. Read more about them in the [PHP documentation on classes](#).

Hook to WordPress

After your function is defined, the next step is to "hook" or register it with WordPress. To do this, call `add_action()` in the global execution space of your plugin file:

```
add_action ( 'hook_name', 'your_function_name', [priority], [accepted_args] );
```

where:

hook_name

The name of an action hook provided by WordPress, that tells what event your function should be associated with.

your_function_name

The name of the function that you want to be executed following the event specified by `hook_name`. This can be a standard php function, a function present in the WordPress core, or a function defined by you in the plugin file (such as `'email_friends'` defined above).

priority

An optional integer argument that can be used to specify the order in which the functions associated with a particular action are executed (default: 10). Lower numbers correspond with earlier execution, and functions with the same priority are executed in the order in which they were added to the action.

accepted_args

An optional integer argument defining how many arguments your function can accept (default 1), useful because some hooks can pass more than one argument to your function. This parameter is new in release 1.5.1.

In the example above, we would put the following line in the plugin file:

```
add_action ( 'publish_post', 'email_friends' );
```

Likewise, you can also [Remove Actions](#) from action hooks. See that section for details.

Install and Activate

The last step in getting your action hook to work is to install the file and activate the plugin. The PHP function you wrote and the `add_action` call must go into a PHP file together, and the PHP file must be installed in the `wp-content/plugins` directory. Once it is installed, you will need to visit the admin section of WordPress and activate your plugin; see [Managing Plugins](#) for more details.

Current Hooks For Actions

See [Plugin API/Action Reference](#) for a current list of action hooks in WordPress, and links to previous versions of WordPress.

Filters

Filters are functions that WordPress passes data through, at certain points in execution, just before taking some action with the data (such as adding it to the database or sending it to the browser screen). Filters sit between the database and the browser (when WordPress is generating pages), and between the browser and the database (when WordPress is adding new posts and comments to the database); most input and output in WordPress passes through at least one filter. WordPress does some filtering by default, and your plugin can add its own filtering.

The basic steps to adding your own filters to WordPress (described in more detail below) are:

1. Create the PHP function that filters the data.
2. Hook to the filter in WordPress, by calling `add_filter()`
3. Put your PHP function in a plugin file, and activate it.

Create a Filter Function

A filter function takes as input the unmodified data, and returns modified data (or in some cases, a null value to indicate the data should be deleted or disregarded). If the data is not modified by your filter, then the original data must be returned so that subsequent plugins can continue to modify the value if necessary.

So, the first step in creating a filter in your plugin is to create a PHP function to do the filtering, and put it in your plugin file (your plugin file must go into the `wp-content/plugins` directory). For example, if you want to make sure that your posts and comments contain no profanity, you might define a variable with a list of forbidden words, and then create the following PHP function:

```
function filter_profanity($content) {
    $profanities = array('badword','alsobad','...');
    $content=str_ireplace($profanities,'{censored}', $content);
    return $content;
}
```

Why does this work without a loop? Because \$profanities is an array, and [str_ireplace](#) loops through the array for you. The [str_ireplace](#) function is used instead of [str_replace](#) because [str_ireplace](#) is case insensitive.

NOTE: Keep in mind that other plugins or the WordPress core may already be using the function name you have thought of. See the [Plugin Development Suggestions](#) for more information.

Hook in your Filter

After your function is defined, the next step is to "hook" or register it with WordPress. To do this, call `add_filter()` in the global execution space of your plugin file:

```
add_filter ( 'hook_name', 'your_filter', [priority], [accepted_args] );
```

where:

hook_name

The name of a filter hook provided by WordPress, which defines when your filter should be applied.

your_filter

The name of the function that you want to use for filtering. This can be a standard PHP function, a function present in the WordPress core, or a function defined by you in the plugin file.

priority

An optional integer argument that can be used to specify the order in which the functions associated with a particular filter are executed (default: 10). Lower numbers correspond with earlier execution, and functions with the same priority are executed in the order in which they were added to the filter.

accepted_args

An optional integer argument defining how many arguments your function can accept (default 1), useful because some hooks can pass more than one argument to your function.

In the example above, we would put the following in the main executing section of the plugin file, to tell WordPress to filter comments

for profanity:

```
add_filter('comment_text','filter_profanity');
```

You can also remove filters from filter hooks using the `remove_filter()` function. See [Removing Actions and Filters](#).

Install and Activate

The last step in getting your filter hook to work is to install the file and activate the plugin. The PHP function you wrote and the `add_filter()` call must go into a PHP file together, and the PHP file must be installed in the `wp-content/plugins` directory. Once it is installed, you will need to visit the admin section of WordPress and activate your plugin; see [Managing Plugins](#) for more details.

Current Hooks for Filters

See [Plugin API/Filter Reference](#) for a current list of filter hooks in WordPress, and links to previous versions of WordPress.

Example

This is an example, [as described by Ozh on the wp-hackers email list](#), for a plugin to modify (or overwrite) the default `bloginfo()` function. This will require modifying a core function behavior.

```
add_filter('bloginfo', 'mybloginfo', 1, 2);
add_filter('bloginfo_url', 'mybloginfo', 1, 2);

function mybloginfo($result='', $show='') {
    switch ($show) {
        case 'wpurl':
            $result = SITE_URL;
            break;
        case 'template_directory':
            $result = TEMPL_DIR;
            break;
        default:
    }
    return $result;
}
```

Removing Actions and Filters

In some cases, you may find that you want your plugin to disable one of the actions or filters built into WordPress, or added by another plugin. You can do that by calling `remove_filter('filter_hook','filter_function')` or `remove_action('action_hook','action_function')`.

For example, `remove_action('publish_post','generic_ping');` would prevent your weblog from sending pings whenever a new post is created.

Note that if a hook was registered using a priority other than the default of 10, then you must also specify the priority in the call to `remove_action()`. Also note that in general, you shouldn't remove anything unless you know what it does and why it does it – check the WordPress or other plugin source code to be sure.

Pluggable Functions

Besides the hooks (actions and filters) described above, another way for a plugin to modify WordPress's behavior is to override WordPress functions. In fact, there is a small set of functions WordPress intends for plugins to redefine. These are called [Pluggable Functions](#) and they are defined in `wp-includes/pluggable.php`. WordPress loads these functions only if they are still undefined after all plugins have been loaded. For more details examine `wp-settings.php` file.

Activation/Deactivation

If your plugin has tasks to complete only at activation or deactivation time, it can use `register_activation_hook` and `register_deactivation_hook`. Many plugins do not need to use these, as the plugins only modify current behavior. However, if your plugin (for example) needs to change a default option on activation, it can use these functions.

[Creating Tables with Plugins](#) has an example using the `register_activation_hook` function to make the database compatible with the current version of the plugin.

See Also

- [Pluggable Functions](#)

- [Filter Reference](#)
- [Action Reference](#)
- Adam Brown's [WordPress Hooks Database](#), a database of all WordPress' hooks, showing which version they come from, and linking to the source code spots that use them. This is the most complete.

Categories: [Plugins](#) | [Advanced Topics](#) | [WordPress Development](#) | [API](#) | [UI Link](#)

[Privacy](#) | [License / GPLv2](#) See also: [WordPress.com](#) | [WordPress.TV](#) | [WordCamp](#) | [WP Jobs](#) | [Matt](#) | [Blog RSS](#)

Gefällt mir

391

16k