Beginner’s Guide to WordPress Plugin Development

The WordPress CMS has changed the face of our Internet and allowed a surge of new ideas to prosper, and its open-source movement holds a strong presence rooted in software and web development.

WordPress is a blogging platform that has the ability to launch into many other scripts such as web forums, job boards, and even a classic webpage Content Management System.

We’ll be going over a few ways to get started in plug-ins development for WordPress. The steps are relatively simple and don’t require immense dedication to study. A rudimentary knowledge of PHP would be useful even with a basic understanding of the WordPress file structure and Administration panel.

In this brief tutorial, we’ll be going over the necessary steps required to create a simple WordPress plug-in. The functionality will be used to develop dynamic excerpts based on the number passed into our function call.

You’ll need to upload the plug-in file and activate it from the Admin panel, then follow up by calling our function from whatever pages we want the excerpt to appear. Links to completed plug-in source code is already added later in this article :)

Have you ever came across a WordPress blog, saw something you liked, and thought; how they did that,...

Read more

Why develop for WordPress?

Plug-ins are a great way to enhance the functionality of your blog by adding extra features. These can be placed anywhere inside your template by function hooks.

Over time the extensibility of WordPress' plug-in system has allowed tremendous growth and hundreds of developer-submitted pieces of software. WordPress offers explicitly such advanced features in its CMS that unique plug-ins are few and far between.

As a developer, you hold complete control over the backend specifics of your weblog. Hiring a PHP developer to create a system plugin would cost a lot more than you may imagine, and the API is relatively easy enough to work with and learn yourself.

As a secondary argument, developing over WordPress is an excellent practice for tuning yourself into other areas. Building smaller plugins and sidebar widgets in WordPress will help you develop an understanding of how the backend system works.

This isn't just limited to WordPress, as you'll gain a deeper understanding of the vast majority of Content Systems.

1. WP folder structure

An introduction to the WordPress folder structure will show the primary app directories. Inside wp-content, you'll find a **plugins** directory. Here is where all of your individual plug-ins will be housed, either single files or properly named sub-directories.

For smaller plug-ins which only require a single .php file, you have the option to place this directly into the plug-ins/ directory. However, when you start developing more complicated applications, it's much more useful to create a subdirectory named after your plug-in.

Inside, you can house JavaScript, CSS, and HTML includes along with your PHP functions.

A `readme.txt` file can also be useful if you're planning on offering your plugin for download. This file should include your name and what the plugin does. As the author, you may also consider including details about each revision and which updates have come out.

2. Starting your PHP file

When creating a new plugin, you'll need to start with a simple PHP file. This can be named anything but should generally reflect your plug-in's official name.

So for example I have created our base code and have named my file hongkiat-excerpt.phps (save and rename the file to .php).

The first lines of your plug-in **must** be comment information for the parsing engine.

This is extremely important as WordPress will be unable to process your file without. Below is an example code snippet you can copy and mold towards your own.

```
<?php /* Plugin Name: Plugin Name here Plugin URI: http://www.yourpluginurlhere.com/ Version: Current Version Author: Name please Description: What does your plugin do and what features does it offer... */
```

The Plugin Name is what will show up in your Admin backend panel when you go to activate. Same with the URI, which will be placed in the details pane inside the plug-ins panel.

Although it’s not required to include a version or description, it does make your plugin look much more professional.

3. WordPress naming conventions and best practices

There are a few ways to actually structure your plug-in.

Many times PHP developers will create an entire class system in order to avoid collisions with functions and variable names. If you are unfamiliar with the advanced OOP functionality of PHP, then it’s best to just write your code in sample functions.

So for our example code, we will write a single function to house our data. We also need to define a few variables which are crucial to implement inside our template files.

Below is an example bit of code taken from our plugin file with the core logic removed.

When writing your sample code, it’s best to follow regulations and guides set up by WordPress. Since there are so many internal functions already defined, you can avoid duplicates by prefixing a label to all your variables and function names.

```
<?php define("HK_EXAMPLE_CONSTANT", "this is a value"); function hk_example_function( $limit ) { // Some code goes here. } ?>
```

In the above example, we prefixed all our setting names with *hongkiat*.

This can be replaced with any keyword of your choosing usually related to your plugin name. The above code is just **sample settings** and shouldn't pertain to our final plug-in.

This is just to give you some insight into how your variable names and function calls should be written.

4. Diving into Filters and Actions

There is another concept noteworthy of mentioning before we jump into our raw code.

Actions and **filters** are two completely different concepts that relate genuinely to the ways they manipulate plugin data.

These two bits of code come standard within the WordPress API. Filters and actions allow for plug-in developers to update bits of code throughout the WordPress admin panel pertaining to your new plug-in.

This means you could add a new tab in the sidebar or additional settings links for your Plug-in options.

CodeLobster PHP IDE

Understanding add_filter()

A **filter** is used on a bit of text or data being passed into WordPress. With filters you are quite literally able to *filter content* through your own custom written functions to change data in any way.

For example, you may create a filter to change `$the_content` which is a variable set by WordPress containing the entire post content of a WordPress article.

For our plug-in we will be taking `$the_content` and shortening the length of characters into an excerpt.

Filters come in handy when you are writing plug-ins to customize the looks and feel of your blog. These are especially popular when writing sidebar widgets or smaller functions to change how a post should be displayed.

Below is a sample line of code showing how to apply a filter.

```
add_filter('wp_title', 'hongkiat_func');
```

Here we are adding a filter into the WordPress page title. Note this code doesn't relate to our official plugin and is only being used as an example here.

The `add_filter` function is native to WordPress and used to add a new filter to a variable found within page content.

In the line above we're targeting `$wp_title` which contains the title of our current page.

We are then passing this variable into a fake function titled `hongkiat_func()` which could then manipulate and return a new title tag for whatever purposes.

Understanding add_action()

Actions are similar to filters in that they don't work on bits of data but instead target pre-defined areas in your templates and admin panel. As an example you can apply an action whenever you update or edit a page's content.

WordPress offers a comprehensive actions list in their API documentation. Below is a small list of example actions for you to get familiar with some of the pre-defined target areas.

- **publish_post** – called when a post is published or when status is changed into "published"
- **save_post** – called when a post/page is created from start or updated
- **wp_head** – called when the template is loaded and runs the `wp_head()` function
- **loop_end** – called immediately after the final post has been processed through the WordPress loop
- **trackback_post** – called whenever a new trackback is added into a post

Again we can see how simple this bit of code boils down to. If you can understand the difference between actions and filters you'll be that much closer to building comprehensive, working WordPress plugins.

Below is another line of code initializing an action function on the `save_post` hook. To clarify again this doesn't pertain to our current developing plugin and is only used as a piece of example code to understand the `add_action()` function.

```
add_action('save_post', 'notify');
```

So here we see a similar setup to before with `add_filter()`. We need 2 variables, the first holds the name of our hook we're targeting.

In this case `save_post` which means whenever a new post is saved we're going to call our function defined in the second position (`notify()`). You could obviously update notify to be whatever function name you'd want to run, however this isn't required for our current example plug-in.

Finishing our plugin logic

Finishing up on our path we'll be adding our final function right into our plug-in file. The API documentation is very specific and provides an excellent resource to developers who may hold advanced questions.

The material may seem difficult if you are not familiar with PHP but take your time with the concepts and things will start to flow naturally!

The function below should be added directly after your plugin's header comment. Alternatively this could also be placed inside your theme's `functions.php` file.

The code is used to create dynamic post content based on a limited range of characters.

So for our example we can limit story excerpts only 55 characters long with the `hk_trim_content()` function. You could easly call this bit of code from a sidebar widget or one of your theme files to replace `$the_content`.

```
<?php function hk_trim_content( $limit ) { $content = explode( ' ', get_the_content(), $limit ); if ( count( $content ) >= $limit ) { array_pop( $content ); $content = implode(" ",$content).'...'; } else { $content = implode(" ",$content); } $content = preg_replace('/\[.+\]/','', $content); $content = apply_filters('the_content', $content); return $content; } ?>
```

It shouldn't be expected that you fully understand all internal variables or functions used here. Just getting a general understanding of how your functions should be written and what an example set would look like is a very good start.

You may also notice we're using a call to `apply_filters` which is another WordPress-specific function.

This is another aspect you don't need to fully grasp but it does help with future programming over WP. Check out the apply_filters reference page for more details and FAQs on the subject.

The core function above is named `hk_trim_content()`. This only requires 1 parameter named `$limit`.

This could also be shortened to `$lim` which should store an integer specifying how many characters to limit your excerpt to. The content is used on full post pages and also static pages (about us, contact).

Therefore, in order to call this function we would need to add the parameter into our template files. This would be placed somewhere possibly in your `index.php` or `loop.php` file(s) and will require you to install the plugin first. Example below:

```
<?php echo hk_trim_content(55); // display page content limited at 55 chars ?>
```

Installing and running the plugin

I've created a sample file for the plugin to demo if you'd like to skip the hard coding.

Simply download this file (save and rename the file to `.php`) or copy/paste the code into a new PHP document and upload this to your `/wp-content/plugins` directory.

HK Demo Plugin Installed

Once completed you'll want to access the WordPress administration panel and browse your current set of plug-ins for the demo just installed. Once you activate nothing new will happen, not until we manually add in our function call.

To do this simply navigate `Appearance -> Editor` and look for **`single.php`**.

This file contains all the template HTML/CSS for your basic article post page. Scroll down until you find `the_content()` and replace with the example code above. This will limit all your article pages to 55 characters no matter what view is being used.

You could also add in this function to similar pages in your templates directory such as `search.php` or `archive.php`.

Conclusion

These are some of the basics to get you started working within WordPress development. The plugin system is vast and contains a great deal of internal functionality.

If you already have an idea for a plug-in try it out on a local installation of WordPress to practice these topics.

If you’re still confused by much of the information you can review the WordPress documentation and search for your answer there.

The development community is full of helpful users and the forums hold archives with questions from years back

---

Revision #2
Created 29 October 2025 02:43:43 by AI API
Updated 19 November 2025 05:08:04 by AI Channel