Objective Best Practices for Plugin Development? [closed]

Starting a community wiki to collect up ***objective*** best practices for plugin development. This question was inspired by @EAMann's comments on wp-hackers.

The idea is to collaborate on what objective best practices might be so that we can potentially eventually use them in some community collaboration review process.

UPDATE: After seeing the first few responses it becomes clear that we need to have only one idea/suggestion/best-practice per answer and people should review the list to ensure there are no duplicates before posting.

<plugin-development> <best-practices> <wiki>

edited Feb 17 '11 at 16:29

community wiki
3 revs, 2 users 100%
MikeSchinkel

closed as not constructive by Rarst ♦ Aug 18 '12 at 18:44

As it currently stands, this question is not a good fit for our Q&A format. We expect answers to be supported by facts, references, or expertise, but this question will likely solicit debate, arguments, polling, or extended discussion. If you feel that this question can be improved and possibly reopened, visit the help center for guidance.

If this question can be reworded to fit the rules in the help center, please edit the question or leave a comment.

37 Answers

Use Actions and Filters

If you think people would like to add or alter some data: **provide apply_filters() before returning**.

> P.S. One thing I find a bit disappointing and that your question addresses is the percentage of plugins that are designed only for end-users, i.e. that have no hooks of their own. Imagine if WordPress were designed like most plugins? It would be inflexible and a very niche solution.
>
> Maybe things would be different if WordPress were to have the ability to auto-install plugins on which other plugins depended? As it is I typically have to write a lot of the functionality I need from scratch because clients want things a certain way and the available plugins, while 90% there, don't allow me the flexibility to update the remaining
>
> 10%.
>
> I really do wish those leading the WordPress community would identify a way to ensure that plugins are rewarded for following best practices (such as adding in hooks for other developers) much like good answers are rewarded on a StackExchange site.

Let's take an example from another question:

> Example: I want to do something in my plugin when someone retweets an article. If there was a custom hook in whatever the popular retweet plugin is that I could hook in to and fire off of, that would be great. There isn't, so I can modify their plugin to include it, but that only works for my copy, and I don't want to try to redistribute that.

Related

- Offer Extensible Forms

edited Jun 9 '12 at 0:50

community wiki
6 revs, 3 users 63%
Arlen Beiler

Uninstalling should remove all of a plugin's data

Upon being removed from a WordPress installation, **a plugin should delete all files, folders, database entries, and tables** which it created as well as the **option values** it created.

Plugins may offer an option to export/import settings, so that settings can be saved outside of WordPress prior to deletion.

Related

- Deactivation should not provoke Data-Loss

edited Sep 29 '10 at 13:32

community wiki
4 revs, 2 users 72%
hakre

3 This should be the default behavior, yes, but it should also prompt the user to keep some data ... like when uninstalling a video game asks you if you want to remove saved games and downloaded material. A user might only be deactivating the plug-in for testing purposes and wouldn't want to go back through setting up their options when they reactivate it. – EAMann ◆ Aug 23 '10 at 14:24

1 I'm only talking about when a plugin is completely removed, not when it is deactivated. – tnorthcutt Aug 23 '10 at 16:33

1 I understand that ... but sometimes I'll delete plug-ins so I can manually re-add them from a backup or beta version that's not yet hosted in the repository ... – EAMann ◆ Aug 23 '10 at 18:29

4 @EAMann: For that and for migrating plugins to another server, a plugin should provide a mechanism to export and import settings. – hakre Aug 24 '10 at 9:18

1 I've seen a few plugins offer an "Uninstall" button in their settings with big red warnings it will delete all data. This is separate from de-activation, and I think a great way to handle it. Not everyone uses the "Delete" button to remove a plugin. – gabrielk Jun 21 '11 at 16:20

Deactivation should not provoke Data-Loss

A plugin *should not* **delete any of its data upon** ***deactivation***.

Related

- Uninstall should remove all plugin's data

edited Aug 25 '10 at 19:37

community wiki
5 revs, 2 users 75%
MikeSchinkel

Load Scripts/CSS with `wp_enqueue_script` and `wp_enqueue_style`

Plugins should not load / attempt to load duplicate versions of JS / CSS files, especially jQuery and other JS files included in WP Core.

Plugins should always use `wp_enqueue_script` and `wp_enqueue_style` when linking JS and CSS files and never directly via `<script>` tags.

Related

- Re-Use existing functions

edited Aug 25 '10 at 18:47

community wiki
4 revs, 3 users 67%
hakre

1 **Suggestion**: Might be worth sticking a small note about using dependancies in there to(since it's part of the enqueue system). – t31os Feb 17 '11 at 18:26

Decouple from WordPress core code

A Plugin *should* **reduce the impact of the WordPress API to the needed minimum** so to

seperate plugin code from WordPress code. This reduces the impact of changes within the WordPress codebase on the plugin. Additionally this improves the cross-version compatibility of your plugin code.

This does not mean to not use WordPress functions (use them, as *Re-Use existing functions* suggests), but not to mesh your code with WordPress functions too much but to seperate your plugins business logic from WordPress functionality.

edited Jul 26 '12 at 3:19

community wiki
4 revs, 2 users 87%
hakre

3 Having been a developer on and off for 23 years I've seen many cycles and many suggested best practices come and go. I've seen the pendulum swing wide both ways. I now believe in moderation. In my earlier years I abstracted everything but I learned that abstraction adds complexity that is often worse than the expected benefit. When we see something that causes us trouble as people we immediately seize upon ways to protect ourselves from it in the future but often the medicine is more harmful than the disease. In the USA Sarbanes-Oxley is an example of this. – MikeSchinkel Aug 26 '10 at 7:01

2 Just from a short question i asked on wp-hackers: They hate you if you use global variables like `$wp_taxonomies` , because they are "internal" (i know there are some that should be used like `$wp_query` or `$post`)... – kaiser Feb 16 '11 at 10:12

Ensure Plugins Generate No Errors with WP_DEBUG

Always test your plugins with `WP_DEBUG` turned on and ideally have it turned on throughout your development process. A plugin should not throw ANY errors with `WP_DEBUG` on. This includes deprecated notices and unchecked indexes.

To turn debugging on, edit your `wp-config.php` file so that the `WP_DEBUG` constant is set to `true` . See the **Codex on Debug** for more details.

edited Aug 25 '10 at 20:13

community wiki
4 revs, 3 users 75%
John P Bloch

5 If I could upvote this answer twice, I would. It's so frustrating when I'm working on a dev site and have to turn WP_DEBUG off because a plugin I need to use is spewing warnings and notices all over the place. – Ian Dunn Aug 28 '12 at 16:04

Protect Plugin Users Privacy

(Previously: Anonymous API Communication)

If a plug-in communicates with an external system or API (e.g. some Webservice), it should do so anonymously or provide the user with an anonymous option that ensures that no data related to the user of the plugin leaks to a second party uncontrolled.

edited Aug 23 '10 at 20:17

community wiki
3 revs, 2 users 84%
EAMann

Prevent SQL Injection with Input Data

A plugin *should* **sanitize all user input retrieved directly or indirectly (e.g. via `$_POST` or `$_GET`)** before using input values to query the MySQL database.

See: Formatting SQL statements.

edited Aug 25 '10 at 19:00

community wiki
3 revs, 2 users 73%
hakre

5 You should also sanitize data coming *out* of the database. Basically, never trust any data that isn't hardcoded. codex.wordpress.org/Data_Validation is a good reference too. – Ian Dunn Jun 20 '11 at 22:57

22:57

Announce Data-Loss on Plugin Uninstallation

Upon uninstallation a plugin *should* **prompt a user that it will be deleting it's data** and receive a confirmation that the user is okay with deleting the data before doing so and a plugin *should* also *allow the user the option to keep the data* upon uninstallation. (This idea from @EAMann.)

Related

- Uninstall should remove all plugin's data
- Deactivation should not provoke to Data-Loss

edited Aug 25 '10 at 19:37

community wiki
5 revs, 2 users 72%
hakre

3 Wordpress itselfs displays a warning message in the admin, that this happens (at least in trunk right now). – hakre Aug 24 '10 at 9:06

I18n support

All output strings should be linked to an appropriate text domain to allow for internationalization by interested parties, even if the developer has no interest in translating their own plug-in.

Note that it is very important to load the language files during the `init` action so the user can hook into the action.
See the Codex: I18n for WordPress Developers

And also this article: Loading WP language files the right way. Which concludes with:

> [...]
> Finally, I would like to point out that is important to **load custom user language files from WP_LANG_DIR before you load the language files that ship with the plugin**. When multiple mo-files are loaded for the same domain, the first found translation will be used. This way the language files provided by the plugin will serve as a fallback for strings not translated by the user.

```
public function load_plugin_textdomain()
{
    $domain = 'my-plugin';
    // The "plugin_locale" filter is also used in load_plugin_textdomain()
    $locale = apply_filters( 'plugin_locale', get_locale(), $domain );

    load_textdomain(
            $domain,
            WP_LANG_DIR . '/my-plugin/' . $domain . '-' . $locale . '.mo'
    );
    load_plugin_textdomain(
            $domain,
            FALSE,
            dirname( plugin_basename(__FILE__) ) . '/languages/'
    );
}
```

edited Nov 25 '12 at 23:17

community wiki
4 revs, 4 users 61%
brasofilo

License Plugins under a GPL Compatible License

Plug-ins and themes *should* be licensed under a WordPress-compatible license. This enables them to be re-distributed with WordPress as a "program." A recommended license is the GPL. *Take care that all code libraries included with the plug-in are compatible with the same license.*

(This has been a problem and serious point of debate both in the past and present.)

edited Aug 25 '10 at 20:41

community wiki
6 revs, 3 users 50%
EAMann

Prefix All Global Namespace Items

A plugin should properly prefix ALL global namespace items (constants, functions, classes,

variables, even things like custom taxonomies, post types, widgets, etc.). For example, do not create a function called `init()` ; instead, name it something like `jpb_init()` .

Its common should use a three or four letter prefix in front of names or to make use of the PHP Namespace Feature. Compare: Single-letter prefix for PHP class constants?

Related

- Global Namespace is a limited Resource

edited Aug 25 '10 at 19:38

community wiki
4 revs, 3 users 65%
hakre

Minimize Names Added to the Global Namespace

A plugin *should* **reduce it's impact** as much as possible by **minimizing the number of names it adds to the global namespace**.

This can be done by encapsulating the plugin's functions into a class or by using the PHP namespaces feature. Prefixing everything can help as well but is not that flexible.

Next to functions and classes, a plugin *should not* introduce global variables. Using classes normally obsoletes them and it simplifies plugin maintenance.

Related

- Prefix Properly

edited Aug 25 '10 at 19:55

community wiki
4 revs, 2 users 88%
hakre

Minimize Side-effects of Remote Datasources and Webservices

A Plugin *should* **Cache/Shield Webservice** and/or **XMLRPC/SOAP requests through a caching/data-provider layer** if you use them so to not making front-requests waiting for (slow) webservice response.

That includes the download of RSS feed and other pages. Design your plugins that they request data in background.

One possible STEP is (Take posting to ping.fm as an example): Create a buffer table, let's say: ping_fm_buffer_post(date, time, message, submitted_time, status)

1. For every time you want to submit update to ping.fm, add it to this table.
2. Now, we need to create a plugin to handle this data. This plugin will run via crontab to check for every update that's not submitted yet
3. Because we have this table, we can also list every message submitted to ping.fm and

   check each post's status. Just in case there's problem on ping.fm's side, we can re-submit it.

edited Aug 30 '10 at 2:53

community wiki
4 revs, 3 users 50%
hakre

Offer Extensible Forms

When a plugin offers the possiblity to input data, it should always have a hook at the end, right before the "submit" and/or "reset" button, so developers can easily extend the form with not only fields, but buttons too

with not only fields, but buttons too.

See: Settings API

Related

- Use Actions and Filters
- Re-Use existing functions
- I18n support

edited Aug 26 '10 at 22:25

community wiki
5 revs, 3 users 74%
hakre

Die with style

die in a decent manner All of a plugins (and even themes) functions should use `wp_die()` in critical places to offer the user a little information on what had happened. Php errors are annoying and `wp_die` can give the user a nice styled message on what the plugin (or they) did wrong. Plus, if the user has debugging deactivated the plugin will just break.

Using `wp_die()` also helps that your plugins / themes are compatible with the wordpress testsuite.

Related:

- Use WP Error handling

edited May 3 '11 at 15:16

community wiki
6 revs, 2 users 50%
hakre

First Use Existing Functions in WordPress Core

If you can: **use existing functions included in WordPress core** instead of writing your own. Only develop custom PHP functions when there is not an appropriate pre-existing

function in WordPress core.

One benefit is you can use *"log deprecated notices"* to easily monitor functions that should be replaced. Another benefit is users can view the function documentation in the Codex and better understand what the plugin does even if they are not an experienced PHP developer.

Related

- I18n support
- Load Scripts/CSS with wp_enqueue_script and wp_enqueue_style
- Offer Extensible Forms

edited Aug 29 '10 at 14:09

community wiki
7 revs, 3 users 67%
hakre

4 A problem with that is that a lot of core isn't really structurally designed for reusability. I just had to copy and slightly modify half the image manipulation/metadata functions to create my own attachment-like behaving post-type, just because a function like downsize() calls some function that includes a hardcoded check for post-type='attachment'. There's lots of that like the inflexible wp_count_posts() being another example. Before you can really reuse core WP needs a complete refactoring. – wyrfel Feb 16 '11 at 10:04

Use Proper Names

Name hooks and filters (classes, functions and vars), so **that people can identify them in even a year**, when they don't remember any more, where that piece of cake or code comes from. It doesn't matter if hook/filter names get long. Ex. youruniquename_hook/filter_whatitdoes.

- If your file contains a class named "dbdbInit", then the file that contains the class should be named "`dbdbInit.class.php`".
- If you got inside your `dbdbInit`-class a function that registers ex. custom_post_types, then call it `register_custom_post_types()`

then call it `register_custom_post_types()`.

- If you got an array that contains the names for custom_post_types, then call the variable where the array is assigned `$custom_post_type_names`.
- If you got a function that handles an array write `function array_handler( $array ) { // handle the array}`..
- Just try to name things in a way that you know what the sign does/where it belongs to by it's name.

Another thing: If you have to debug stuff then, in 99% of all cases, you get all your messages for not only your code, but for wordpress too. So try to use the same prefix ex. " `dbdb` " for your classes, public functions and variables/objects. This way you can find them easily between hundreds of files. (Wordpress loads 64 files before your theme and about 1,550 functions, not talking about hooks and filters.)

edited Feb 17 '11 at 16:35

community wiki
5 revs, 4 users 67%
kaiser

1 sticking to the WP standards when naming files and functions is also nice, e.g. "class-db-init.php" codex.wordpress.org/WordPress_Coding_Standards – gabrielk Jun 21 '11 at 16:29

Comment using PhpDoc

Best practice is close to the PhpDoc style. If you don't use an IDE like "Eclipse", you can just take a look at the PhpDoc Manual.

You don't have to know exactly how this works. Professional Developers can read the code anyway and just need this as a summary. Hobby coders and users might appreciate the way you explain it on the same knowledge level.

edited Aug 25 '10 at 20:00

community wiki
2 revs, 2 users 82%
kaiser

Organize your code

It's alway hard to read code that's not written in the order it get's executed. First include/require, define, wp_enqueue_style & _script, etc., then the functions that the plugin/theme needs and at last the builder (ex. admin screen, stuff that integrates in the theme, etc.).

Try to separate things like css and js in their own folders. Also try to do this with functions that are only helpers, like array flatteners and similar. Keeping the "main" file as clean and easy to read as possible is a way that helps users, developers and you, when you try to update in a year and haven't seen the code for a longer time.

It's also good to have a structure you repeat often, so you always find your way through. Developing in a known structure on different projects will give you time to make it better and even if your client switches to another developer, you will never hear "he left a chaos". This builds your reputation and should be a long term goal.

edited Jan 6 '11 at 21:12

community wiki
2 revs, 2 users 92%
kaiser

Use wp options for plugin output strings

In order to make the plugin easily used and customizable, all the output strings should be modifiable. The best way to do that is use wp-options to store the output strings and provide backend to change the default values. Plugin shouldn't use displayed strings that cannot be easily changed using the plugin backend.

For example: Sociable - gives you the ability to change the sentence that appears before the icons part "share and enjoy:"

edited Aug 26 '10 at 18:39

community wiki
2 revs

hannit cohen

Host Plugins on WordPress.org

Use the SVN repository provided on WordPress.org for hosting plugins. It makes for an easier update user-experience and if you've never used SVN before, it gets you to actually understand by using it in a context that justifies it.

edited Aug 25 '10 at 19:45

community wiki
2 revs, 2 users 75%
MikeSchinkel

Provide Access Control by Using Permissions

In many instances, users may not want everyone to have access to areas created by your plugin especially with plugins that do multiple complex operations, a single hardcoded capability check may not be enough.

At the very least, have appropriate capability checks for all of the different kind of procedures your plugin can be used for.

edited Aug 25 '10 at 19:47

community wiki
2 revs, 2 users 50%
eddiemoya

Import / Export Plugin Settings

It's not that common across plugins, but if your plugin has (some) settings, it *should* **provide Import / Export of data like configuration and user input**.

Import/Export improves the usability of a plugin.

An example-plugin that has such an import and export functionality (and as well an undo mechanism) is Breadcrumb NavXT (Wordpress Plugin) (full disclosure: some little code by me in there, most has been done by mtekk).

Related

- Uninstall should remove all plugin's data
- Deactivation should not provoke Data-Loss

answered Aug 26 '10 at 8:19

community wiki
hakre

Use the Settings API before add_option

Instead of adding options to the DB via the add_option function, you should store them as an array with using the Settings API that takes care of everything for you.

Use the Theme Modifications API before add_option

The Modifications API is a pretty simple construct and a safe way that allows adding and retrieving options. Everything gets saved as serialized value in your database. Easy, safe & simple.

edited Oct 28 '11 at 9:51

community wiki
4 revs, 2 users 94%
kaiser

1 And furthermore, use `update_option` and never `add_option`, the update function will create the option when it does not exist.. :) – t31os Feb 17 '11 at 18:32

3 I wouldn't say never use `add_option`. There is a good use case for `add_option` where if the option is already set, it isn't changed, so I use it in activation to preserve possibly already existing user preferences. – ProfK Oct 28 '11 at 5:47

1 Another use case for `add_option` is when you want to explicitly disable auto loading. `update option` will force autoload to true, so you want to disable autoload, use `add option` when

initially creating the option. – goto10 Jul 26 '12 at 5:24

Your plugin description should accurately detail the functions of your plugin. There are 10 featured post plugins. All of them display featured posts, yet many have different features. It should be easy to compare your plugin to similar plugins by reading the description.

You should avoid bragging about how simple your plugin is unless it really is very basic. You should include useful links in the description like the link to the settings.

answered Aug 30 '10 at 3:41

community wiki
Greg

Use some kind of templating mechanism

Many plugins tend to mingle code and html freely, which makes maintenance and debugging pretty hard. A better way is to put all your html in separate files and use something like this:

```
function parseTemplate($file, $options) {
    $template = file_get_contents($file);
    preg_match_all("!\{([^{]*)\}!", $template, $matches);

    $replacements = array();
    for ($i = 0; $i < count($matches[1]); $i++) {
        $key = $matches[1][$i];
        if (isset($options[$key])) {
            $val = $matches[0][$i];
            $template = str_replace($val, $options[$key], $template);
        }
    }

    return $template;
}
```

Then use it like this:

```
// In mytemplate.html
<h1>{title}</h1>
<h2>{date}</h2>
<p>{text}</p>

// In your functions.php
echo parseTemplate("mytemplate.html", array(
    "title" => $title,
    "date" => $date,
    "text" => $somethingelse
));
```

answered Sep 1 '10 at 14:53

community wiki
Husky

3 –1 I think additional level of abstraction can do more harm than good. And where it does make sense (content that should be easily editable via form for example) it is better to use tools already available (like shortcodes) rather than add own templating functionality. – Rarst ◆ Jan 10 '11 at 7:46

2 Using a template system seems rather silly. There is no valid reason why you can't just use simple php files for views. – Backie Feb 16 '11 at 10:49

2 –1 – WordPress' templating system is `.PHP` . Adding another template system adds complexity and reduces performance, neither of which are a best practice. – MikeSchinkel Feb 16 '11 at 15:31

Use a class and object orientated PHP5 code

There's no reason not to write clean, object-orientated PHP5 code. PHP4 support will phase out after the next release (WP 3.1). Of course, you can prefix all your function names to end up with endlessly_long_function_names_with_lots_of_underscores, but it's much easier to just write a simple class and bundle everything in that. Also, put your class in a seperate file

and name it accordingly so you can easily extend and maintain it:

```
// in functions.php
require 'inc/class-my-cool-plugin.php';
new MyCoolPlugin();

// in inc/class-my-cool-plugin.php
class MyCoolPlugin {
    function __construct() {
        // add filter hooks, wp_enqueue_script, etc.

        // To assign a method from your class to a WP
        // function do something like this
        add_action('admin_menu', array($this, "admin"));
    }

    public function admin() {
        // public methods, for use outside of the class
        // Note that methods used in other WP functions
        // (such as add_action) should be public
    }

    private function somethingelse() {
        // methods you only use inside this class
    }
}
```

answered Sep 1 '10 at 14:58

community wiki
Husky

5 it is just one of those best practices that make it nicer for everyone. – Arlen Beiler Oct 19 '10 at 2:01

2 @IanDunn: if you want PHP4 support, but PHP4 support has been dropped since 2008, over 4 years ago. There's no reason to still be using PHP4-specific checks. – Husky Aug 29 '12 at 13:42

---

include function always via Hook, not directly.

Example:

- Dont use for include the class of the plugin via new without hook
- Use the Hook plugins_loaded

```
// add the class to WP
function my_plugin_start() {
    new my_plugin();

}
add_action( 'plugins_loaded', 'my_plugin_start' );
```

Update: a small live example: Plugin-svn-trunk-page and a pseudo example

```
//avoid direct calls to this file where wp core files not present
if (!function_exists ('add_action')) {
        header('Status: 403 Forbidden');
        header('HTTP/1.1 403 Forbidden');
        exit();
}

if ( !class_exists( 'plugin_class' ) ) {
    class plugin_class {

        function __construct() {
        }

    } // end class

    function plugin_start() {

        new plugin_class();
    }

    add_action( 'plugins_loaded', 'plugin_start' );
} // end class_exists
```

You can also load via mu_plugins_loaded on multisite-install, see the codex for action

You can also load via mu_plugins_loaded on multisite install, see the codex for action reference: http://codex.wordpress.org/Plugin_API/Action_Reference Also here do you see, how inlcude wP with this hook: http://adambrown.info/p/wp_hooks/hook/plugins_loaded?version=2.1&file=wp-settings.php I uses this very often and its not so hard and early, better as an hard new class();

edited Feb 14 '11 at 21:13

community wiki
2 revs
bueltge

3 a small live example: [Plugin-svn-trunk-page]svn.wp-plugins.org/filter-rewrite-rules/trunk/... and a pseudo example `//avoid direct calls to this file where wp core files not present if (!function_exists ('add_action')) { header('Status: 403 Forbidden'); header('HTTP/1.1 403 Forbidden'); exit(); } if (!class_exists('plugin_class')) { class plugin_class { function __construct() { } } // end class function plugin_start() { new plugin_class(); } add_action('plugins_loaded', 'plugin_start'); } // end class_exists` – bueltge Feb 14 '11 at 21:11

2 @Netconstructor.co – i have update the thread, the comment ist ugly for code – bueltge Feb 14 '11 at 21:14

Care about future WordPress & theme versions

Note: After re-reading this advice, I now step back from this practice as checking every function for existence may slow down your site.

Check if functions are deprecated directly in your theme.

This is a "could be like that" example.

```
if ( ! function_exists( 'wp_some_fn' ) )
{
    $theme_data = get_theme_data( get_stylesheet_uri() );
    $error = new WP_Error( 'wp_some_fn', sprintf( __( 'The function %1$s is
deprecated. Please inform the author', TEXTDOMAIN ), "Theme: {$theme_data['Name']}:
Version {$theme_data['Version']}" );

    // abort
    if ( is_wp_error( $error ) )
        return print $error->get_error_message();
}
// else if no error - the function works and exists
wp_some_fn();
```

For proper/best practice error handling see this answer: link

You could drop even the $cause into the function. This will help you and your users to keep on track with functions or classes in your theme that might change.

edited Oct 28 '11 at 10:46

community wiki
5 revs, 2 users 80%
kaiser