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

prev 1 2

Let plugin's folder name be changed

/plugins/pluginname/{various}

The "pluginname" used for the folder should always be changeable.

This is normally handled by defining constants and consistantly using them throughout the plugin.

Needless to say many popular plugins are sinners.

Related:

- `plugins_url()` for easy linking to resources, included with plugin.

edited Jan 10 '11 at 7:39

community wiki
2 revs, 2 users 70%
AndyBeard

Only include files that you need...

If you're in the front end, don't include code that relates to the admin area.

answered Jan 13 '11 at 22:05

community wiki
Denis

Test your plugin

We should definitively have some testing tools on our plugin development environment.

Based on this answer by Ethan Seifert to a testing question, these are good practices to follow:

- Your Unit Testing should test the smallest amount of behavior that a class can perform.
- When you get up to the level of functional testing this is where you can test you code

with Wordpress dependencies.

- Depending on what your plugin does -- consider using Selenium-based tests which test for the presence of data in the DOM by using IDs

answered Feb 16 '11 at 4:44

community wiki
Fernando Briano

1 But covering the smallest first is basic, so that you can reach the functional testing with WordPress as the answer says, isn't that right? – Fernando Briano Feb 17 '11 at 3:55

Provide Help Screens for users

It is nicer to say RTFM (click help) as an answer than having to answer the question time and time again.

```
/**
  * Add contextual help for this screen
  *
  * @param $rtfm
  * @uses get_current_screen
  */
  function ContextualHelp( /*string*/ $rtfm)
  {
     $current_screen = get_current_screen();
     if ($current_screen->id == $this->_pageid)
     {
        $rtfm .= '<h3>The WordPress Plugin - Screen A</h3>';
        $rtfm .= '<p>Here are some tips: donate to me ' .
     }
     return $rtfm;
  }
add_action('contextual_help', array($this,'ContextualHelp'),1,1);
```

**update / note: ** (see comments from kaiser): the above example is to be used in a class

edited Feb 19 '11 at 8:56

community wiki
3 revs
edelwater

Use WordPress (built in) Error handling

Don't just `return;` if some user input was wrong. Deliver them some information about was was done wrong.

```
function some_example_fn( $args = array() )
{
    // If value was not set, build an error message
    if ( ! isset( $args['some_value'] ) )
        $error = new WP_Error( 'some_value', sprintf( __( 'You have forgotten to
specify the %1$s for your function. %2$s Error triggered inside %3$s on line %4$s.',
TEXTDOMAIN ), '$args[\'some_value\']', "\n", __FILE__, __LINE__ ) );

    // die & print error message & code - for admins only!
    if ( isset( $error ) && is_wp_error( $error ) && current_user_can(
'manage_options' ) )
        wp_die( $error->get_error_code(), 'Theme Error: Missing Argument' );

    // Elseif no error was triggered continue...
}
```

One error (object) for all

You can set up a global error object for your theme or plugin during the bootstrap:

```
function bootstrap_the_theme()
{
    global $prefix_error, $prefix_theme_name;
    // Take the theme name as error ID:
    $theme_data = get_theme_data( get_stylesheet_uri() );
    $prefix_theme_name = $theme_data['Name'];
    $prefix_error = new WP_Error( $theme_data['Name'] );
```

```
    include // whatever, etc...
}
add_action( 'after_setup_theme', 'bootstrap_the_theme' );
```

Later you can add unlimited Errors on demand:

```
function some_theme_fn( $args )
{
    global $prefix_error, $prefix_theme_name;
    $theme_data = get_theme_data( get_stylesheet_uri() );
    if ( ! $args['whatever'] && current_user_can( 'manage_options' ) ) // some
required value not set
        $prefix_error->add( $prefix_theme_name, sprintf( 'The function %1$s needs the
argument %2$s set.', __FUNCTION__, '$args[\'whatever\']' ) );

    // continue function...
}
```

Then you can fetch them all at the end of your theme. This way you don't interrupt rendering the page and can still output all your errors for developing

```
function dump_theme_errors()
{
    global $prefix_error, $prefix_theme_name;

    // Not an admin? OR: No error(s)?
    if ( ! current_user_can( 'manage_options' ) ! is_wp_error( $prefix_error ) )
        return;

    $theme_errors = $prefix_error->get_error_messages( $prefix_theme_name );
    echo '<h3>Theme Errors</h3>';
    foreach ( $theme_errors as $error )
        echo "{$error}\n";
}
add_action( 'shutdown', 'dump_theme_errors' );
```

You can find further information at this Q. A related ticket to fix the "working together" of `WP_Error` and `wp_die()` is linked from there and another ticket will follow. Comments, critics & such is appreciated.

edited Oct 28 '11 at 18:01

community wiki
4 revs
kaiser

Use WordPress's Coding Standards

http://codex.wordpress.org/WordPress_Coding_Standards

You know how much easier it is to update code you've worked on vs. code someone else has put together? Coding standards make it easier for any developer working on a project to come in and see what's going on.

We know your plugin or theme is your own, and the way you break your lines and add your curly braces is an expression of your individuality. Every indent is a carefully thought-out statement. But with your custom code, you're contributing to WordPress, even if your code isn't in the core application. Coding standards help developers quickly get up-to-speed with your code.

answered Jun 21 '11 at 16:40

community wiki
gabrielk

Use uninstall, activate and deactivate hooks

There are three different hooks for this:

- Uninstall `register_uninstall_hook();`
- Deactivation `register_deactivation_hook();`
- Activation `register_activation_hook();`

A detailed instruction with a working example can be found here..

answered Oct 28 '11 at 11:24

community wiki

answered Oct 28 '11 at 11:24

community wiki
kaiser