Creating a plug-in file

## Plug-in file structure

unique-plugin-name.php - Primary Plugin PHP file

uninstall.php - The uninstall file

/js - Folder for JavaScript

/css - Folder for CSS

/includes - Folder for additional PHP includes

/images - Folder for plugin images

## Plug-in Header

<?php

/\*

Plugin Name: Your-Plugin-Name

Plugin URI: <http://example.com>

Description: The description

Version: 1.0

Author: Your Name

Author URI: <http://example.com/your-page>

Licence: GPLv2

\*/

?>

See: <http://codex.wordpress.org/Writing_a_Plugin#File_Headers>

NOTE: More information on GPLv2: <http://www.gnu.org/licenses/gpl-2.0.html>

(only required header object is “Plugin Name”)

## Activating and Deactivating Functions

<?php

register\_activation\_hook ( \_\_FILE\_\_, ‘myplugin\_install’ );

function myplugin\_install() {

//do something

}

?>

Helpful to have your app do something at startup.

One helpful thing to do is check version of WP

if ( version\_compare ( $wp\_version, ‘3.5’, ‘<’ ) ) {

wp\_die ( ‘This plugin requires WordPress version 3.5 or higher.’ );

}

NOTE: $wp\_version is a global variable

See: <http://codex.wordpress.org/Global_Variables>

register\_deactivation\_hook ( \_\_FILE\_\_, ‘myplugin\_deactivate()’ );

function myplugin\_deactivate() {

//do something

}

NOTE: deactivation is not uninstalling. You should never include uninstall functionality in your deactivation function. Preserve data until it is deleted.

## Internationalization

Localization is the process of translating the text displayed by the theme or plugin into different languages.

<https://developer.wordpress.org/plugins/>

See: <http://codex.wordpress.org/I18n_for_WordPress_Developers>

<https://developer.wordpress.org/plugins/internationalization/>

## Determining Paths

Never use hard coded paths in a plugin.

## Local Path

<?php echo plugin\_dir\_path( \_\_FILE\_\_ ); ?>

This returns full local path: /public\_html/wp-content/plugins/myplugin/

For subdirectories: <?php echo plugin\_dir\_path( \_\_FILE\_\_ . ‘js/script.js’ ); ?>

This returns: /public\_html/wp-content/plugins/myplugin/js/script.js

## URL Paths

To determine the full URL to any file in your plugin directory, use the plugins\_url() function.

<?php echo ‘<img src=”’ .plugins\_url( ‘images/icon.png’, \_\_FILE\_\_ ) . ‘“>’; ?>

The function accepts two parameters. The first parameter is the path relative to the plugin URL. The second is the plugin file that you want to be relative to. In this case, use the \_\_FILE\_\_ PHP constant. This example will return a full URL.

<img src=”http://example.com/wp-content/plugins/myplugin/images/icon.png”>

Advantages Include:

* Supports the /mu-plugins plugin directory
* Auto detects SSL
* Can detect the location of the plugin even if the user has moved his /wp-content directory to a custom location
* Supports Multisite

Form more location based functions see: <http://codex.wordpress.org/Determining_Plugin_and_Content_Directories>

## Security

*If a plugin contains security holes, it opens up the entire WordPress website.*

WordPress features some built-in security tools that you should always utilize to make your plugins secure.

## 

## 

## Nonces

Nonces, which stands for “number used once,” are used in requests (saving options, form posts, Ajax requests, actions) to stop unauthorized access by generating a secret key.

Example:

<form method=”post”>

<?php wp\_nonce\_field( ‘myplugin\_settings\_form\_save’, ‘prop\_nonce\_field’ ); ?>

Enter your name: <input type=”text” name=”text” /> <br />

<input type=”submit” name=”submit” value=”Save Options” />

</form>

When creating a form nonce, the function wp\_nonce\_field() must be called inside of your <form> tags. There are actually no required parameters for this function to work, but for increased security there are two parameters you should set.

1. $action, which should be a unique string that is descriptive of the action being performed.
2. A unique name for the field, $name. (By default the field name will be \_wpnonce, but you can define a custom unique name in this parameter.

When it is called, it will generate a unique secret key that will be added as a hidden form field and passed with your form data.

After your form is posted, the first thing you need to do is check your nonce secret key using the check\_admin\_referer() function.

Example:

function myplugin\_update\_options() {

if ( isset ( $\_POST[‘submit’] ) ) {

//check nonce for security

check\_admin\_referer( ‘myplugin\_settings\_form\_save’ , ‘myplugin\_nonce\_field’ );

//nonce passed, now do stuff

}

}

Verifying that the nonce is valid is as simple as calling the check\_admin\_referer() function and passing it your unique nonce action and name that you defined earlier. If the nonce secret key does not match the secret key created on your form, WordPress will stop processing the page and issue an error message. This primarily protects it from cross-site request forgery, or CSRF.

Nonces can also be used on links that perform actions. To create a URL nonce, you use the wp\_nonce\_url() function.

Example:

<?php

$link = ‘my-url.php?action=delete&ID=15’;

?>

<a href=”<?php echo wp\_nonce\_url( $link, ‘myplugin\_nonce\_url\_check’ ); ?>”>Delete</a>

the wp\_nonce\_url() function accepts two parameters: the URL to add the nonce to and the unique nonce name you are creating. The preceding code would generate a link that looks like this:

http://example.com/wp-admin/my-url.php?action=delete&ID=15&\_wpnonce-e9d6673015

Verify if the url nonce is correct just as you did with your form - by using the check\_admin\_referer()

Example:

function myplugin\_update\_options() {

if ( isset( $\_GET[‘action’] ) ) {

//check nonce for security

check\_admin\_referer ( ‘myplugin\_nonce\_url\_check’ );

//do stuff

}

}

See: <http://codex.wordpress.org/WordPress_Nonces>

## Data Validation and Sanitization

Any data that comes from somewhere external to your code (such as user input) needs to be scrubbed to verify that it’s free from illegal characters and potentially unsafe data. Data validation is essential to proper plugin security. Improperly validated data can lead to SQL injection hacks, exploits, errors, and much more.

WordPress features a set of escaping functions that you can use to verify that your data is escaped properly when being displayed to the screen. These escaping functions allow a set naming standard, which makes it easy to identify what they are escaping.

* esc\_ : The prefix for the escaping functions.
* attr: The escaping context (attr, html, textarea, js, sql, url, and url\_raw)
* \_e: The optional translation suffix.

Example: esc\_attr\_e()

Example:

<?php esc\_html( $text ); ?>

The esc\_attr() function is used for escaping HTML attributes. This function should be used whenever you need to display data inside an HTML element:

Example:

<input type=”text” name=”first\_name” value=”<?php echo esc\_attr( $text ); ?>”>

A very powerful function for processing and sanitizing untrusted HTML is wp\_kses(). This function is used in WordPress to verify that only allowed HTML tags you can avoid cross-site scripting (XSS) attacks through your code.

Example:

$allowed\_tags = array(

‘strong’ => array(),

‘a’ => aray(

‘href’ => array(),

‘title => array()

)

);

$html = ‘<a href=”#” class=”external”>link</a>.

This is <b>bold</b> and <strong>strong</strong>’;

echo wp\_kses( $html, $allowed\_tags );

The output would be:

<a href=”#”>link</a>. This is bold and <strong>strong</strong>

NOTE: Notice what has been removed what passed.

See: <http://codex.wordpress.org/Data_Validation>

# Hooks: Actions and Filters

Hooks are simply a standardized way of “hooking” into WordPress. Using hooks, you can execute functions at specific times in the WordPress process, allowing you to alter how WordPress functions and the expected output.

A hook is simply a PHP function call with various parameters that can be sent.

EXAMPLE:

<?php add\_action( $tag, $function\_to\_add, $priority, $accepted\_args ); ?>

Two types of hooks can be used: actions and filters.

## Filter Hook

Action hooks are triggered by events in WordPress. For example, an Action hook is triggered when a new post is published.

Filter hooks are used to modify WordPress content before saving it to the database or displaying it to the screen. For example, a Filter hook is available for the content of the post or page.

EXAMPLE

Filter Hook

<?php add\_filter( ‘the\_content’, ‘myplugin\_function’ ); ?>

Function can accept four parameters:

1. filter\_action (string): The filter to use
2. custom\_filter\_function (string): The custom function to pass the filter through
3. priority (integer): The priority in which this filter should run.
4. accepted args (integer): The number of arguments the function accepts

EXAMPLE

<?php

add\_filter( ‘the\_content’, ‘myplugin\_profanity\_filter’ );

function myplugin\_profanity\_filter( $content ) {

$profanities = array( ‘sissy’, ‘dummy’ );

$content = str\_ireplace( $profanities, ‘[censored]’, $content );

return $content;

}

?>

This will replace the words “sissy” and “dummy” with “[censored]” automatically on all posts and pages on your website.

The content isn’t modified in the database; instead, it’s modified during processing of the\_post(), before being displayed, when this filter is invoked. The words remain in the database.

Filter hooks always receive data; in this case, the $content variable is passed t your function and contains your post content. Also notice the last line of your function returns the $content variable.

You must always return the content you are modifying or else it returns empty and therefore displays nothing.

## Action Hook

The Action Hook is triggered by events in WordPress. WordPress doesn’t require any return values from your Action Hook function; the WordPress just notifies your code that a specific event has taken place.

EXAMPLE:

<?php

add\_action( ‘comment\_post’, ‘myplugin\_email\_new\_comment’ );

function myplugin\_email\_new\_comment() {

wp\_mail( ‘[me@example.com](mailto:me@example.com)’, ‘New blog comment’,

‘There is a new comment on your website: <http://example.com>’ );

}

?>

Notice that you are using the comment\_post Action Hook. This action is triggered whenever a new comment is posted in WordPress.

Action hooks don’t require your function to send in or return variables. It’s not required, but it possible.

SEE: <http://codex.wordpress.org/Plugin_API/Filter_Reference>