Codex tools: Log in
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

Interested in functions, hooks, classes, or methods? Check out the new WordPress Code Reference!

Administration Menus

Languages: English • 中文(简体) • Administration Menus 日本語 Русский • (Add your language)

Introduction

Usually, plugin and theme authors need to provide access to a settings (options) screen so users can customize how the plugin or theme is used. The best way to present the user with such a screen is to create an administration menu item that allows the user to access that settings screen from all the Administration Screens. This article explains how plugin authors can add administration menus and screens. Note: the following information assumes a knowledge of the basics of Writing a Plugin and using the Plugin API of Actions and Filters.

Contents

Function Reference

General Functions	WordPress Administration Menus
Menu Pages • add_menu_page() • add_object_page() • add_utility_page() • remove_menu_page() SubMenu Pages • add_submenu_page() • remove_submenu_page()	• add_dashboard_page() • add_posts_page() • add_media_page() • add_pages_page() • add_comments_page() • add_theme_page() • add_plugins_page() • add_users_page() • add_management_page() • add_options_page()

Every Plot Needs a Hook

To add an administration menu, you must do three things:

1. Create a function that contains the menu-building code
2. Register the above function using the admin_menu action hook. (If you are adding an admin menu for the Network, use network_admin_menu instead).

3. Create the HTML output for the page (screen) displayed when the menu item is clicked

It is that second step that is often overlooked by new developers. You cannot simply call the menu code described; you must put it inside a function, and then register the function.

Here is a very simple example of the three steps just described. This plugin will add a sub-level menu item under the Settings top-level menu in Administration Screens, and when selected, that menu item will cause a very basic screen to display. Note: this code should be added to a main plugin PHP file or a separate PHP include file.

```
<?php
/** Step 2 (from text above). */
add_action( 'admin_menu', 'my_plugin_menu' );

/** Step 1. */
function my_plugin_menu() {
        add_options_page( 'My Plugin Options', 'My Plugin', 'manage_options', 'my-unique-identifier',
'my_plugin_options' );
}

/** Step 3. */
function my_plugin_options() {
        if ( !current_user_can( 'manage_options' ) )  {
                wp_die( __( 'You do not have sufficient permissions to access this page.' ) );
        }
        echo '<div class="wrap">';
        echo '<p>Here is where the form would go if I actually had options.</p>';
        echo '</div>';
}
?>
```

In this example, the function `my_plugin_menu()` adds a new item to the Settings administration menu via the `add_options_page()` function. Note: more complicated multiple menu items can be added, but that will be described later. Notice the `add_action()` call in Step 2 that registers the `my_plugin_menu()` function under the admin_menu hook. Without that **`add_action()`** call, a PHP error for undefined function will be thrown when attempting to activate the plugin. Finally, the `add_options_page()` call refers to the `my_plugin_options()` function which contains the actual page to be displayed (and PHP code to be processed) when someone clicks the menu item.

These processes are described in more detail in the sections below. Remember to enclose the creation of the menu and the page in functions, and use the admin_menu hook to get the whole process started at the right time!

Determining Location for New Menus

Before creating a new menu, first decide if the menu should be a top-level menu, or a sub-level menu item. A top-level menu displays as new section in the administration menus and contains sub-level menu items. A sub-level menu means the menu item is a member of an existing menu.

It is rare that a plugin would require the creation of a top-level menu. If the plugin introduces an entirely new concept or feature to WordPress, and needs many screens to do it, then that plugin may warrant a new top-level menu. Adding a top-level menu should only be considered if you really need multiple, related screens to make WordPress do something it was not originally designed to accomplish. Examples of new top-level menus might include job management or conference management. Please note with the native post type registration, WordPress automatically creates top-level menus to manage those features.

If the creation of a top-level menu is not necessary, decide under what top-level menu to place your sub-level menu item. As a point of reference, most plugins add sub-level menu items underneath existing WordPress top-level menus. For example, the Backup plugin adds a sub-level menu option to the Tools top-level menu. Please note with the taxonomy registration, WordPress automatically creates sub-level menus under the applicable top-level menu to manage those features.

Use this guide of the WordPress top-level menus to determine the correct location for your sub-level menu item:

Dashboard
: Information central for your site and include the Updates option for updating WordPress core, plugins, and themes.

Posts
: Displays tools for writing posts (time oriented content).

Media

Uploading and managing your pictures, videos, and audio.

Pages
: Displays tools for writing your static content called pages.

Comments
: Controlling and regulation reader to responses to posts.

Appearance
: Displays controls for manipulation of theme/style files, sidebars, etc.

Plugins
: Displays controls dealing with plugin management, not configuration options for a plugin itself.

Users
: Displays controls for user management.

Tools
: Manage the export, import, and even backup of blog data.

Settings
: Displays plugin options that only administrators should view (also see Creating Settings Pages).

Network Admin
: Displays plugin options that are set on the Network. Instead of "admin_menu", you should use "network_admin_menu" (see also Create A Network)

Admin Menu Functions

Now that you have decided where to add your top-level or sub-level menu, the next step is to tell WordPress about your new pages. All of this will take place in a function registered to the `'admin_menu'` action. A working example is presented at the end of this section.

Top-Level Menus

If you have decided your plugin requires a brand-new top-level menu, the first thing you'll need to do is to create one with the `add_menu_page` function. Note: skip to Sub-Level Menus if you don't need a top-level menu.

```
<?
php add_menu_page( $page_title, $menu_title, $capability, $menu_slug, $function, $icon_url, $position ); ?
>
```

Parameter values:

page_title
: The text to be displayed in the title tags of the page when the menu is selected.

menu_title
: The on-screen name text for the menu.

capability
: The capability required for this menu to be displayed to the user. When using the Settings API to handle your form, you should use 'manage_options' here as the user won't be able to save options without it. User levels are deprecated and should not be used here!

menu_slug
: The slug name to refer to this menu by (should be unique for this menu). Prior to Version 3.0 this was called the file (or handle) parameter. If the function parameter is omitted, the menu_slug should be the PHP file that handles the display of the menu page content.

function

The function that displays the page content for the menu page.

Technically, the function parameter is optional, but if it is not supplied, then WordPress will basically assume that including the PHP file will generate the administration screen, without calling a function. The page-generating code can be written in a function within the main plugin file.

In the event that the function parameter is specified, it is possible to use any string for the menu_slug parameter. This allows usage of pages such as ?page=my_super_plugin_page instead of ?page=my-super-plugin/admin-options.php.

The function must be referenced in one of two ways:

1. if the function is a member of a class within the plugin it should be referenced as `array( $this, 'function_name' )`
2. in all other cases, using the function name itself is sufficient

icon_url

The url to the icon to be used for this menu. This parameter is optional.

position

The position in the menu order this menu should appear. By default, if this parameter is omitted, the menu will appear at the bottom of the menu structure. To see the current menu positions, use `print_r($GLOBALS['menu'])` after the menu has loaded.

Sub-Level Menus

Once your top-level menu is defined, or you have chosen to use an existing WordPress top-level menu, you are ready to define one or more sub-level menu items using the `add_submenu_page` function. Make sure to add the sub-level menu items in the order you want them displayed.

Using add_submenu_page

```
<?php add_submenu_page( $parent_slug, $page_title, $menu_title, $capability, $menu_slug, $function); ?>
```

Parameter values:

parent_slug

The slug name for the parent menu, or the file name of a standard WordPress admin file that supplies the top-level menu in which you want to insert your submenu, or your plugin file if this submenu is going into a custom top-level menu.

Examples:

1. For Dashboard: add_submenu_page('index.php',...)
2. For Posts: add_submenu_page('edit.php',...)
3. For Media: add_submenu_page('upload.php',...)
4. For Pages: add_submenu_page('edit.php?post_type=page',...)
5. For Comments: add_submenu_page('edit-comments.php',...)
6. For Custom Post Types: add_submenu_page('edit.php?post_type=your_post_type',...)
7. For Appearance: add_submenu_page('themes.php',...)
8. For Plugins: add_submenu_page('plugins.php',...)
9. For Users: add_submenu_page('users.php',...)
10. For Tools: add_submenu_page('tools.php',...)
11. For Settings: add_submenu_page('options-general.php',...)

page_title

Text that will go into the HTML page title for the page when the submenu is active.

menu_title

The text to be displayed in the title tags of the page when the menu is selected.

capability

The capability required for this menu to be displayed to the user. User levels are deprecated and should not be used here!

menu_slug

For existing WordPress menus, the PHP file that handles the display of the menu page content. For submenus of a custom top-level menu, a unique identifier for this sub-menu page.

In situations where a plugin is creating its own top-level menu, the first submenu will normally have the same link title as the top-level menu and hence the link will be duplicated. The duplicate link title can be avoided by calling the add_submenu_page function the first time with the parent_slug and menu_slug parameters being given the same value.

function

The function that displays the page content for the menu page.

Technically, as in the `add_menu_page` function, the function parameter is optional, but if it is not supplied, then WordPress will basically assume that including the PHP file will generate the administration screen, without calling a function. Most plugin authors choose to put the page-generating code in a function within their main plugin file.

In the event that the function parameter is specified, it's possible to use any string for the menu_slug parameter. This allows usage of pages such as ?page=my_super_plugin_page instead of ?page=my-super-plugin/admin-options.php.

See the Top-Level Menus for notes about how to reference class members as function parameters.

Example

Here's a quick example illustrating how to insert a top-level menu page and a sub-menu page, where the title on the sub-menu page is different from the top-level page. In this example, 'my_magic_function' is the name of the function that displays the first sub-menu page:

```
<?php
add_menu_page('Page title', 'Top-level menu title', 'manage_options', 'my-top-level-
handle', 'my_magic_function');
add_submenu_page( 'my-top-level-handle', 'Page title', 'Sub-menu title', 'manage_options', 'my-submenu-
handle', 'my_magic_function');
?>
```

Here's an example of adding an option page under a custom post type menu block (see also here):

```
<?php add_submenu_page('edit.php?post_type=wiki', 'Options', 'Options', 'manage_options', 'wiki-
options', array(&$this, 'options_page') ); ?>
```

Using Wrapper Functions

Since most sub-level menus belong under the Settings, Tools, or Appearance menus, WordPress supplies wrapper functions that make adding a sub-level menu items to those top-level menus easier. Note that the function names may not match the names seen in the Administration Screens as they have changed over time:

Dashboard

```
<?php add_dashboard_page( $page_title, $menu_title, $capability, $menu_slug, $function); ?>
```

Posts

```
<?php add_posts_page( $page_title, $menu_title, $capability, $menu_slug, $function); ?>
```

Media

```
<?php add_media_page( $page_title, $menu_title, $capability, $menu_slug, $function); ?>
```

Pages

```
<?php add_pages_page( $page_title, $menu_title, $capability, $menu_slug, $function); ?>
```

Comments

```
<?php add_comments_page( $page_title, $menu_title, $capability, $menu_slug, $function); ?>
```

Appearance

```
<?php add_theme_page( $page_title, $menu_title, $capability, $menu_slug, $function); ?>
```

Plugins

```
<?php add_plugins_page( $page_title, $menu_title, $capability, $menu_slug, $function); ?>
```

Users

```
<?php add_users_page( $page_title, $menu_title, $capability, $menu_slug, $function); ?>
```

Tools

```
<?php add_management_page( $page_title, $menu_title, $capability, $menu_slug, $function); ?>
```

Settings

```
<?php add_options_page( $page_title, $menu_title, $capability, $menu_slug, $function); ?>
```

Also see Creating Options Pages for more on this.

Inserting the Pages

Here is an example of a WordPress plugin that inserts new menus into various places:

```
<?php
/*
Plugin Name: Menu Test
Plugin URI: http://codex.wordpress.org/Adding_Administration_Menus
Description: Menu Test
Author: Codex authors
Author URI: http://example.com
*/

// Hook for adding admin menus
add_action('admin_menu', 'mt_add_pages');

// action function for above hook
function mt_add_pages() {
    // Add a new submenu under Settings:
    add_options_page(__('Test Settings','menu-test'), __('Test Settings','menu-test'),
'manage_options', 'testsettings', 'mt_settings_page');

    // Add a new submenu under Tools:
    add_management_page( __('Test Tools','menu-test'), __('Test Tools','menu-test'), 'manage_options',
'testtools', 'mt_tools_page');

    // Add a new top-level menu (ill-advised):
    add_menu_page(__('Test Toplevel','menu-test'), __('Test Toplevel','menu-test'), 'manage_options',
'mt-top-level-handle', 'mt_toplevel_page' );

    // Add a submenu to the custom top-level menu:
    add_submenu_page('mt-top-level-handle', __('Test Sublevel','menu-test'), __('Test Sublevel','menu-
test'), 'manage_options', 'sub-page', 'mt_sublevel_page');

    // Add a second submenu to the custom top-level menu:
    add_submenu_page('mt-top-level-handle', __('Test Sublevel 2','menu-test'), __('Test Sublevel
2','menu-test'), 'manage_options', 'sub-page2', 'mt_sublevel_page2');
}

// mt_settings_page() displays the page content for the Test Settings submenu
function mt_settings_page() {
    echo "<h2>" . __( 'Test Settings', 'menu-test' ) . "</h2>";
}

// mt_tools_page() displays the page content for the Test Tools submenu
```

```
function mt_tools_page() {
    echo "<h2>" . __( 'Test Tools', 'menu-test' ) . "</h2>";
}

// mt_toplevel_page() displays the page content for the custom Test Toplevel menu
function mt_toplevel_page() {
    echo "<h2>" . __( 'Test Toplevel', 'menu-test' ) . "</h2>";
}

// mt_sublevel_page() displays the page content for the first submenu
// of the custom Test Toplevel menu
function mt_sublevel_page() {
    echo "<h2>" . __( 'Test Sublevel', 'menu-test' ) . "</h2>";
}

// mt_sublevel_page2() displays the page content for the second submenu
// of the custom Test Toplevel menu
function mt_sublevel_page2() {
    echo "<h2>" . __( 'Test Sublevel2', 'menu-test' ) . "</h2>";
}

?>
```

Sample Menu Page

Note: See the Settings API for information on creating settings pages.

The example above contains several dummy functions, such as `mt_settings_page`, as placeholders for actual page content. We need to turn them into real menu pages. So, let's assume that our plugin has an option called mt_favorite_color, and that we want to allow the site owner to type in his/her favorite color via a Settings page. The `mt_options_page` function will need to put a data entry form on the screen to enable this, and also process the entered data. Here is a function that does this:

```
// mt_settings_page() displays the page content for the Test Settings submenu
function mt_settings_page() {

    //must check that the user has the required capability
    if (!current_user_can('manage_options'))
    {
      wp_die( __('You do not have sufficient permissions to access this page.') );
    }

    // variables for the field and option names
    $opt_name = 'mt_favorite_color';
    $hidden_field_name = 'mt_submit_hidden';
    $data_field_name = 'mt_favorite_color';

    // Read in existing option value from database
    $opt_val = get_option( $opt_name );

    // See if the user has posted us some information
    // If they did, this hidden field will be set to 'Y'
    if( isset($_POST[ $hidden_field_name ]) && $_POST[ $hidden_field_name ] == 'Y' ) {
        // Read their posted value
        $opt_val = $_POST[ $data_field_name ];

        // Save the posted value in the database
        update_option( $opt_name, $opt_val );

        // Put a "settings saved" message on the screen

?>
<div class="updated"><p><strong><?php _e('settings saved.', 'menu-test' ); ?></strong></p></div>
<?php

    }

    // Now display the settings editing screen

    echo '<div class="wrap">';

    // header

    echo "<h2>" . __( 'Menu Test Plugin Settings', 'menu-test' ) . "</h2>";

    // settings form

    ?>

<form name="form1" method="post" action="">
<input type="hidden" name="<?php echo $hidden_field_name; ?>" value="Y">

<p><?php _e("Favorite Color:", 'menu-test' ); ?>
<input type="text" name="<?php echo $data_field_name; ?>" value="<?php echo $opt_val; ?>" size="20">
</p><hr />

<p class="submit">
<input type="submit" name="Submit" class="button-primary" value="<?php esc_attr_e('Save Changes') ?>"
```

```
/>
</p>

</form>
</div>

<?php

}
```

A few notes:

- The WordPress functions such as `add_menu_page` and `add_submenu_page` take a capability which will be used to determine whether the top-level or sub-level menu is displayed.
- The function which is hooked in to handle the output of the page must check that the user has the required capability as well.
- The WordPress administration functions take care of validating the user login, so you don't have to worry about it in your function.
- The function example above has been internationalized -- see the Internationalization section of Writing a Plugin for more information.
- The function processes any entered data before putting the data entry form on the screen, so that the new values will be shown in the form (rather than the values from the database).
- You don't have to worry about this working the first time, because the WordPress `update_option` function will automatically add an option to the database if it doesn't already exist.
- These admin-menu-adding procedures are parsed every single time you navigate to a page in Admin. So if you are writing a plugin which has no options page, but add one later, you can just add it using the instructions above and re-upload, and tweak until you're happy with it. In other words, menus are not "permanently added" or put into a database upon activating a plugin. They're parsed on the fly, so you can add or subtract menu items at will, re-upload, and the change will be reflected right away.

Page Hook Suffix

Every function that adds a new administration menu (`add_menu_page()`, `add_submenu_page()` and its specialized versions such as `add_options_page()`) returns a special value called Page Hook Suffix. It can be used later as a hook to which an action called only on that particular page can be registered.

One such action hook is load-{page_hook}, where {page_hook} is the value returned by one of these `add_*_page()` functions. This hook is called when the particular page is loaded. In the example below, it is used to display the "Plugin is not configured" notice on all admin pages except for plugin's options page:

```
<?php
add_action('admin_menu', 'my_plugin_menu');

// Here you can check if plugin is configured (e.g. check if some option is set). If not, add new hook.
// In this example hook is always added.
add_action( 'admin_notices', 'my_plugin_admin_notices' );

function my_plugin_menu() {
        // Add the new admin menu and page and save the returned hook suffix
        $hook_suffix = add_options_page('My Plugin Options', 'My Plugin', 'manage_options', 'my-unique-
identifier', 'my_plugin_options');
        // Use the hook suffix to compose the hook and register an action executed when plugin's
options page is loaded
        add_action( 'load-' . $hook_suffix , 'my_load_function' );
}

function my_load_function() {
        // Current admin page is the options page for our plugin, so do not display the notice
        // (remove the action responsible for this)
        remove_action( 'admin_notices', 'my_plugin_admin_notices' );
}

function my_plugin_admin_notices() {
        echo "<div id='notice' class='updated fade'><p>My Plugin is not configured yet. Please do it
now.</p></div>\n";
}

function my_plugin_options() {
        if (!current_user_can('manage_options'))  {
                wp_die( __('You do not have sufficient permissions to access this page.') );
        }
        echo '<div class="wrap">';
        echo '<p>Here is where the form would go if I actually had options.</p>';
        echo '</div>';
```

```
}
?>
```

Resources

- Admin Menu Editor Plugin - allows adding, deleting, hiding, and re-ordering of administration items

Categories:

- Plugins
- Advanced Topics
- WordPress Development