Administration Menus

- Function Reference
  - Menu Pages
  - Sub-menu Pages
  - WordPress Administration Menus
- Every Plot Needs a Hook
- Determining Location for New Menus
  - Top-Level Menus
  - Sub-Level Menus
  - Using Wrapper Functions
    - Dashboard
    - Posts
    - Media
    - Links
    - Pages
    - Comments
    - Appearance
    - Plugins
    - Users
    - Tools
    - Settings
  - Example
- Inserting the Pages
- Sample Menu Page
- Page Hook Suffix
- Resources

This is no longer the recommended way to work with theme options. The Customizer API is the recommended way to give more control and flexibility to your users

Theme authors might need to provide a settings screen, so users can customize how their theme is used or works. The best way to do this is to create an administration menu item that allows the user to access that settings screen from all the administration screens.

Function Reference

Menu Pages

add_menu_page()

add_object_page()

add_utility_page()

remove_menu_page()

Sub-menu Pages

add_submenu_page()

remove_submenu_page()

WordPress Administration Menus

add_dashboard_page()

add_posts_page()

add_media_page()

add_links_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, there are three things you need to do:

1. Create a function that contains the menu-building code.
2. Register the above function using the `admin_menu` action hook – or `network_admin_menu`, if you are adding a menu for the Network.
3. Create the HTML output for the screen, displayed when the menu item is clicked.

Most developers overlook the second step in this list. You can't simply call the menu code. You need to put it **inside a function**, and then register this function.

Here is a simple example of these three steps described. This will add a sub-level menu item under the Settings top-level menu. When selected, that menu item will show a very basic screen.

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

/** Step 1. */
function my_menu() {
	add_options_page(
		'My Options',
		'My Menu',
		'manage_options',
		'my-unique-identifier',
		'my_options'
	);
}

/** Step 3. */
function my_options() {
	if ( ! current_user_can( 'manage_options' ) ) {
		wp_die( __( 'You do not have sufficient permissions to access this page.' ) );
	}
	echo 'Here is where I output the HTML for my screen.';
	echo '</div><pre>';
}
```

In this example, the function `my_menu()` adds a new item to the Settings administration menu via the `add_options_page()` function.

Note: Note that the add_action() call in Step 2 *registers* the my_menu() function under the admin_menu hook. **Without that, add_action() call, a PHP error for undefined function will be thrown**. Finally, the `add_options_page()` call refers to the `my_options()` function which contains the actual page to be displayed (and PHP code to be processed) when someone clicks the menu item.

These steps 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. This means a sub-level menu item is a member of an existing top-level menu.

It is rare that a theme would require the creation of a new top-level menu. If the theme introduces an entirely new concept to WordPress and needs many screens to do it, then that theme 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 that, with the native post type registration, WordPress automatically creates top-level menus to manage this kind of features.

If the creation of a top-level menu is not necessary, you need to decide under what top-level menu to place your new sub-level menu item. As a point of reference, several themes add sub-level menu items underneath existing WordPress top-level menus.

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.
- Links – manage references to other blogs and sites of interest.
- 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.
- 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)

Top-Level Menus

If you have decided your theme 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.

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.
- `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.

Sub-Level Menus

If you want your new menu item to be a sub-menu item, you can create it using the `add_submenu_page()` function instead.

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 sub-menu, or your plugin file if this sub-menu is going into a custom top-level menu. Examples:
  - Dashboard – `add_submenu_page('index.php', ...)`
  - Posts – `add_submenu_page('edit.php', ...)`
  - Media – `add_submenu_page('upload.php', ...)`
  - Links – `add_submenu_page('link-manager.php', ...)`
  - Pages – `add_submenu_page('edit.php?post_type=page', ...)`
  - Comments – `add_submenu_page('edit-comments.php', ...)`
  - Custom Post Types – `add_submenu_page('edit.php?post_type=your_post_type', ...)`
  - Appearance – `add_submenu_page('themes.php', ...)`
  - Plugins – `add_submenu_page('plugins.php', ...)`
  - Users – `add_submenu_page('users.php', ...)`
  - Tools – `add_submenu_page('tools.php', ...)`
  - 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 sub-menus of a custom top-level menu, a unique identifier for this sub-menu page.
- `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.

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 these top-level menus easier. Note that the function names may not match the names seen in the admin UI 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
);
```

Links

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

Pages

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

Comments

```
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 Theme Options for the currently recommended way of creating options via the Customizer API.

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, `register_my_theme_more_settings_menu` is the name of the function that displays the first sub-menu page:

```
<?php
function register_my_theme_settings_menu() {
□add_menu_page(
□□"My Theme's Settings",
□□'My Theme',
□□'manage_options',
□□'my-theme-settings-menu'
□);
}

function register_my_theme_more_settings_menu() {
□add_submenu_page(
□□'my-theme-settings-menu',
□□'More Settings for My Theme',
□□'More Settings',
□□'manage_options',
□□'my-theme-more-settings-menu'
□);
}

add_action( 'admin_menu', 'register_my_theme_settings_menu' );
add_action( 'admin_menu', 'register_my_theme_more_settings_menu' );
```

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

CODE

Inserting the Pages

Here is an example of how to insert multiple menus into various places:

```
<?php
// Hook for adding admin menus
add_action( 'admin_menu', 'mt_add_pages' );
```

```
// Action function for hook above
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 Top-level', '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 Sub-Level', 'menu-test' ), __( 'Test Sub-
Level', '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 Sub-Level 2', 'menu-test' ), __( 'Test Sub-
Level 2', 'menu-test' ), 'manage_options', 'sub-page2', 'mt_sublevel_page2' );
}

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

// mt_tools_page() displays the page content for the Test Tools sub-menu.
function mt_tools_page() {
	echo '</pre><h2>' . __( 'Test Tools', 'menu-test' ) . '</h2><pre>';
}

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

```
// mt_sublevel_page() displays the page content for the first sub-menu of the custom Test
Toplevel menu.
function mt_sublevel_page() {
	echo '</pre><h2>' . __( 'Test Sub-Level', 'menu-test' ) . '</h2><pre>';
}

// mt_sublevel_page2() displays the page content for the second sub-menu of the custom Test
Top-Level menu.
function mt_sublevel_page2() {
	echo '</pre><h2>' . __( 'Test Sub-Level 2', 'menu-test' ) . '</h2><pre>';
}
```

Sample Menu Page

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

The previous example contains several dummy functions, such as `mt_settings_page()`, as placeholders for actual page content. Let's expand on them. What if you wanted to create an option called `mt_favorite_color` that allows the site owner to type in their favorite color via a Settings page? The `mt_options_page()` function will need to output a data entry form on the screen, as well as to process the entered data.

Here is a function that does this:

```
<?php
// mt_settings_page() displays the page content for the Test settings sub-menu.
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 updated" message on the screen
        ?>
        <div class="updated"></div><!-- .updated -->

        <div class="wrap">
            <?php echo '<h2>' . __( 'Menu Test Settings', 'menu-test' ) . '</h2>'; ?>
            <form action="" method="post" name="form1"></form>
            <?php _e( 'Favorite Color:', 'menu-test' ); ?>
            <hr />
        </div><!-- .wrap -->

        <?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 I18n for WordPress Developers for more details.
- 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 theme 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 theme. 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 "Theme is not configured" notice on all admin pages except for plugin's options page:

```
<?php
add_action( 'admin_menu', 'my_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_admin_notices' );

function my_menu() {
	// Add the new admin menu and page and save the returned hook suffix
	$hook_suffix = add_options_page( 'My Options', 'My Theme', 'manage_options', 'my-unique-
identifier', 'my_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_admin_notices' );
}

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

function my_options() {
if ( ! current_user_can( 'manage_options' ) ) {
wp_die( __( 'You do not have sufficient permissions to access this page.' ) );
}

echo '</pre><div class="wrap">';
echo 'Here is where the form would go if I actually had options.';
echo '</div><pre>';
}
```

Resources

[Top Level Menu discussion on wp-hackersAdmin Menu Editor Plugin]

□□□ #2
□ □□□ □□□ 2026-08-29 15:36:32 UTC
□ □□□ □□□ 2026-08-29 15:36:32 UTC