Creating a Basic MainWP Extension

- Quick Start for Experienced Developers
- Prerequisites
- Step 1: Plan Your Extension
- Step 2: Start with the Development Extension
- Step 3: Customize the Plugin Information
- Step 4: Update Namespace and Class Names
- Step 5: Register Your Extension with MainWP
- Step 6: Implement Core Functionality
- Step 7: Create Admin Pages
- Step 8: Interact with Child Sites
- Step 9: Test Your Extension
- Step 10: Package Your Extension
- Common Integration Points
  - Actions
  - Filters
- Example: A Simple "Hello World" Extension
- Next Steps
- Related Resources

This guide walks you through the process of creating a basic MainWP extension. By the end, you'll have a functional extension that integrates with the MainWP Dashboard.

Quick Start for Experienced Developers

```
// 1. Clone the development extension
// git clone https://github.com/mainwp/mainwp-development-extension.git my-extension

// 2. Rename files and update plugin header
// mainwp-development-extension.php → my-extension.php

// 3. Update namespace and class names
namespace MyCompany\MyExtension;

// 4. Register your extension with MainWP
add_filter('mainwp_getextensions', function($extensions) {
    $extensions[] = array(
        'plugin' => __FILE__,
        'api' => 'MyExtension',
        'mainwp' => true,
        'callback' => array(__NAMESPACE__ . '\\MyExtension', 'get_instance'),
        'name' => 'My Extension'
    );
    return $extensions;
});

// 5. Implement core functionality in your main class
class MyExtension {
    private static $instance = null;

    public static function get_instance() {
        if (null === self::$instance) {
            self::$instance = new self();
        }
        return self::$instance;
    }

    public function __construct() {
        // Hook into MainWP actions and filters
        add_action('mainwp_pageheader_extensions', [$this, 'page_header']);
        add_action('mainwp_pagefooter_extensions', [$this, 'page_footer']);
    }

    // Implement required methods...
```

```
}
```

Prerequisites

Before you begin creating a MainWP extension, ensure you have:

- A development environment with WordPress and MainWP Dashboard installed (see Setting Up a Development Environment)
- Basic understanding of WordPress plugin development
- Familiarity with PHP object-oriented programming
- A code editor (like VS Code, PhpStorm, or Sublime Text)

Step 1: Plan Your Extension

Before writing any code, clearly define what your extension will do:

1. **Purpose**: What problem will your extension solve for MainWP users?
2. **Features**: What specific functionality will it provide?
3. **Integration Points**: How will it interact with the MainWP Dashboard and child sites?
4. **User Interface**: What admin pages, settings, or widgets will it need?

Having a clear plan will guide your development process and help you create a more focused, useful extension.

Step 2: Start with the Development Extension

MainWP provides a development extension template that includes the basic structure and boilerplate code you need to get started quickly.

1. Clone or download the MainWP Development Extension from GitHub:

   ```
   git clone https://github.com/mainwp/mainwp-development-extension.git my-extension
   ```

2. Navigate to your new extension directory:

   ```
   cd my-extension
   ```

The development extension includes a well-structured codebase with classes for different aspects of extension functionality, making it easier to understand and extend.

Step 3: Customize the Plugin Information

1. Rename the main plugin file from `mainwp-development-extension.php` to match your extension name (e.g., `my-extension.php`).
2. Update the plugin header information in the main file:

```
/**
 * Plugin Name: My Extension for MainWP
 * Plugin URI: https://yourwebsite.com/my-extension
 * Description: A brief description of what your extension does.
 * Version: 1.0
 * Author: Your Name
 * Author URI: https://yourwebsite.com
 * Text Domain: my-extension
 */
```

3. Update the text domain throughout the plugin to match your extension's slug.

Step 4: Update Namespace and Class Names

1. Change the namespace in all PHP files from `MainWP\Dev` to your own namespace (e.g., `MyCompany\MyExtension`).
2. Rename the main class and other classes to match your extension name.
3. Update all references to these classes throughout the codebase.

For example:

```
namespace MyCompany\MyExtension;


class MyExtension {
```

```
    // Class implementation
}
```

Step 5: Register Your Extension with MainWP

MainWP uses a filter to discover and load extensions. Update the `mainwp_getextensions` filter in your main plugin file:

```
namespace MyCompany\MyExtension;

add_filter('mainwp_getextensions', function($extensions) {
    $extensions[] = array(
        'plugin' => __FILE__,
        'api' => 'MyExtension',
        'mainwp' => true,
        'callback' => array(__NAMESPACE__ . '\\MyExtension', 'get_instance'),
        'name' => 'My Extension'
    );
    return $extensions;
});
```

This tells MainWP:

- The location of your plugin file
- The API identifier for your extension
- That this is a MainWP extension (not a third-party integration)
- The callback function to initialize your extension
- The display name of your extension

Step 6: Implement Core Functionality

The development extension includes several classes that handle different aspects of extension functionality:

1. **Main Class** (`class/class-mainwp-development-extension.php`): The core class that initializes your extension and hooks into MainWP.
2. **Admin Class** (`class/class-mainwp-development-extension-admin.php`): Handles admin pages, settings, and UI elements.
3. **Database Class** (`class/class-mainwp-development-extension-db.php`): Manages database operations for your extension.
4. **Utility Class** (`class/class-mainwp-development-extension-utility.php`): Contains helper functions and utilities.
5. **AJAX Class** (`class/class-mainwp-development-extension-ajax.php`): Handles AJAX requests for your extension.

Modify these classes to implement your extension's specific functionality. The development extension includes examples and comments to guide you.

Step 7: Create Admin Pages

Most MainWP extensions provide admin pages for configuration and displaying information. The development extension includes examples of how to create these pages.

1. Register your admin pages in the Admin class:

```
public function init_menu() {
    add_submenu_page(
        'mainwp_tab',
        __('My Extension', 'my-extension'),
        __('My Extension', 'my-extension'),
        'read',
        'MyExtension',
        array($this, 'render_admin_page')
    );
}
```

2. Create the page rendering function:

```
public function render_admin_page() {
    // Check if user has permissions
    if (!mainwp_current_user_can('read')) {
        mainwp_do_not_have_permissions(__('My Extension', 'my-extension'));
        return;
    }
```

```
    // Render your admin page content
    ?>
    <div class="ui segment">
        <h2 class="ui header"><?php esc_html_e('My Extension', 'my-extension'); ?></h2>
        <!-- Your admin page content here -->
    </div>
    <?php
}
```

Step 8: Interact with Child Sites

If your extension needs to interact with child sites, you can use MainWP's communication system:

1. **Dashboard to Child**: Send requests from the dashboard to child sites.

```
// Example of sending a request to a child site
use MainWP\Dashboard\MainWP_DB;
use MainWP\Dashboard\MainWP_Connect;

$data = array('action' => 'my_custom_action', 'param' => 'value');
$website = MainWP_DB::instance()->get_website_by_id($website_id);
$information = MainWP_Connect::fetch_url_authed($website, 'my_extension', $data);
```

2. **Child to Dashboard**: Process requests on the child site.

You'll need to create a separate child plugin or use the MainWP Child plugin's hooks to handle these requests.

Step 9: Test Your Extension

Thoroughly test your extension to ensure it works correctly:

1. Install and activate it on a test MainWP Dashboard.
2. Test all features and functionality.
3. Check for any errors or warnings in the browser console and PHP error logs.
4. Test with multiple child sites to ensure compatibility.

Step 10: Package Your Extension

When your extension is ready for distribution:

1. Remove any development or debugging code.
2. Ensure all text is properly internationalized.
3. Create a readme.txt file with installation and usage instructions.
4. Package your extension as a ZIP file for distribution.

Common Integration Points

Here are some common MainWP hooks you might use in your extension:

Actions

- `mainwp_pageheader_extensions`: Add content to the page header in extension pages
- `mainwp_pagefooter_extensions`: Add content to the page footer in extension pages
- `mainwp_after_header`: Add content after the MainWP header
- `mainwp_before_footer`: Add content before the MainWP footer
- `mainwp_site_synced`: Triggered when a site is synced

Filters

- `mainwp_getextensions`: Register your extension with MainWP
- `mainwp_extension_enabled_check`: Check if your extension is enabled
- `mainwp_extension_available_check`: Check if your extension is available

For a complete list of available hooks, see the MainWP Hooks Reference.

Example: A Simple "Hello World" Extension

Here's a minimal example of a MainWP extension that adds a new admin page:

```php
<?php
/**
 * Plugin Name: Hello World for MainWP
 * Plugin URI: https://example.com/hello-world
 * Description: A simple Hello World extension for MainWP.
 * Version: 1.0
 * Author: Your Name
 * Author URI: https://example.com
 * Text Domain: hello-world-mainwp
 */

namespace MyCompany\HelloWorld;

// Register the extension with MainWP
add_filter('mainwp_getextensions', function($extensions) {
    $extensions[] = array(
        'plugin' => __FILE__,
        'api' => 'HelloWorld',
        'mainwp' => true,
        'callback' => array(__NAMESPACE__ . '\\HelloWorld', 'get_instance'),
        'name' => 'Hello World'
    );
    return $extensions;
});

class HelloWorld {
    private static $instance = null;

    public static function get_instance() {
        if (null === self::$instance) {
            self::$instance = new self();
        }
        return self::$instance;
    }

    public function __construct() {
        add_action('admin_init', [$this, 'admin_init']);
        add_action('mainwp_admin_menu', [$this, 'init_menu']);
    }
```

```
    public function admin_init() {
        // Initialize admin functionality
    }

    public function init_menu() {
        // Add submenu page to MainWP menu
        add_submenu_page(
            'mainwp_tab',
            __('Hello World', 'hello-world-mainwp'),
            __('Hello World', 'hello-world-mainwp'),
            'read',
            'HelloWorld',
            [$this, 'render_admin_page']
        );
    }

    public function render_admin_page() {
        // Check if user has permissions
        if (!mainwp_current_user_can('read')) {
            mainwp_do_not_have_permissions(__('Hello World', 'hello-world-mainwp'));
            return;
        }

        // Render admin page
        ?>
        <div class="ui segment">
            <h2 class="ui header"><?php esc_html_e('Hello World Extension', 'hello-world-
mainwp'); ?></h2>
            <p><?php esc_html_e('Welcome to your first MainWP extension!', 'hello-world-
mainwp'); ?></p>
        </div>
        <?php
    }
}
```

Next Steps

After creating your basic extension, you might want to explore:

- Using MainWP Actions & Filters for deeper integration
- Building Admin Interfaces for more advanced UI elements
- Data Storage and Retrieval for managing extension data
- Debugging Extensions for troubleshooting issues

Related Resources

- MainWP Development Extension on GitHub
- MainWP Dashboard API Documentation
- MainWP Dashboard Hooks

□□□ #2
□ □□□ □□□ 2026-08-29 22:03:44 UTC
□ □□□ □□□ 2026-08-29 22:03:44 UTC