ScatterBytes - Code Injector Plugin Documentation

Table of Contents

User Guide: How to Use ScatterBytes

Overview

ScatterBytes is a WordPress plugin that allows you to inject custom JavaScript and CSS code into specific pages or globally across your entire website. It's perfect for adding tracking codes, custom styling, analytics scripts, and other page-specific functionality without editing theme files.

Getting Started

Installation

1. Upload the plugin file to your WordPress /wp-content/plugins/ directory
2. Activate the plugin through the 'Plugins' menu in WordPress
3. Navigate to **Settings > ScatterBytes** in your WordPress admin panel

Main Features

1. Global Code Injection

Add code that runs on **every page** of your website.

How to Add Global Code:

1. Go to the "Global Code Injection" section
2. Enter a descriptive **Script Name** (e.g., "Google Analytics Tracking")
3. Select **Code Type**:
    - 📜 **JavaScript** - For tracking codes, analytics, custom scripts
    - 🎨 **CSS** - For global styling, fonts, layout tweaks

4. Enter your **Code Content** with proper tags:
5. <!-- For JavaScript --><script> console.log('Global script loaded!');</script><!-- For CSS --><style> body { font-family: Arial, sans-serif; }</style>
6. Choose **Position**:
    - **Header** - Loads early, good for CSS and critical scripts
    - **Footer** - Loads after content, good for analytics and non-critical scripts
7. Click "🌐 **Save Global Code**"

2. Page-Specific Code Rules

Add code that only runs on selected pages or posts.

How to Create a Rule:

1. Go to the "Add New Code Rule" section
2. Enter a **Rule Name** (e.g., "Contact Form Styling")
3. Select **Code Type** (JavaScript or CSS)
4. Enter your **Code Content**
5. Select **Target Pages** from the checkbox list:
    - 🏠 **Home Page** - Your site's front page
    - 📄 **Pages** - Individual WordPress pages
    - 📝 **Posts** - Individual blog posts
6. Click "💾 **Save Code Rule**"

Managing Your Code

Viewing Active Code

Both global and page-specific code are displayed in tables showing:

- **Name** - Your descriptive title
- **Type** - JavaScript or CSS badge
- **Position** (Global only) - Header or Footer
- **Target Pages** (Rules only) - Which pages it affects
- **Status** - Active (green ✓) or Inactive (red ✗)

Managing Code

For each code entry, you can:

- **Activate/Deactivate** - Toggle without deleting
- **Delete** - Permanently remove the code

Common Use Cases

Google Analytics

```
<script async src="https://www.googletagmanager.com/gtag/js?id=GA_MEASUREMENT_ID"></script>
<script>
  window.dataLayer = window.dataLayer || [];
  function gtag(){dataLayer.push(arguments);}
  gtag('js', new Date());
  gtag('config', 'GA_MEASUREMENT_ID');
</script>
```

Custom CSS for Contact Page

```
<style>
  .contact-form {
    background: #f9f9f9;
    padding: 20px;
    border-radius: 8px;
  }
  .submit-button {
    background: #667eea;
    color: white;
  }
</style>
```

Facebook Pixel

```
<script>
  !function(f,b,e,v,n,t,s)
  {if(f.fbq)return;n=f.fbq=function(){n.callMethod?
  n.callMethod.apply(n,arguments):n.queue.push(arguments)};
  if(!f._fbq)f._fbq=n;n.push=n;n.loaded=!0;n.version='2.0';
```

```
n.queue=[];t=b.createElement(e);t.async=!0;
t.src=v;s=b.getElementsByTagName(e)[0];
s.parentNode.insertBefore(t,s)}(window, document,'script',
'https://connect.facebook.net/en_US/fbevents.js');
fbq('init', 'YOUR_PIXEL_ID');
fbq('track', 'PageView');
</script>
```

Best Practices

Security

- Only add code from trusted sources
- Test code on a staging site first
- Keep backups of your code snippets

Performance

- Use **Footer** position for non-critical JavaScript
- Use **Header** position for CSS and critical scripts
- Minimize the amount of code when possible

Organization

- Use descriptive names for your rules
- Group related functionality together
- Regularly review and clean up unused code

Developer Guide: How It Works with WordPress

Architecture Overview

ScatterBytes is built as a WordPress plugin using object-oriented PHP. It integrates with WordPress through hooks, database interactions, and the admin interface system.

Class Structure

Main Class: ScatterBytesInjector

The plugin is encapsulated in a single class that handles all functionality:

```
class ScatterBytesInjector {
```

```
    private $table_name;          // Page-specific rules table
    private $global_table_name;   // Global code table

    public function __construct() {
        // Initialize database table names
        // Register WordPress hooks
        // Set up AJAX handlers
    }
}
```

Database Schema

Page-Specific Rules Table: wp_scatterbytes_page_code

```
CREATE TABLE wp_scatterbytes_page_code (
    id mediumint(9) NOT NULL AUTO_INCREMENT,
    rule_name varchar(255) NOT NULL,
    code_content longtext NOT NULL,
    code_type varchar(20) NOT NULL DEFAULT 'javascript',
    target_pages longtext NOT NULL,
    is_active tinyint(1) DEFAULT 1,
    created_at datetime DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY (id)
);
```

Global Code Table: wp_scatterbytes_global_code

```
CREATE TABLE wp_scatterbytes_global_code (
    id mediumint(9) NOT NULL AUTO_INCREMENT,
    script_name varchar(255) NOT NULL,
    code_content longtext NOT NULL,
    code_type varchar(20) NOT NULL DEFAULT 'javascript',
    position varchar(20) NOT NULL DEFAULT 'header',
    is_active tinyint(1) DEFAULT 1,
```

```
    created_at datetime DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY (id)
);
```

WordPress Integration Points

1. Plugin Activation Hooks

```
register_activation_hook(__FILE__, array($this, 'create_table'));
register_activation_hook(__FILE__, array($this, 'create_global_table'));
```

- Creates database tables when plugin is activated
- Uses dbDelta() for safe table creation/updates

2. Admin Interface Integration

```
add_action('admin_menu', array($this, 'add_admin_menu'));
```

- Adds plugin page under **Settings** menu
- Uses add_options_page() for proper WordPress admin integration

3. Code Injection Hooks

```
// Global header scripts (priority 5 - early loading)
add_action('wp_head', array($this, 'inject_global_header'), 5);

// Page-specific scripts (priority 10 - default, after global)
add_action('wp_head', array($this, 'inject_code'), 10);

// Global footer scripts (priority 5 - early loading)
add_action('wp_footer', array($this, 'inject_global_footer'), 5);
```

Hook Priority Strategy:

- Global scripts load first (priority 5)
- Page-specific scripts load after (priority 10)
- Ensures proper loading order and dependency management

4. AJAX Integration

```
add_action('wp_ajax_save_code_rule', array($this, 'save_code_rule'));
add_action('wp_ajax_delete_code_rule', array($this, 'delete_code_rule'));
```

```
add_action('wp_ajax_save_global_code', array($this, 'save_global_code'));
```

- Handles form submissions without page reloads
- Uses WordPress nonce system for security

Security Implementation

1. Nonce Protection

```
wp_verify_nonce($_POST['nonce'], 'scatterbytes_nonce')
```

- All AJAX requests protected against CSRF attacks
- Nonces generated with wp_create_nonce()

2. Capability Checks

```
if (!current_user_can('manage_options')) {
   wp_die('Insufficient permissions');
}
```

- Only administrators can manage code injection
- Uses WordPress capability system

3. Data Sanitization

```
'rule_name' => sanitize_text_field($_POST['rule_name']),
'code_content' => wp_unslash($_POST['code_content']),
'code_type' => sanitize_text_field($_POST['code_type']),
```

- Input sanitization for safe database storage
- Special handling for code content with wp_unslash()

Page Detection Logic

The plugin determines which page is currently being viewed:

```
public function inject_code() {
   $current_page = '';

   if (is_front_page()) {
      $current_page = 'home';
   } elseif (is_page()) {
      $current_page = 'page_' . get_queried_object_id();
```

```
    } elseif (is_single() && get_post_type() === 'post') {
        $current_page = 'post_' . get_queried_object_id();
    }

    // Match against stored rules
}
```

Page Identification Format:

- Home page: 'home'
- Pages: 'page_123' (where 123 is the page ID)
- Posts: 'post_456' (where 456 is the post ID)

Code Injection Process

1. Rule Matching

1. Get all active rules from database
2. Determine current page type and ID
3. Check each rule's target pages (stored as JSON array)
4. Inject matching code

2. Safe Code Output

```
echo "\n<!-- ScatterBytes: " . esc_html($rule->rule_name) . " -->\n";
echo wp_unslash($rule->code_content) . "\n";
echo "<!-- End ScatterBytes: " . esc_html($rule->rule_name) . " -->\n\n";
```

- Adds HTML comments for debugging
- Uses wp_unslash() to properly handle code content
- Escapes rule names for security

AJAX Implementation

Frontend JavaScript

```
$.ajax({
    url: ajax_object.ajax_url,
    type: 'POST',
    data: {
```

```
        action: 'save_code_rule',
        nonce: ajax_object.nonce,
        // ... form data
    },
    success: function(response) {
        // Handle success
    }
});
```

Backend PHP Handler

```
public function save_code_rule() {
    // 1. Verify nonce
    // 2. Check user capabilities
    // 3. Validate input data
    // 4. Insert/update database
    // 5. Return JSON response
}
```

WordPress Best Practices Implemented

1. Proper Database Handling

- Uses $wpdb for database operations
- Prepared statements through WordPress methods
- Proper table prefix handling

2. Internationalization Ready

- All text strings are ready for translation
- Uses WordPress text domain system

3. WordPress Coding Standards

- Proper hook usage
- Consistent naming conventions
- WordPress-style comments and documentation

4. Performance Considerations

- Minimal database queries
- Efficient page detection
- Conditional loading based on active rules

Extensibility Points

The plugin can be extended through:

1. Additional Code Types

- Add new code types beyond JavaScript/CSS
- Modify the code_type field validation

2. Custom Page Types

- Extend page detection for custom post types
- Add support for category/tag pages

3. Additional Injection Points

- Add more WordPress hooks for different injection locations
- Implement conditional loading based on user roles or other criteria

4. Import/Export Functionality

- Add methods to export rules as JSON
- Implement bulk import capabilities

This architecture ensures the plugin integrates seamlessly with WordPress while maintaining security, performance, and extensibility.