Code Review Checklist for MainWP Documentation

- Formatting and Style
- WordPress Coding Standards
- Security Practices
- Error Handling
- MainWP-Specific Standards
- Performance Considerations
- Defensive Programming
- Documentation
- Compatibility
- Testing Considerations

This checklist should be used when reviewing code examples in MainWP documentation to ensure consistency, security, and adherence to WordPress coding standards.

Formatting and Style

- ☐ **Indentation**: Uses 4 spaces per WordPress standards (not tabs)
- ☐ **Brace Placement**: Opening brace on same line as statement, closing brace on its own line
- ☐ **Spacing**: Space after commas, around operators, and after control structure keywords
- ☐ **Line Length**: Lines kept to a reasonable length (generally under 100 characters)
- ☐ **Naming Conventions**:
  - ☐ Functions and variables use snake_case
  - ☐ Classes use CamelCase
  - ☐ Constants use UPPERCASE_WITH_UNDERSCORES

- [ ] **Commenting**:
  - [ ] DocBlocks for functions, classes, and methods
  - [ ] Inline comments for complex logic
  - [ ] Comments are clear and add value (not just restating the code)

WordPress Coding Standards

- [ ] **WordPress Functions**: Uses WordPress functions instead of PHP equivalents where appropriate
  - [ ] `wp_remote_get()` instead of `curl_exec()`
  - [ ] `sanitize_*()` functions for sanitization
  - [ ] `esc_*()` functions for escaping output
  - [ ] `wp_enqueue_script()` and `wp_enqueue_style()` for assets
- [ ] **Internationalization**:
  - [ ] Text strings are wrapped in `__()`, `_e()`, or equivalent
  - [ ] Text domain is included in translation functions
  - [ ] Placeholders use `sprintf()` instead of string concatenation
- [ ] **Capability Checks**:
  - [ ] Uses `current_user_can()` or `mainwp_current_user_can()` before operations
  - [ ] Uses appropriate capabilities (`manage_options` instead of just `read`)
- [ ] **Database Operations**:
  - [ ] Uses `$wpdb->prepare()` for SQL queries with variables
  - [ ] Uses WordPress functions for common operations (`get_option()`, `update_option()`, etc.)

Security Practices

- [ ] **Input Validation**:
  - [ ] All user inputs are validated before use
  - [ ] Type checking is performed where appropriate
  - [ ] Boundary checking for numeric values
- [ ] **Input Sanitization**:
  - [ ] All user inputs are sanitized using appropriate functions

- [ ] Uses context-appropriate sanitization functions
- [ ] **Output Escaping**:
  - [ ] All output is properly escaped using appropriate functions
  - [ ] Context-specific escaping functions are used
- [ ] **Nonce Verification**:
  - [ ] Nonces are created for forms and AJAX requests
  - [ ] Nonces are verified before processing data
- [ ] **Capability Checks**:
  - [ ] Appropriate capability checks before performing operations
- [ ] **File Operations**:
  - [ ] Secure file handling practices
  - [ ] Path traversal prevention

Error Handling

- [ ] **Graceful Failures**:
  - [ ] Functions return appropriate error values/objects
  - [ ] User-friendly error messages
- [ ] **Logging**:
  - [ ] Errors are logged when appropriate
  - [ ] Logging is conditional based on debug settings
- [ ] **Exception Handling**:
  - [ ] Try/catch blocks for operations that might throw exceptions
  - [ ] Proper exception handling and reporting

MainWP-Specific Standards

- [ ] **MainWP Hooks**:
  - [ ] Uses appropriate MainWP actions and filters
  - [ ] Hook naming follows MainWP conventions
- [ ] **MainWP Functions**:
  - [ ] Uses MainWP helper functions where available

- [ ] Follows MainWP patterns for common operations
- [ ] **Extension Registration**:
  - [ ] Proper extension registration with MainWP
  - [ ] Correct callback structure

Performance Considerations

- [ ] **Caching**:
  - [ ] Implements caching for expensive operations
  - [ ] Uses transients or object cache appropriately
- [ ] **Database Efficiency**:
  - [ ] Minimizes database queries
  - [ ] Uses efficient query patterns
- [ ] **Pagination**:
  - [ ] Implements pagination for large datasets
- [ ] **Resource Usage**:
  - [ ] Avoids memory-intensive operations
  - [ ] Cleans up resources after use

Defensive Programming

- [ ] **Null Checks**:
  - [ ] Checks for null/undefined values before use
- [ ] **Type Checking**:
  - [ ] Verifies variable types before operations
- [ ] **Error States**:
  - [ ] Handles error states gracefully
  - [ ] Returns meaningful error messages
- [ ] **Default Values**:
  - [ ] Provides sensible defaults for parameters
  - [ ] Handles empty or invalid inputs gracefully

Documentation

- [ ] **DocBlocks**:
  - [ ] All functions, classes, and methods have DocBlocks
  - [ ] Parameters and return values are documented
  - [ ] Types are specified for parameters and return values
- [ ] **Inline Comments**:
  - [ ] Complex logic is explained with inline comments
  - [ ] Comments add value beyond what the code itself conveys
- [ ] **Example Usage**:
  - [ ] Examples show typical usage patterns
  - [ ] Examples include error handling

Compatibility

- [ ] **PHP Version**:
  - [ ] Code is compatible with PHP 7.0+ (MainWP's minimum requirement)
  - [ ] Avoids deprecated PHP functions
- [ ] **WordPress Version**:
  - [ ] Code is compatible with WordPress 5.0+ (MainWP's minimum requirement)
  - [ ] Avoids deprecated WordPress functions
- [ ] **MainWP Version**:
  - [ ] Code is compatible with current MainWP version
  - [ ] Notes any version-specific considerations

Testing Considerations

- [ ] **Testability**:
  - [ ] Code can be reasonably tested
  - [ ] Functions have clear inputs and outputs
- [ ] **Edge Cases**:
  - [ ] Handles edge cases appropriately

- [ ] Considers empty inputs, large datasets, etc.

- [ ] **Error Scenarios**:
  - [ ] Handles error scenarios gracefully
  - [ ] Provides useful feedback for debugging

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