Shortcodes

- Shortcodes
- Basic Shortcodes
- Enclosing Shortcodes
- Shortcodes with Parameters
- TinyMCE Enhanced Shortcodes

Shortcodes

As a security precaution, running PHP inside WordPress content is forbidden; to allow dynamic interactions with the content, Shortcodes were presented in WordPress version 2.5.

Shortcodes are macros that can be used to perform dynamic interactions with the content. i.e creating a gallery from images attached to the post or rendering a video.

Why Shortcodes?

Shortcodes are a valuable way of keeping content clean and semantic while allowing end users some ability to programmatically alter the presentation of their content.

When the end user adds a photo gallery to their post using a shortcode, they’re using the least data possible to indicate how the gallery should be presented.

Advantages:

- No markup is added to the post content, which means that markup and styling can easily be manipulated on the fly or at a later state.
- Shortcodes can also accept parameters, allowing users to modify how the shortcode behaves on an instance by instance basis.

Built-in Shortcodes

By default, WordPress includes the following shortcodes:

- `[caption]` – allows you to wrap captions around content
- `[gallery]` – allows you to show image galleries
- `[audio]` – allows you to embed and play audio files
- `[video]` – allows you to embed and play video files
- `[playlist]` – allows you to display collection of audio or video files
- `[embed]` – allows you to wrap embedded items

Shortcode Best Practices

Best practices for developing shortcodes include the plugin development best practices and the list below:

- **Always return!** Shortcodes are essentially filters, so creating "side effects" will lead to unexpected bugs.
- Prefix your shortcode names to avoid collisions with other plugins.
- Sanitize the input and escape the output.
- Provide users with clear documentation on all shortcode attributes.

Quick Reference

See the complete example of using a basic shortcode structure, taking care of self-closing and enclosing scenarios, shortcodes within shortcodes and securing output.

External Resources

- WordPress Shortcodes Generator

Basic Shortcodes

Add a Shortcode

It is possible to add your own shortcodes by using the Shortcode API. The process involves registering a callback `$func` to a shortcode `$tag` using `add_shortcode()`.

```
add_shortcode(
  string $tag,
  callable $func
);
```

`[wporg]` is your new shortcode. The use of the shortcode will trigger the `wporg_shortcode` callback function.

```
add_shortcode( 'wporg', 'wporg_shortcode' );
function wporg_shortcode( $atts = [], $content = null) {
  // do something to $content
  // always return
  return $content;
}
```

Remove a Shortcode

It is possible to remove shortcodes by using the Shortcode API. The process involves removing a registered `$tag` using `remove_shortcode()`.

```
remove_shortcode(
  string $tag
);
```

Make sure that the shortcode have been registered before attempting to remove. Specify a higher priority number for `add_action()` or hook into an action hook that is run later.

Check if a Shortcode Exists

To check whether a shortcode has been registered use `shortcode_exists()`.

Enclosing Shortcodes

The are two scenarios for using shortcodes:

- The shortcode is a self-closing tag like we seen in the Basic Shortcodes section.
- The shortcode is enclosing content.

Enclosing Content

Enclosing content with a shortcode allows manipulations on the enclosed content.

```
[wporg]content to manipulate[/wporg]
```

As seen above, all you need to do in order to enclose a section of content is add a beginning `[$tag]` and an end `[/$tag]`, similar to HTML.

Processing Enclosed Content

Lets get back to our original `[wporg]` shortcode code:

```
function wporg_shortcode( $atts = array(), $content = null ) {
  // do something to $content
  // always return
  return $content;
}
add_shortcode( 'wporg', 'wporg_shortcode' );
```

Looking at the callback function we see that we chose to accept two parameters, `$atts` and `$content`. The `$content` parameter is going to hold our enclosed content. We will talk about `$atts` later.

The default value of `$content` is set to `null` so we can differentiate between a self-closing tag and enclosing tags by using PHP function `is_null()`.

The shortcode `[$tag]`, including its content and the end `[/$tag]` will be replaced with the **return value** of the handler function.

[alert]It is the responsibility of the handler function to secure the output.[/alert]

Shortcode-ception

The shortcode parser performs a **single pass** on the content of the post.

This means that if the `$content` parameter of a shortcode handler contains another shortcode, it won't be parsed. In this example, `[shortcode]` will not be processed:

```
[wporg]another [shortcode] is included[/wporg]
```

Using shortcodes inside other shortcodes is possible by calling `do_shortcode()` on the **final return value** of the handler function.

```
function wporg_shortcode( $atts = array(), $content = null ) {
  // do something to $content
  // run shortcode parser recursively
  $content = do_shortcode( $content );
  // always return
  return $content;
}
add_shortcode( 'wporg', 'wporg_shortcode' );
```

Limitations

The shortcode parser is unable to handle mixing of enclosing and non-enclosing forms of the same `[$tag]`.

```
[wporg] non-enclosed content [wporg]enclosed content[/wporg]
```

Instead of being treated as two shortcodes separated by the text "`non-enclosed content`", the parser treats this as a single shortcode enclosing "`non-enclosed content [wporg]enclosed content`".

Shortcodes with Parameters

Now that we know how to create a basic shortcode and how to use it as self-closing and enclosing, we will look at using parameters in shortcode `[$tag]` and handler function.

Shortcode `[$tag]` can accept parameters, known as attributes:

```
[wporg title="WordPress.org"]
  Having fun with WordPress.org shortcodes.
[/wporg]
```

Shortcode handler function can accept 3 parameters:

- `$atts` – array – `[$tag]` attributes
- `$content` – string – The content inside your shortcode. In the exampe above, it will be "Having fun with WordPress.org shortcodes".
- `$tag` – string – the name of the `[$tag]` (i.e. the name of the shortcode)

```
function wporg_shortcode( $atts = array(), $content = null, $tag = '' ) {}
```

Parsing Attributes

For the user, shortcodes are just strings with square brackets inside the post content. The user have no idea which attributes are available and what happens behind the scenes.

For plugin developers, there is no way to enforce a policy on the use of attributes. The user may include one attribute, two or none at all.

To gain control of how the shortcodes are used:

- Declare default parameters for the handler function.
- Performing normalization of the key case for the attributes array with `array_change_key_case()`.
- Parse attributes using `shortcode_atts()` providing default values array and user `$atts`.
- Secure the output before returning it.

Complete Example

Complete example using a basic shortcode structure, taking care of self-closing and enclosing scenarios and securing output.

A `[wporg]` shortcode that will accept a title and will display a box that we can style with CSS.

```
/**
 * The [wporg] shortcode.
 *
 * Accepts a title and will display a box.
 *
 * @param array  $atts    Shortcode attributes. Default empty.
 * @param string $content Shortcode content. Default null.
 * @param string $tag     Shortcode tag (name). Default empty.
 * @return string Shortcode output.
 */
function wporg_shortcode( $atts = [], $content = null, $tag = '' ) {
  // normalize attribute keys, lowercase
  $atts = array_change_key_case( (array) $atts, CASE_LOWER );

  // override default attributes with user attributes
  $wporg_atts = shortcode_atts(
    array(
      'title' => 'WordPress.org',
    ), $atts, $tag
  );

  // start box
  $o = '<div class="wporg-box">';

  // title
  $o .= '<h2>' . esc_html( $wporg_atts['title'] ) . '</h2>';

  // enclosing tags
  if ( ! is_null( $content ) ) {
    // $content here holds everything in between the opening and the closing tags of your
shortcode. eg.g [my-shortcode]content[/my-shortcode].
    // Depending on what your shortcode supports, you will parse and append the content to
```

```
your output in different ways.
    // In this example, we just secure output by executing the_content filter hook on
$content.
    $o .= apply_filters( 'the_content', $content );
  }

  // end box
  $o .= '</div>';

  // return output
  return $o;
}

/**
 * Central location to create all shortcodes.
 */
function wporg_shortcodes_init() {
  add_shortcode( 'wporg', 'wporg_shortcode' );
}

add_action( 'init', 'wporg_shortcodes_init' );
```

TinyMCE Enhanced Shortcodes

It's possible to parse shortcodes within the visual editor of TinyMCE and make them render actual content, rather than the shortcode itself.

Switching to the `Text` tab allows you to see the actual shortcode again.

Below are the built-in WordPress shortcodes that use this functionality.

Audio Shortcode

The `[audio]` shortcode allows you to embed a single audio file.

The audio shortcode

Caption Shortcode

The `[caption]` shortcode wraps the image in a div and puts a `<p class="wp-caption-text">` tag around the caption.

The caption shortcode

Gallery Shortcode

The `[gallery]` shortcode allows you to embed several images at once in a div.

The gallery shortcode

Playlist Shortcode

The `[playlist]` shortcode allows you to attach more than one media file and render with an html5 playlist.

The playlist shortcode

Video Shortcode

The `[video]` shortcode is very similar to the `[audio]` shortcode; it simply renders a video instead of audio.

The video shortcode