Inserting and using jQuery plugins in WordPress

One of the most frequently asked questions on forums, blogs and mailing lists concerns the integration of jQuery plugins in WordPress. Integrating jQuery plugins in WordPress is not a matter of queuing scripts only, but it also means understanding the correct hierarchy and order by which scripts are executed within WordPress. Let's see the details of this topic.

Order and hierarchy

In WordPress, a jQuery plugin depends on the jQuery library itself. For that reason, you should always specify such a dependence when you register your plugins:

```
wp_enqueue_script('jquery');
wp_register_script('plugin', 'url/to/plugin.js',
array('jquery'), '1.0', false);
wp_enqueue_script('plugin');
```

The wp_register_script() function takes as its third parameter an array containing all the scripts our plugin depends on. In this case, the main dependence script is jQuery.

If you don't follow this approach, your plugin is likely to fail.

Invoking the plugin

You should invoke your plugin just after the main plugin file, always respecting the hierarchy mentioned above. For that reason, your code should be put within a separate JavaScript file.

Within this file, you should always wrap your code in the main jQuery namespace, like so:

```
(function($) {

 $(function() {

  $(element).plugin();


 });


})(jQuery);
```

In the same way, your JavaScript file should be enqueued just after the main jQuery plugin:

```
wp_enqueue_script('jquery');
wp_register_script('plugin', 'url/to/plugin.js',
array('jquery'), '1.0', false);
wp_enqueue_script('plugin');
wp_register_script('script', 'url/to/script.js',
array('jquery'), '1.0', false);
wp_enqueue_script('script');
```

As you can see, we're always respecting the script hierarchy established by WordPress.

Use the wp_enqueue_scripts action

All the code showed so far should be put in a separate function to be added to the `wp_enqueue_scripts` action:

```
function add_my_scripts() {
        if(!is_admin()) {
                wp_enqueue_script('jquery');
                wp_register_script('plugin',
'url/to/plugin.js', array('jquery'), '1.0', false);
                wp_enqueue_script('plugin');
                wp_register_script('script',
'url/to/script.js', array('jquery'), '1.0', false);
                wp_enqueue_script('script');
        }
}

add_action('wp_enqueue_scripts', 'add_my_scripts');
```

When a plugin must accept theme options

If your jQuery plugin must accept theme options as its parameters, you can insert the plugin call directly in the document's content. I'm not a big fan of this approach because it tends to create several bottlenecks during page loading. However, it's the most used approach followed by many theme developers.

Either you choose to use the `header.php` or the `footer.php` template, take a look at the position of the `wp_head()` and `wp_footer()` functions within your templates.

If you've included your scripts using the `wp_enqueue_scripts` action, then WordPress will use such functions as a sort of placeholders to insert your files within the markup.

For that reason, your plugin's call must always be inserted **after** these functions or you'll get a JavaScript error because you're attempting to use a jQuery's method which has not been defined yet:

```
<?php wp_head(); ?>
<script type="text/javascript">
(function($) {
        $(function() {
                <?php
                        $slideshow_effect =
get_option('my-theme-slideshow-effect');
                ?>

                $('#slideshow').nivoSlider({
                        effect: '<?php echo
$slideshow_effect; ?>'
                });
        });

})(jQuery);
</script>
```

Once you've properly understood this process, achieving your goals is pretty simple.