Advanced Plugin Topics (part 1): Meta Boxes and Cron

The next few items (Meta Boxes, and Cron) and are nice to have, require a lot more work and are not necessary for the majority of plugin projects. I include them here because I'm interested in what they do and can think of a few projects that would benefit from one or more of them.

Custom Meta Boxes

Custom Meta Boxes allow you to create custom sections of the editor screen where users can enter custom data and you can then sanitize it and save it to the database.

Because they allow custom data we need to follow all the security rules: sanitize the input and check roles and capabilities before we do anything with the data we're working with.

Since this has admin-facing data we should also take care of internatinalization for the UI labels and strings.

Adding the box

The first step is to add a meta box to the editor screen.

add_meta_box() create an empty box.

The add_meta_boxes() takes three parameters:

- the name of the action to take, `add_meta_boxes`
- A callback function with the specific code we want to run
- The post type we want to run this meta box for

Potential Gotcha

The function is `add_meta_box()` (singular). The hook to register the metaboxes is `add_meta_boxes()` (plural). Keep this in mind when writing code that adds meta boxes to the editor.

```
<?php
function rivendellweb_custom_meta() {
  add_meta_box( 'rivendellweb_meta',
    __( 'Meta Box Title',
    'rivendellweb-textdomain' ),
    'rivendellweb_meta_callback',
    'post'
  );
}

add_action( 'add_meta_boxes', 'rivendellweb_custom_meta' );
```

You can also use `add_meta_boxes_{post_type}` for best practice, so your hook will only run when editing a specific post type. This will only receive 1 parameter – $post.

The modified code below will only run when editing post.

```
<?php
function rivendellweb_custom_meta() {
  add_meta_box( 'rivendellweb_meta',
    __( 'Meta Box Title',
    'rivendellweb-textdomain' ),
    'rivendellweb_meta_callback',
    'post'
  );
}

add_action( 'add_meta_boxes_post', 'rivendellweb_custom_meta' );
```

This customized version is particularly useful when working with Custom Post Types.

Creating The Box Contents

This is, to me, the most tedious part of the meta box process. Once we've defined the container box we must write the content that will go inside.

It is at this point that we also need to incorporate all the security measures we want our code to use, particularly nonces, capabilities and internationalization.

In the code below we do two things:

- We provide the form for the user to enter data
- If there is a value for the field we display it for the user to change. There is no need to manually retrieve the data

Note:

there are no submit buttons in meta boxes. The value of the meta boxes are transfered via POST when the user clicks on the Publish or Update buttons.

```
<?php
function rivendellweb_meta_callback( $post ) {
  wp_nonce_field( basename( __FILE__ ), 'rive'rivendellweb_nonce' $rivende
?>

  <p>
    <label for="meta-text" class="rivendellweb-row-title"><?php _e( 'Exam
    <input type="text" name="meta-text" id="meta-text" value="<?php if ( 
  </p>

  <p>
    <span class="rivendellweb-row-title"><?php _e( 'Exam'Example Checkbox
    <div class="rivendellweb-row-content">
      <label for="meta-checkbox">
        <input type="checkbox" name="meta-checkbox" id="meta-checkbox" va
        <?php _e( 'Chec'Checkbox label'vendellweb-textdomain' )?>
      </label>
      <label for="meta-checkbox-two">
```

```
        <input type="checkbox" name="meta-checkbox-two" id="meta-checkbox-
        <?php _e( 'Anot'Another checkbox'vendellweb-textdomain' )?>
      </label>
    </div>
  </p>
  <?php
}
```

Save the data

There may be times when you want to save the data entered by the user outside the update or publish events or you have chosen to save the data outside of WordPress.

This example checks if the value we're trying to save already exists in the `post_meta` table in the database.

If the post exists, update_post_meta() will update the value. If it doesn't exist, then it will create it. We need to do this for every meta box field that we create.

The key will be to figure out how to batch update operations together to keep performance within acceptable levels.

```
<?php
function rivendellweb_save_postdata( $post_id ) {
    if ( array_key_exists( 'rivendellweb_field', $_POST ) ) {
      update_post_meta(
        $post_id,
        '_rivendellweb_meta_key',
        $_POST['rivendellweb_field']
      );
    }
}

add_action( 'save_post', 'rivendellweb_save_postdata' );
```

With these three stages we can save custom data to the WordPress database. There are many things that we can do with Metaboxes, they will require careful

planning and layout to make sure they make sense to your users and remain accessible.

You can also use OOP techniques and classes to create your meta boxes. The plugin handbook has an OOP Example.

On the down side, there is Gutenberg.

Gutenberg support metaboxes as a transitional step towards the new editor. We will discuss the Gutenberg replacement for metaboxes (yes, it exists) in a future post.

Cron in WordPress

WP-Cron is how WordPress handles scheduling time-based tasks in WordPress. Several WordPress core features, such as checking for updates and publishing scheduled post, utilize WP-Cron.

WP-Cron works by checking, on every page load, a list of scheduled tasks to see what needs to be run. Any tasks due to run will be called during that page load.

> **Note:**
>
> WP-Cron only triggered on page load. It does not run constantly as the system cron does

Scheduling errors could occur if you schedule a task for 2:00PM and no page loads occur until 5:00PM.

Why would you want a Cron-like tool in WordPress?

Having a task scheduler like Unix/Linux/macOS is always nice to have so we can run tasks on a set schedule; but it’s also a pain because we don’t always have access to the server’s task scheduler.

The WordPress API is a simpler method for setting scheduled tasks than going outside of WordPress to the system scheduler.

WP-Cron queues all scheduled tasks and will run at the next page load. You can't be 100% sure *when* your task will run, you can be 100% sure that it will run *eventually*.

Now for how it works

WP-Cron is given two arguments: the time for the first task, and an interval (in seconds) after which the task should be repeated.

To simplify scheduling tasks, WordPress provides some default intervals and an easy method for adding custom intervals.

The default intervals provided by WordPress are:

- hourly
- twicedaily
- daily
- weekly

You can also create your own custom intervals using the cron_scedules() with a callbackk function that defines the interval you want to use.

```php
<?php
function example_add_cron_interval( $schedules ) {
    $schedules['five_seconds'] = array(
        'interval' => 5,
        'display'  => esc_html__( 'Every Five Seconds' ), );
    return $schedules;
}

add_filter( 'cron_schedules', 'example_add_cron_interval' );
```

Because WP-Cron only works on page load, it is possible that events will not run on time. You can use your system's task scheduler to run WP-Cron. The process is OS dependent and is documented in Hooking WP-Cron Into the System Task Scheduler