How to Log Custom Events

Developer · https://logtivity.io/docs/log-custom-events/

The Logtivity plugin provides a simple way to create custom log entries beyond what we've built into the plugin already.

Getting Started with Logtivity Logging

The following line of code will send a log entry to Logtivity every time it executes.

```
Logtivity::log()->setAction('My Custom Action')->send();
```

This will create a log entry in your site's Activity Log with My Custom Action in the action column. You can also use this following shorthand:

```
Logtivity::log('My Custom Action')->send();
```

If you want to provide additional context:

```
Logtivity::log()
  ->setAction('My Custom Action')
  ->setContext('My Custom Context')
  ->send();
```

If there is additional information you want to attach to the log entry, you can add as many meta data entries as you need:

```
Logtivity::log('My Custom Action')
  ->setContext('My Custom Context')
  ->addMeta('First Key', 'First Value')
  ->addMeta('Second Key', 'Second Value')
  ->send();
```

If you want to gather data to build your log entry, just wait before using the `send()` method.

```
$log = Logtivity::log('My Custom Action');

// code, code, code

$log->setContext($whateverItNeedsToBe);

// Loop through a collection of information
foreach ($items as $metaKey => $metaValue) {
  $log->addMetaIf($metaValue != 'my expection', $metaKey, $metavalue);
}

$log->send();
```

Note: We introduced a variant of addMeta(), addMetaIf(). The first argument must evaluate to a true/false expression. The metadata will not be added if false.

Integrating Logtivity with plugins

To add log entries for a 3rd party plugin, the key part is figuring out the action hooks the plugin is using.

We will provide some examples below to give you an idea of how to create custom log entries. All of the examples below were tested in the template's function.php file. You could use a code plugin instead.

This first example is for the Gravity Forms plugin. Gravity Forms provides great documentation on available hooks.

```
if (class_exists(Logtivity::class)) {
add_action('gform_after_save_form', function ($form, $is_new) {
Logtivity::log('gform_after_save_form')
->setContext($is_new ? 'Created' : 'Updated')
->addMeta('Title', $form['title'])
->addMeta('Description', $form['description'])
->send();
```

```
}, 10, 2);

add_action('gform_before_delete_form', function ($form_id) {
    gf_delete('gform_before_delete_form', $form_id);
});

add_action('gform_post_form_trashed', function ($form_id) {
    gf_delete('gform_post_form_trashed', $form_id);
});

add_action('gform_post_form_restored', function ($form_id) {
    gf_delete('gform_post_form_restored', $form_id);
});

function gf_delete($action, $form_id)
{
    $form = GFFormsModel::get_form($form_id, true);

    $log = Logtivity::log($action)
        ->setContext($form_id)
        ->addMetaIf($form, 'Title', $form->title)
        ->send();
}
}
```

This second example is for the popular Redirection plugin. We are going track the use of this plugin for creating, updating and deleting redirects. The redirection plugin has two action hooks of interest for this. The `redirection_redirect_updated` hook is called for both updates and creates, so we have to do a little work.

```
add_action('redirection_redirect_updated', function ($id, $created = null) {
    // Verify Logtivity is available
    if (class_exists(Logtivity::class)) {
        if ($id instanceof Red_Item) {
            // First argument is a redirect object
            $redirect = $id;
            $context  = 'Update';

        } else {
            // First argument is a redirect ID
```

```
            $redirect = $created ?: Red_Item::get_by_id($id);
            $context  = 'Create';
        }

        Logtivity::log('redirection_redirect_updated')
            ->setContext($context)
            ->addMeta('URL', $redirect->get_url())
            ->send();
    }
});
```

To track deletes, we use the Redirection's plugin delete hook:

```
add_action('redirection_redirect_deleted', function($redirect) {
  if (class_exists(Logtivity::class)) {
    Logtivity::log('redirection_redirect_deleted')
      ->addMeta('URL', $redirect->get_url())
      ->send();
  }
});
```

Next, let's look at logs for settings. By default, Logtivity catches many changes to options and settings in plugins. When settings are changed in the WP Rocket plugin, Logtivity automatically send the following action/context entries:

- Option Updated/wp_rocket_settings
- Settings Updated/Core:wprocket

But that's all the information available. Using the `rocket_options_changed` action hook, we can provide some additional information:

```
if (class_exists(Logtivity::class)) {
    add_action('rocket_options_changed', function ($settings) {
        $log = Logtivity::log('rocket_options_changed');

        foreach ($settings as $key => $value) {
            $log->addMeta($key, $value);
        }
```

```
        $log->send();
    });

    add_action('wp_logtivity_instance', function ($log) {
        if (in_array(strtolower((string)$log->context), ['wp_rocket_settings', 'core:wprocket']))
{
            $log->stop();
        }
    });
}
```

We loop through all the settings to add meta data for each value. Some of the values are themselves arrays and the addMeta() method is unable to process an array containing arrays at this time. Also note how we use an action hook in the Logtivity plugin itself to catch and stop those two other log entries sent by default. They are no longer useful since our new custom log entry provides more information, more concisely.

Here's a fourth example, this time using the PerfMatters plugin. Perfmatters does not provide its own action hook. However, since Logtivity already logs the `update_option` hook in the WordPress core, two "Option Updated" log entries are created by default whenever you update the settings in Perfmatters. We can cancel these and replace them with our own custom log entry. The sample code below will stop the default logs and send a new log entry that shows the updated values for any settings that were changed in Perfmatters.

```
if (class_exists(Logtivity::class)) {
    add_action('wp_logtivity_instance', function (Logtivity_Logger $log) {
        if (
            strtolower($log->action) == 'option updated'
            && stripos((string)$log->context, 'perfmatters_') === 0
        ) {
            $log->stop();
        }
    });

    add_action('update_option', function ($option, $old_value, $new_value) {
        if (stripos($option, 'perfmatters_') === 0) {
            $log = Logtivity::log('Perfmatters Settings')
                ->setContext($option);
```

```
            $keys = array_unique(
                array_merge(
                    array_keys($old_value), array_keys($new_value)
                )
            );
            foreach ($keys as $key) {
                $new = $new_value[$key] ?? 'Off';
                $old = $old_value[$key] ?? 'Off';

                if ($old != $new) {
                    $log->addMeta($key, $new ?: 'Off/Default');
                }

            }
            $log->send();
        }
    }, 10, 3);
}
```