Drupal Book

Actions

TABLE OF CONTENTSE

views 299

Overview

The Actions module is a core module in Drupal 9 that allows site builders and developers to create automated workflows by defining actions and triggers. In Drupal, actions refer to a set of tasks or operations that can be performed on a site. For example, an action can be sending an email, publishing content, or updating a database record.

With the Actions module, you can create a customized workflow that can be triggered by a specific event. For example, when a user submits a form on your website, you can trigger an action to send an email notification to the site administrator. You can also chain multiple actions together to create complex workflows.

The Actions module provides a user-friendly interface for defining and managing actions. You can create custom actions using PHP code (see below) or use pre-defined actions provided by Drupal core or contributed modules. The module also provides a way to organize and categorize actions, making it easy to find and use them in your workflows.

TL;DR: The Actions module provides a powerful and flexible way to automate tasks on your website by defining actions and triggers, allowing you to create customized workflows that improve your site's functionality and user experience. - ChatGPT (with minor tweaks on my part.)

From the config page in Drupal:

> There are two types of actions: simple and advanced. Simple actions do not require any additional configuration and are listed here automatically. Advanced actions need to be created and configured before they can be used because they have options that need to be specified; for example, sending an email to a specified address or unpublishing content containing certain words. To create an advanced action, select the action from the drop-down list in the advanced action section below and click the Create button.

> NOTE
> The ECA module can use Drupal core actions (and events) if you need to do even more.

Custom Actions

Here, A Drupal action is a fuctionality which performs specific action when executed. For example, Archive Node or Make Content Sticky.

Actions use the annotation class `Drupal\Core\Annotation\Action`, and extend `Drupal\Core\Action\ActionBase` or `Drupal\Core\Action\ConfigurableActionBase` (if the action is configurable.)

Action Plugin definition is defined in Plugin Annotation. It has 3 required keys

```
/**
 * Provides an Archive Node Action.
 *
 * @Action(
 *   id = "action_plugin_examples_archive_node",
 *   label = @Translation("Archive Node"),
 *   type = "node",
 *   category = @Translation("Custom")
 * )
 */
```

id –> ID of the Action Plugin

label –> Name of the Action Plugin

type –> Entity type to which the Action Plugin belongs to

category –> (optional) Category of the Action Plugin

Archive Node Action (simple)

This is a simple action which requires no configuration. When it is run, it changes the alias of the node to /archive/ /. It also sets the title to have the word `Archive` in the front of it. Finally it disables the sticky and promoted flags. [It is also on gitlab here.](https://git.drupalcode.org/sandbox/Bhanu951-3103712/-/blob/8.x-dev/action_plugin_examples/src/Plugin/Action/ArchiveNode.php)

```
<?php

namespace Drupal\action_plugin_examples\Plugin\Action;

use Drupal\Core\Action\ActionBase;
use Drupal\Core\Session\AccountInterface;
use Drupal\Core\Plugin\ContainerFactoryPluginInterface;
use Symfony\Component\DependencyInjection\ContainerInterface;
use Drupal\pathauto\PathautoState;

/**
 * Provides an Archive Node Action.
 *
 * @Action(
 *   id = "action_plugin_examples_archive_node",
 *   label = @Translation("Archive Node"),
 *   type = "node",
 *   category = @Translation("Custom")
 * )
 */
class ArchiveNode extends ActionBase implements ContainerFactoryPluginInterface {


  /**
   * The Messenger service.
   *
   * @var \Drupal\Core\Messenger\MessengerInterface
   */
  protected $messenger;

  /**
   * Logger service.
   *
   * @var \Drupal\Core\Logger\LoggerChannelFactoryInterface
```

```php
 */
protected $logger;

/**
 * The path alias manager.
 *
 * @var \Drupal\path_alias\AliasManagerInterface
 */
protected $aliasManager;

/**
 * Language manager for retrieving the default Langcode.
 *
 * @var \Drupal\Core\Language\LanguageManagerInterface
 */
protected $languageManager;

/**
 * {@inheritdoc}
 */
public static function create(ContainerInterface $container, array $configuration, $plugin_id, $plugin_definition) {
  $instance = new static($configuration, $plugin_id, $plugin_definition);
  $instance->logger = $container->get('logger.factory')->get('action_plugin_examples');
  $instance->messenger = $container->get('messenger');
  $instance->aliasManager = $container->get('path_alias.manager');
  $instance->languageManager = $container->get('language_manager');
  return $instance;
}

/**
 * {@inheritdoc}
 */
public function access($node, AccountInterface $account = NULL, $return_as_object = FALSE) {
  /** @var \Drupal\node\NodeInterface $node */
  $access = $node->access('update', $account, TRUE)
    ->andIf($node->title->access('edit', $account, TRUE));
  return $return_as_object ? $access : $access->isAllowed();
}

/**
 * {@inheritdoc}
 */
public function execute($node = NULL) {

  /** @var \Drupal\node\NodeInterface $node */

  $language = $this->languageManager->getCurrentLanguage()->getId();

  $old_alias = $this->aliasManager->getAliasByPath('/node/' . $node->id(), $language);

  $title = $node->getTitle();
  $date = $node->created->value;
  $year = date('Y', $date);
  // $old_alias = $node->path->alias;
  $new_title = $this->t('[Archive] | @title', ['@title' => $title]);
  $node->setTitle($new_title);
  $node->setSticky(FALSE);
  $node->setPromoted(FALSE);

  $new_alias = '/archive/' . $year . $old_alias;
```

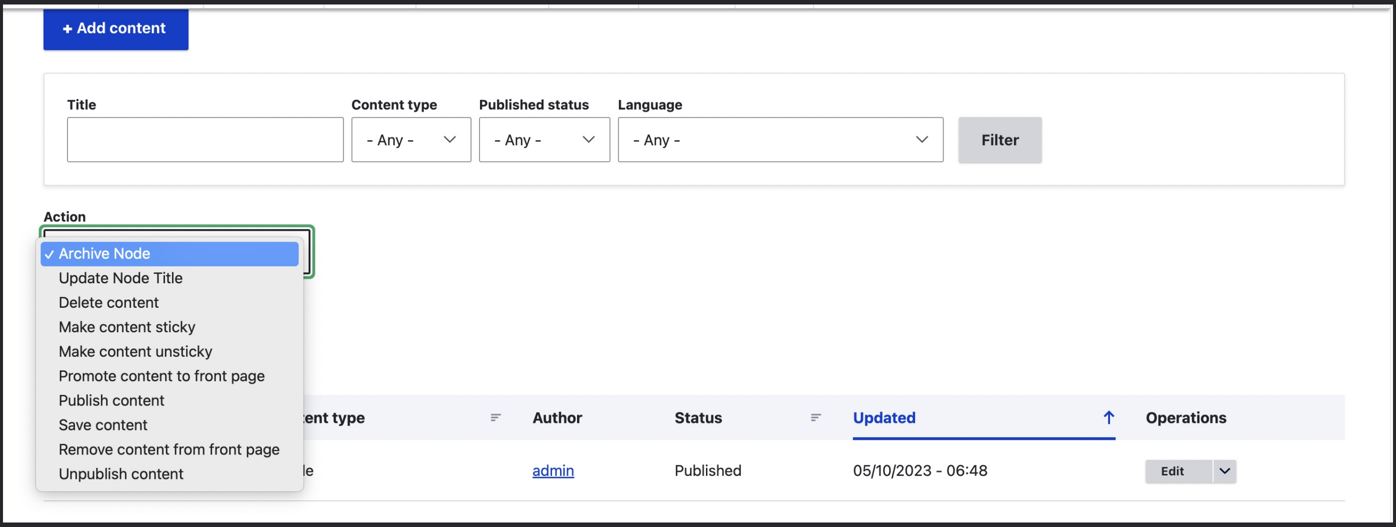
```
    $node->set("path", [
      'alias' => $new_alias,
      'langcode' => $language,
      'pathauto' => PathautoState::SKIP,
    ]);

    $node->save();

    $message = $this->t('Node with NID : @id Archived.', ['@id' => $node->id()]);

    $this->logger->notice($message);
    $this->messenger->addMessage($message);

  }
}
```

In order to get Action Plugin Discoverable you need to add `system.action.<plugin_id>.yml` which is placed in `config/install`

The structure of the `.yml` file is shown below:

```
langcode: en
status: true
id: action_plugin_examples_archive_node
label: 'Archive Node'
type: node
plugin: action_plugin_examples_archive_node
```

Created Action Plugin can be viewed on the `/admin/content` page.



Update Node Title Custom Action Plugin with Configuration

This example updates a node title. It gets the new title info from configuration. [It is also here on Gitlab.]

```
<?php

namespace Drupal\action_plugin_examples\Plugin\Action;

use Drupal\Core\Action\ConfigurableActionBase;
use Drupal\Core\Form\FormStateInterface;
use Drupal\Core\Session\AccountInterface;
use Drupal\Core\Plugin\ContainerFactoryPluginInterface;
use Symfony\Component\DependencyInjection\ContainerInterface;

/**
```

```
 * Provides an Update Node Title Action.
 *
 * @Action(
 *   id = "action_plugin_examples_update_node_title",
 *   label = @Translation("Update Node title"),
 *   type = "node",
 *   category = @Translation("Custom")
 * )
 */
class UpdateNodeTitle extends ConfigurableActionBase implements ContainerFactoryPluginInterface {

  /**
   * The Messenger service.
   *
   * @var \Drupal\Core\Messenger\MessengerInterface
   */
  protected $messenger;

  /**
   * Logger service.
   *
   * @var \Drupal\Core\Logger\LoggerChannelFactoryInterface
   */
  protected $logger;

  /**
   * The path alias manager.
   *
   * @var \Drupal\path_alias\AliasManagerInterface
   */
  protected $aliasManager;

  /**
   * Language manager for retrieving the default Langcode.
   *
   * @var \Drupal\Core\Language\LanguageManagerInterface
   */
  protected $languageManager;

  /**
   * {@inheritdoc}
   */
  public static function create(ContainerInterface $container, array $configuration, $plugin_id, $plugin_definition) {
    $instance = new static($configuration, $plugin_id, $plugin_definition);
    $instance->logger = $container->get('logger.factory')->get('action_plugin_examples');
    $instance->messenger = $container->get('messenger');
    $instance->aliasManager = $container->get('path_alias.manager');
    $instance->languageManager = $container->get('language_manager');
    return $instance;
  }

  /**
   * {@inheritdoc}
   */
  public function defaultConfiguration() {
    return ['title' => 'Updated New Title'];
  }

  /**
   * {@inheritdoc}
   */
  public function buildConfigurationForm(array $form, FormStateInterface $form_state) {
    $form['title'] = [
      '#title' => $this->t('New Title'),
      '#type' => 'textfield',
      '#required' => TRUE,
      '#default_value' => $this->configuration['title'],
    ];
    return $form;
  }

  /**
   * {@inheritdoc}
   */
  public function submitConfigurationForm(array &$form, FormStateInterface $form_state) {
    $this->configuration['title'] = $form_state->getValue('title');
  }

  /**
   * {@inheritdoc}
   */
  public function access($node, AccountInterface $account = NULL, $return_as_object = FALSE) {
    /** @var \Drupal\node\NodeInterface $node */
    $access = $node->access('update', $account, TRUE)
      ->andIf($node->title->access('edit', $account, TRUE));
    return $return_as_object ? $access : $access->isAllowed();
  }

  /**
   * {@inheritdoc}
   */
  public function execute($node = NULL) {
    /** @var \Drupal\node\NodeInterface $node */
    $old_title = $node->getTitle();
    $updated_title = $this->t('@new_title @old_title', ['@old_title' => $old_title, '@new_title' => $this->configuration['title']]);

    $node->setTitle($updated_title)->save();

    $message = $this->t('Node Title for Node with NID : @id Updated.', ['@id' => $node->id()]);

    $this->logger->notice($message);
    $this->messenger->addMessage($message);
  }

}
```

The Schema for Action Plugin `action_plugin_examples.schema.yml` which is in `config/schema`:

```yaml
action_plugin_examples.configuration.action_plugin_examples_update_node_title:
  type: mapping
  label: 'Configuration for "Update node title" action'
  mapping:
    title:
      type: string
      label: Title
```

Action Plugin Definition `system.action.<plugin_id>.yml` is in `config/install`:

```yaml
langcode: en
status: true
dependencies:
  module:
    - node
    - pathauto
id: action_plugin_examples_update_node_title
label: 'Update Node Title'
type: node
plugin: action_plugin_examples_update_node_title
configuration:
  title: 'Updated New Title'
```

Note. You need to install the core module Actions first to be able to configure action plugins in UI.

Configuration Page `/admin/config/system/actions`

On the top of the page your new plugin should be present in the select list for creating advanced actions. When you select the plugin you should see the configuration form of your plugin:

See the change record: Actions are now plugins, configured actions are configuration entities

Reference

- Actions UI module overview updated January 2023
- Change Record - Actions are now plugins, configured actions are configuration entities
- ECA: Event-Condition-Action - no-code solution to orchestrate your Drupal site

---

Back to top