Layout Builder:

Beyond the Basics

DrupalCorn 2019 • November 7, 2019

Presented by Christopher Burge

Digital Experience Group, University of Nebraska-Lincoln

IN OUR GRIT, OUR GLORY™

Today's Presentation

- Create your own layout
- Add settings to a layout
- Make settings available to block/field templates
- Create a Layout Builder Restrictions plugin
- Alter Layout Builder block forms
- Gotchas and current issues

Layout API provides 2 way to register layouts:

- *.layouts.yml
- PHP class with annotations

See https://www.drupal.org/docs/8/api/layout-api/how-to-register-layouts for complete documentation.

Create a Layout Plugin (annotation method)

```
custom_layouts
  css
    two-column.css
  src
    Plugin
      Layout
        TwoColumnLayout.php
  templates
    layout--two-column.html.twig
  custom_layout.info.yml
  custom_layouts.libraries.yml
```

TwoColumnLayout.php

```php
<?php

namespace Drupal\custom_layouts\Plugin\Layout;

use Drupal\Core\Layout\LayoutDefault;
```

```php
/**
 * A custom, two-column layout.
 *
 * @Layout(
 *   id = "custom_two_column",
 *   label = @Translation("Custom: Two-column"),
 *   category = @Translation("Custom Layouts"),
 *   template = "templates/layout--two-column",
 *   library = "custom_layouts/two-column",
 *   regions = {
 *     "first" = {
 *       "label" = @Translation("First"),
 *     },
 *     "second" = {
 *       "label" = @Translation("Second"),
 *     }
 *   },
 *   icon_map = {
 *     {
 *        "first",
 *        "second"
 *     }
 *   }
 * )
 */
class TwoColumnLayout extends LayoutDefault {
```

TwoColumnLayout.php

```
 *    }
 * )
 */
class TwoColumnLayout extends LayoutDefault {
 // Override any methods you'd like to customize here!
}
```

layout--two-column.html.twig

```
{%
  set classes = [
    'layout',
    'layout--custom-two-column',
  ]
%}
{% if content %}
  <div{{ attributes.addClass(classes) }}>

    {% if content.first %}
      <div {{ region_attributes.first.addClass('layout__region', 'layout__region--first') }}>
        {{ content.first }}
      </div>
    {% endif %}

    {% if content.second %}
      <div {{ region_attributes.second.addClass('layout__region', 'layout__region--second')
}}>
        {{ content.second }}
      </div>
    {% endif %}

  </div>
{% endif %}
```

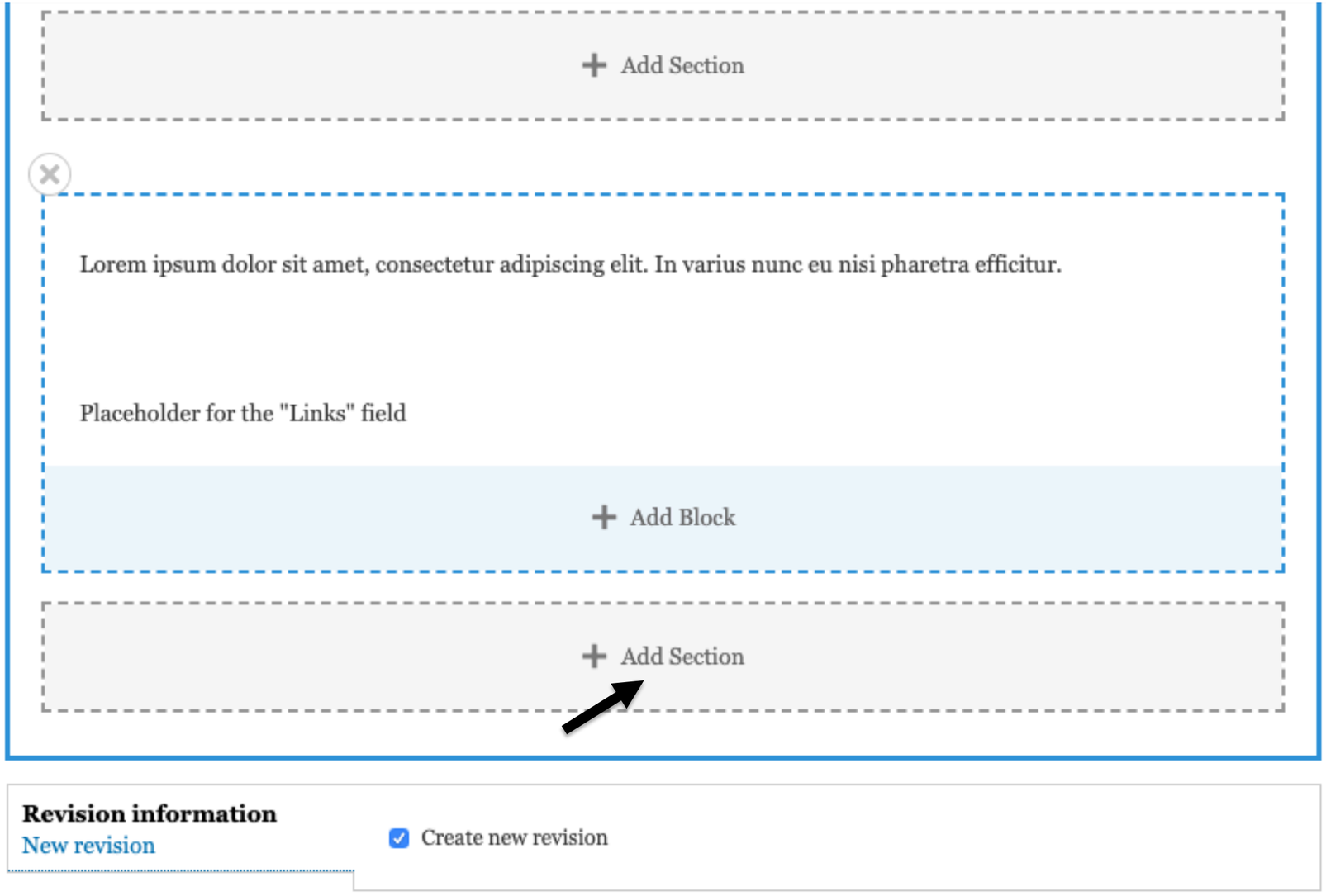
two-column.css

```
.layout--custom-two-column {
  display: grid;
  grid-template-columns: 50% 50%;
  grid-gap: 1em;
}

.layout--custom-two-column .layout__region--first,
.layout--custom-two-column .layout__region--second {
  padding: 1em;
}
```

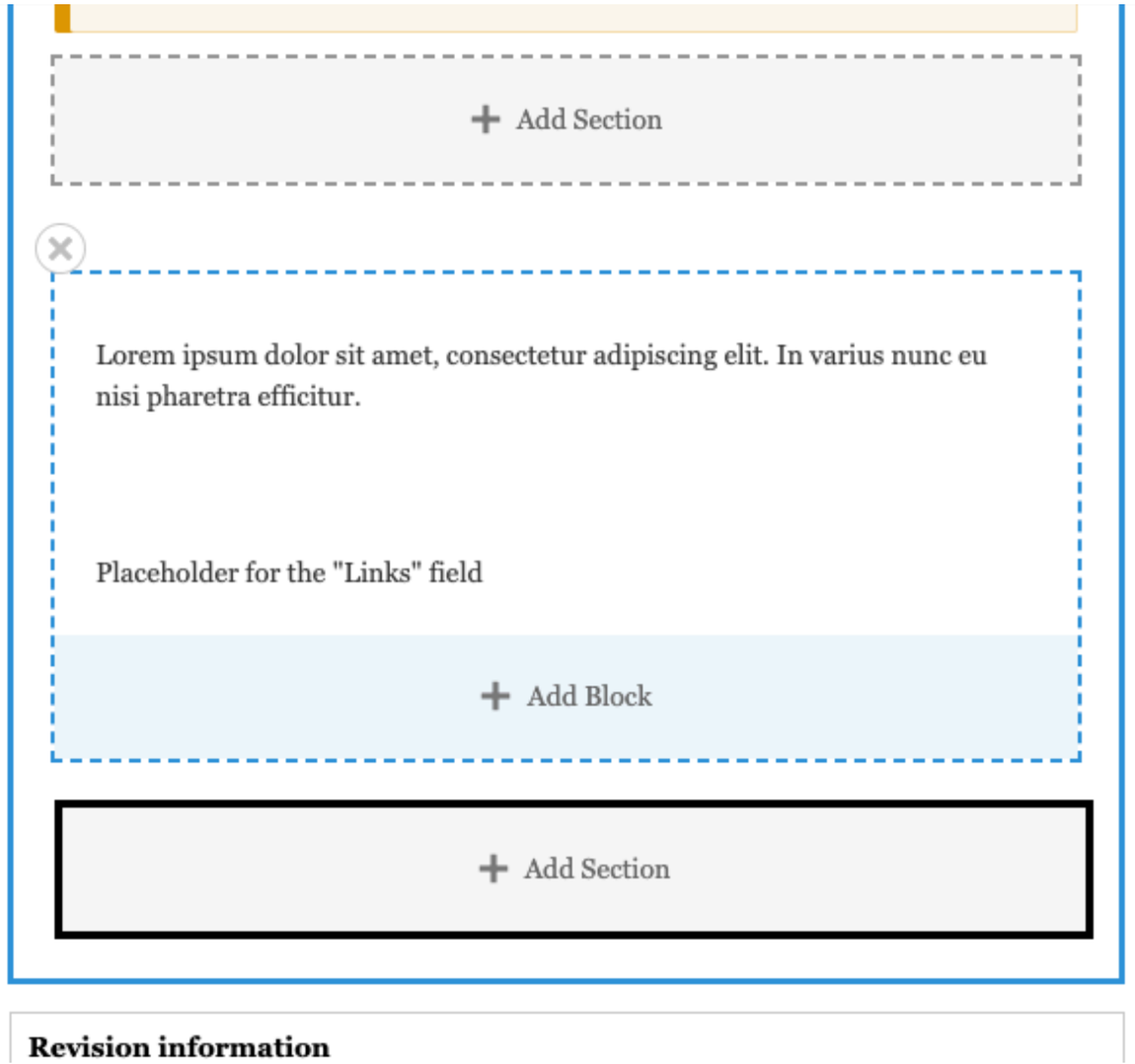
custom_layouts.libraries.yml

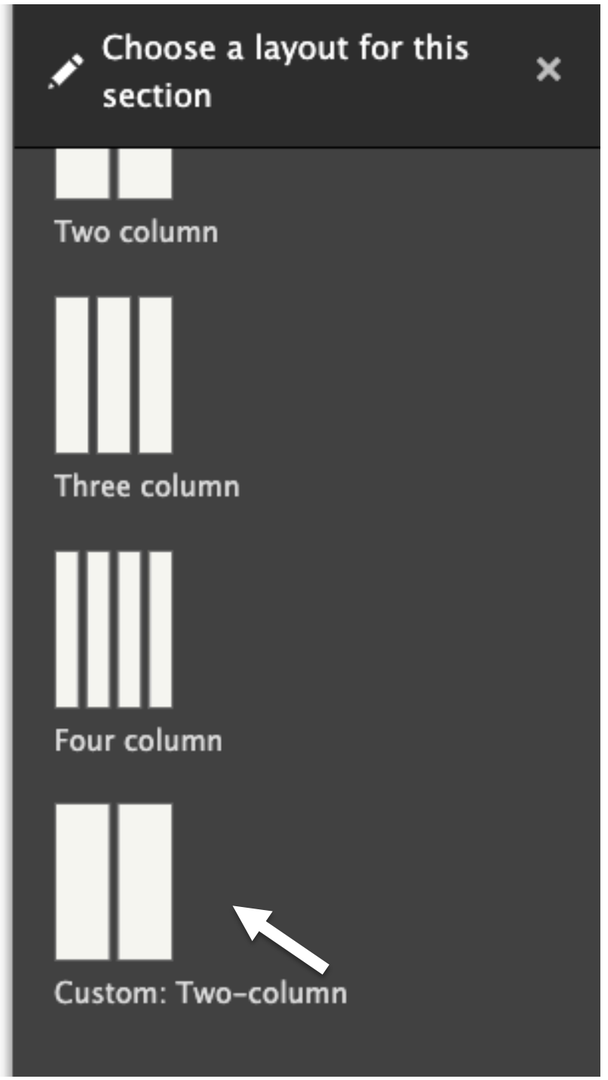
```
two-column:
  version: VERSION
  css:
    theme:
      css/two-column.css: {}
```

Create a your own layout

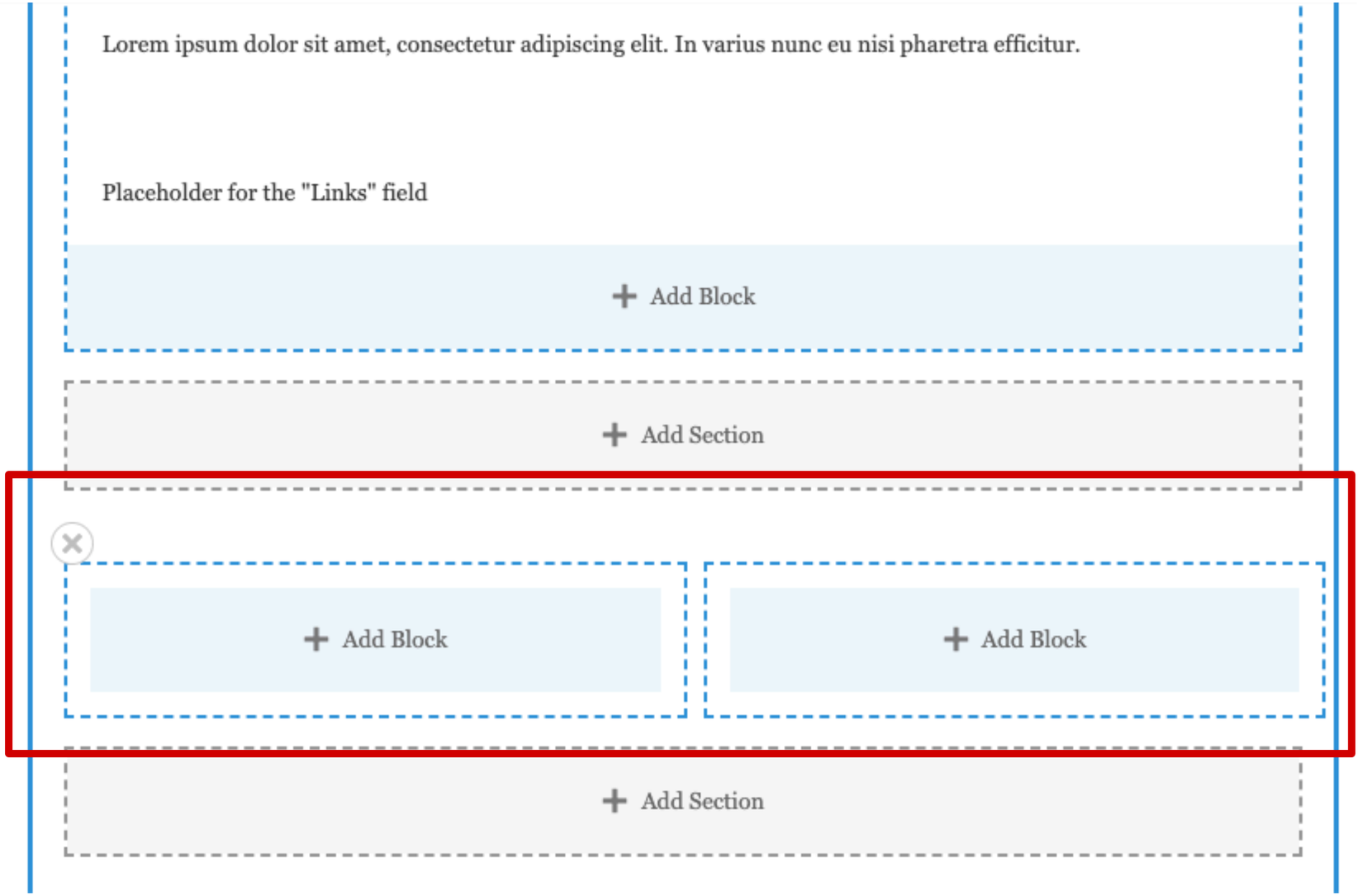


Create a your own layout





Create a your own layout



LayoutDefault Allows for Layout Settings:

- defaultConfiguration()
- buildConfigurationForm()
- validateConfigurationForm()
- submitConfigurationForm()

Add the following settings:

- Column widths (25-75, 50-50, 75-25)
- Layout custom classes

TwoColumnLayout.php

```
<?php

namespace Drupal\custom_layouts\Plugin\Layout;

use Drupal\Core\Form\FormStateInterface;
use Drupal\Core\Layout\LayoutDefault;
use Drupal\Core\Plugin\PluginFormInterface;

/**
 * A custom, two-column layout.



-------------------------------



 *   }
 * )
 */
class TwoColumnLayout extends LayoutDefault implements PluginFormInterface {
 // Override any methods you'd like to customize here!
}
```

buildConfigurationForm()

```
/**
 * {@inheritdoc}
 */
public function buildConfigurationForm(array $form, FormStateInterface $form_state) {
  $configuration = $this->getConfiguration();
  $form['column_widths'] = [
    '#type' => 'select',
    '#title' => $this->t('Column widths'),
    '#default_value' => $configuration['column_widths'],
    '#options' => [
      '25-75' => $this->t('25%/75%'),
      '50-50' => $this->t('50%/50%'),
      '75-25' => $this->t('75%/25%'),
    ],
  ];
  $form['extra_classes'] = [
    '#type' => 'textfield',
    '#title' => $this->t('Custom classes'),
    '#default_value' => $configuration['custom_classes'],
    '#description' => $this->t('Separate classes with a space'),
  ];
  return $form;
}
```

defaultConfiguration()

```
/**
 * {@inheritdoc}
 */
public function defaultConfiguration() {
  return parent::defaultConfiguration() + [
    'column_widths' => '50-50',
    'custom_classes' => '',
  ];
}
```

validateConfigurationForm()

```
/**
 * {@inheritdoc}
 */
public function validateConfigurationForm(array &$form, FormStateInterface $form_state) {
  $custom_classes = explode(' ', $form_state->getValue('custom_classes'));
  foreach ($custom_classes as $class) {
    $class_cleaned = Html::cleanCssIdentifier($class);
    if ($class != $class_cleaned) {
      $form_state->setError($form['custom_classes'],
        $this->t("'@class' is an invalid class value.", ['@class' => $class])
      );
      break;
    }
  }
}
```

Use patch #20 from https://www.drupal.org/project/drupal/issues/2897377; otherwise, form errors will not display.

submitConfigurationForm()

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

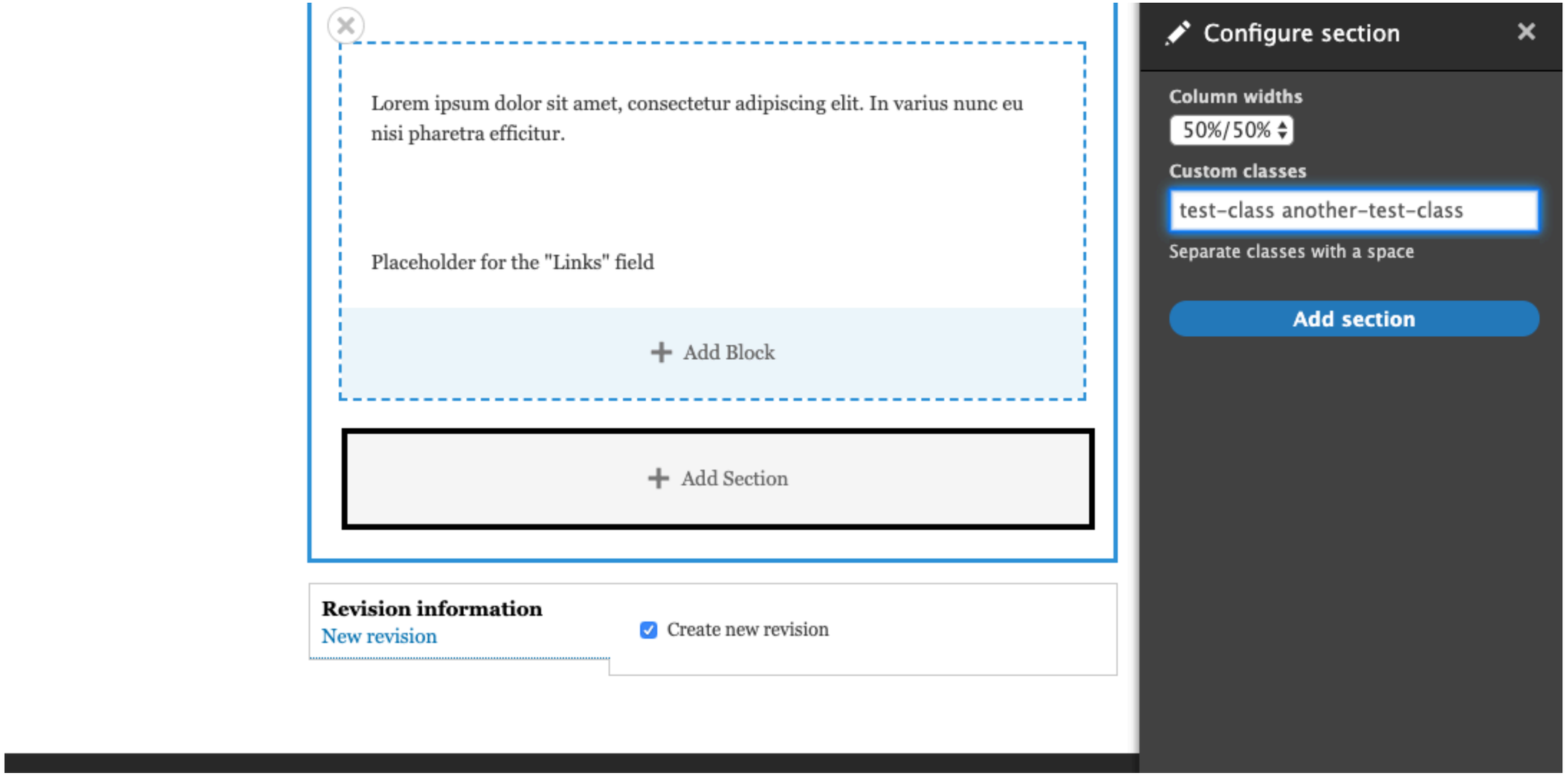
layout--two-column.html.twig

```
{%
  set classes = [
    'layout',
    'layout--custom-two-column',
    'layout--columns-' ~ settings.column_widths,
    settings.custom_classes,
  ]
%}
{% if content %}
  <div{{ attributes.addClass(classes) }}>

    {% if content.first %}
      <div {{ region_attributes.first.addClass('layout__region', 'layout__region--first') }}>
        {{ content.first }}
      </div>
    {% endif %}

    {% if content.second %}
      <div {{ region_attributes.second.addClass('layout__region', 'layout__region--second') }}>
        {{ content.second }}
      </div>
    {% endif %}

  </div>
{% endif %}
```

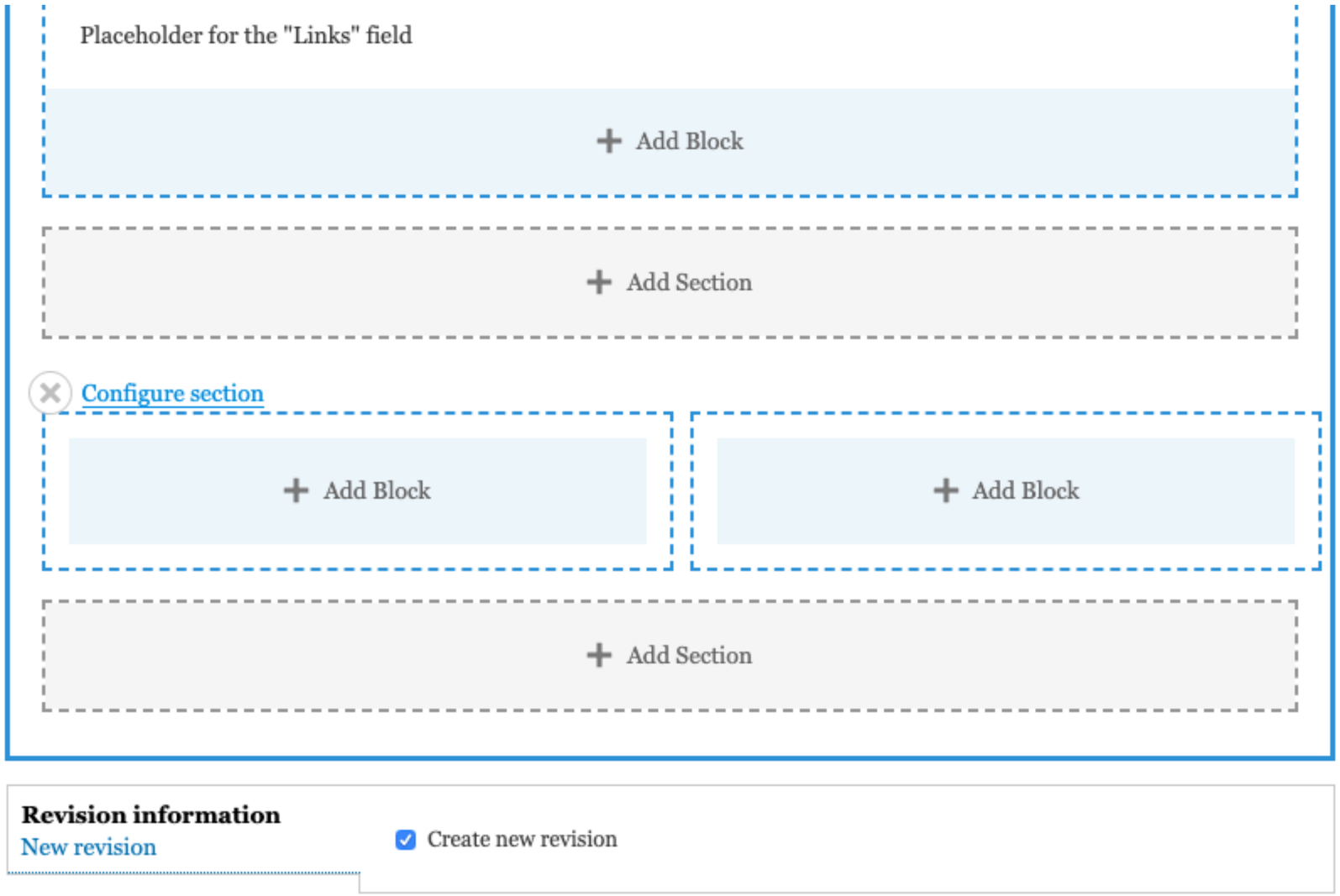
two-column.css

```
.layout--custom-two-column {
  display: grid;
  grid-template-columns: 50% 50%;
  grid-gap: 1em;
}

.layout--custom-two-column.layout--columns-25-75 {
  grid-template-columns: 25% 75%;
}

.layout--custom-two-column.layout--columns-50-50 {
  grid-template-columns: 50% 50%;
}

.layout--custom-two-column.layout--columns-75-25 {
  grid-template-columns: 75% 25%;
}

.layout--custom-two-column .layout__region--first,
.layout--custom-two-column .layout__region--second {
  padding: 1em;
}
```

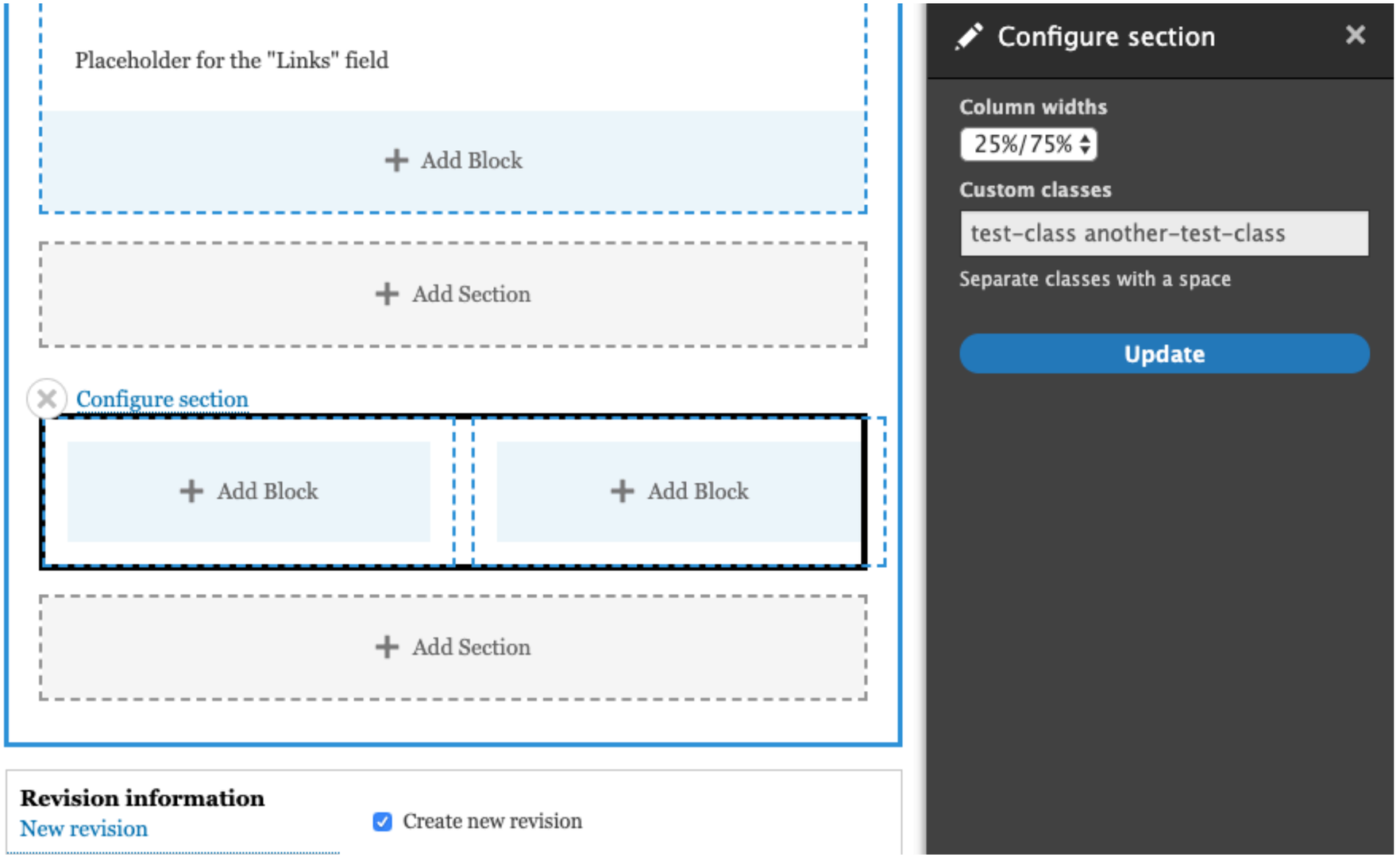
Add settings to a layout



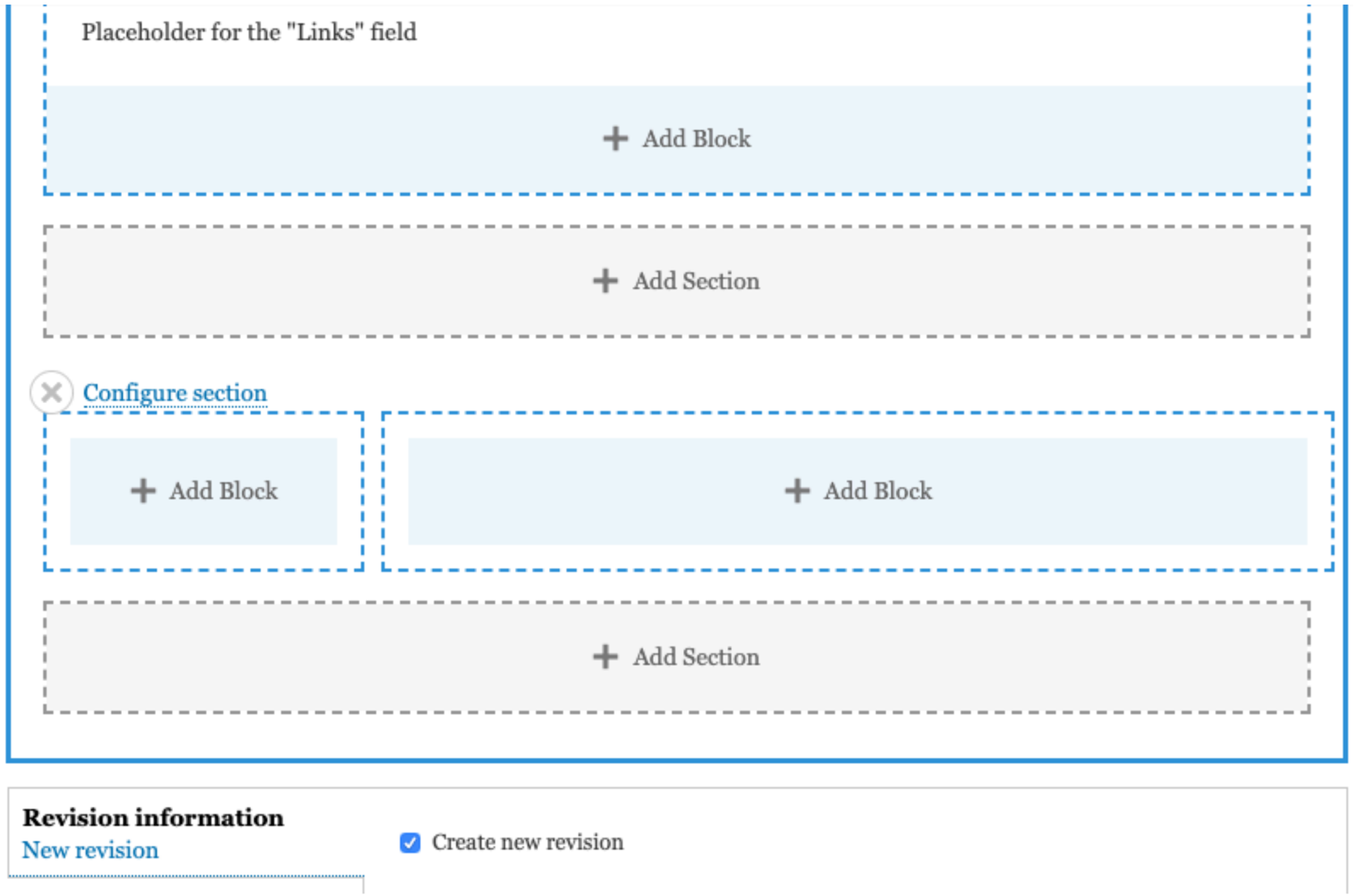
Add settings to a layout



Add settings to a layout



Add settings to a layout



Make settings available to block/field templates

Make settings available to block/field templates

custom_layouts.module

```
/**
 * Implements template_preprocess_layout().
 */
function custom_layouts_preprocess_layout(&$variables) {
  $custom_classes = (!empty($variables['settings']['custom_classes']))
    ? explode(' ', $variables['settings']['custom_classes']) : '';

  // Make custom classes array available to each block in the layout.
  if (isset($custom_classes) && !empty($custom_classes)) {
    // Loop through each region.
    foreach ($variables['content'] as $region_id => $region) {
      if (substr($region_id, 0, 1) !== '#') {
        // Loop through each block.
        foreach ($region as $block_id => $block) {
          if (substr($block_id, 0, 1) !== '#') {
            if (isset($variables['content'][$region_id][$block_id]['content']['#block_content'])) {
              $variables['content'][$region_id][$block_id]['content']['#block_content']
                ->__set('#custom_classes', $custom_classes);
            }
          }
        }
      }
    }
  }
}
```

Make settings available to block/field templates

custom_layouts.module

```
/**
 * Implements template_preprocess_block().
 */
function custom_layouts_preprocess_block(&$variables) {
  if (isset($variables['content']['#block_content'])) {
    $custom_classes = $variables['content']['#block_content']->__get('#custom_classes');
    if ($custom_classes) {
      $variables['data']['custom_classes'] = $custom_classes;
    }
  }
}
```

Make settings available to block/field templates

custom_layouts.module

```
/**
 * Implements template_preprocess_field().
 */
function custom_layouts_preprocess_field(&$variables) {
  $custom_classes = $variables['element']['#object']->__get('#custom_classes');
  if ($custom_classes) {
    foreach ($variables['items'] as $key => $value) {
      $variables['items'][$key]['data']['custom_classes'] = $custom_classes;
    }
  }
}
```

Make settings available to block/field templates

block.html.twig

```
{%
  set classes = [
    'block',
    'block-' ~ configuration.provider|clean_class,
    'block-' ~ plugin_id|clean_class,
  ]
%}
{% if 'test-class' in data.custom_classes %}
  {% set classes = classes|merge(['inverse']) %}
{% endif %}
<div{{ attributes.addClass(classes) }}>
  {{ title_prefix }}
  {% if label %}
    <h2{{ title_attributes }}>{{ label }}</h2>
  {% endif %}
  {{ title_suffix }}
  {% block content %}
    <div{{ content_attributes.addClass('content') }}>
      {{ content }}
    </div>
  {% endblock %}
</div>
```

For block type templates, use the Block Type Templates module:
https://www.drupal.org/project/block_type_templates

Make settings available to block/field templates

field.html.twig

```
{%
  set classes = [
    'field',
    'field--name-' ~ field_name|clean_class,
    'field--type-' ~ field_type|clean_class,
    'field--label-' ~ label_display,
  ]
%}
{% if 'test-class' in items.0.data.custom_classes %}
  {% set classes = classes|merge(['inverse']) %}
{% endif %}
{%
  set title_classes = [
    'field__label',
    label_display == 'visually_hidden' ? 'visually-hidden',
  ]
%}

{% if label_hidden %}
  {% if multiple %}
    <div{{ attributes.addClass(classes, 'field__items') }}>
```

Make settings available to block/field templates

layout--two-column.html.twig

Make settings available to block/field templates

block.html.twig

```
▼<div class="layout layout--custom-two-column layout--columns-25-75 test-class another-test-class">
  ▼<div class="layout__region layout__region--first">
    ▼<div data-layout-content-preview-placeholder-label=""Test Custom Block" block" class="block block-layout-builder block-inline-blockbasic inverse">
       <h2>Test Custom Block</h2>
      ▼<div class="content">
        ▶<div data-quickedit-field-id="block_content/1/body/en/full" class="clearfix text-formatted field field--name-body field--type-text-with-summary field--label-hidden inverse field__item">…</div>
       </div>
     </div>
   </div>
  </div>
```

```
{% if 'test-class' in data.custom_classes %}
  {% set classes = classes|merge(['inverse']) %}
{% endif %}
```

Make settings available to block/field templates

field.html.twig

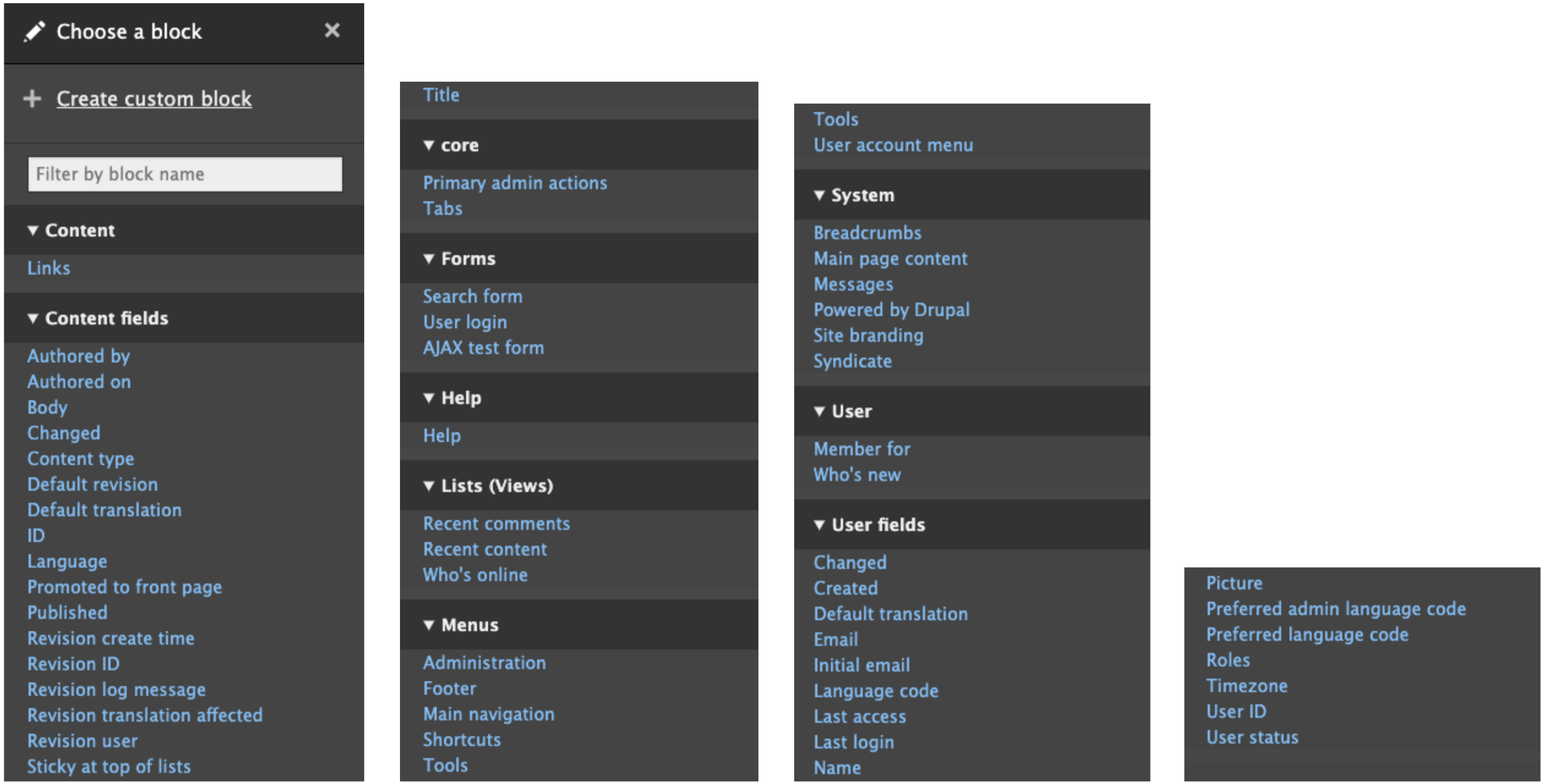
```
▼<div class="layout layout--custom-two-column layout--columns-25-75 test-class another-test-class">
  ▼<div class="layout__region layout__region--first">
    ▼<div data-layout-content-preview-placeholder-label=""Test Custom Block" block" class="block block-layout-builder block-inline-blockbasic inverse">
        <h2>Test Custom Block</h2>
      ▼<div class="content">
        ▶<div data-quickedit-field-id="block_content/1/body/en/full" class="clearfix text-formatted field field--name-body field--type-text-with-summary field--label-hidden inverse field__item">…</div>
        </div>
      </div>
    </div>
  </div>
```

```
{% if 'test-class' in items.0.data.custom_classes %}
  {% set classes = classes|merge(['inverse']) %}
{% endif %}
```

Layout Builder Restrictions

- Provides a UI for restricting layouts/blocks
- Built with plugins (so you can write your own)

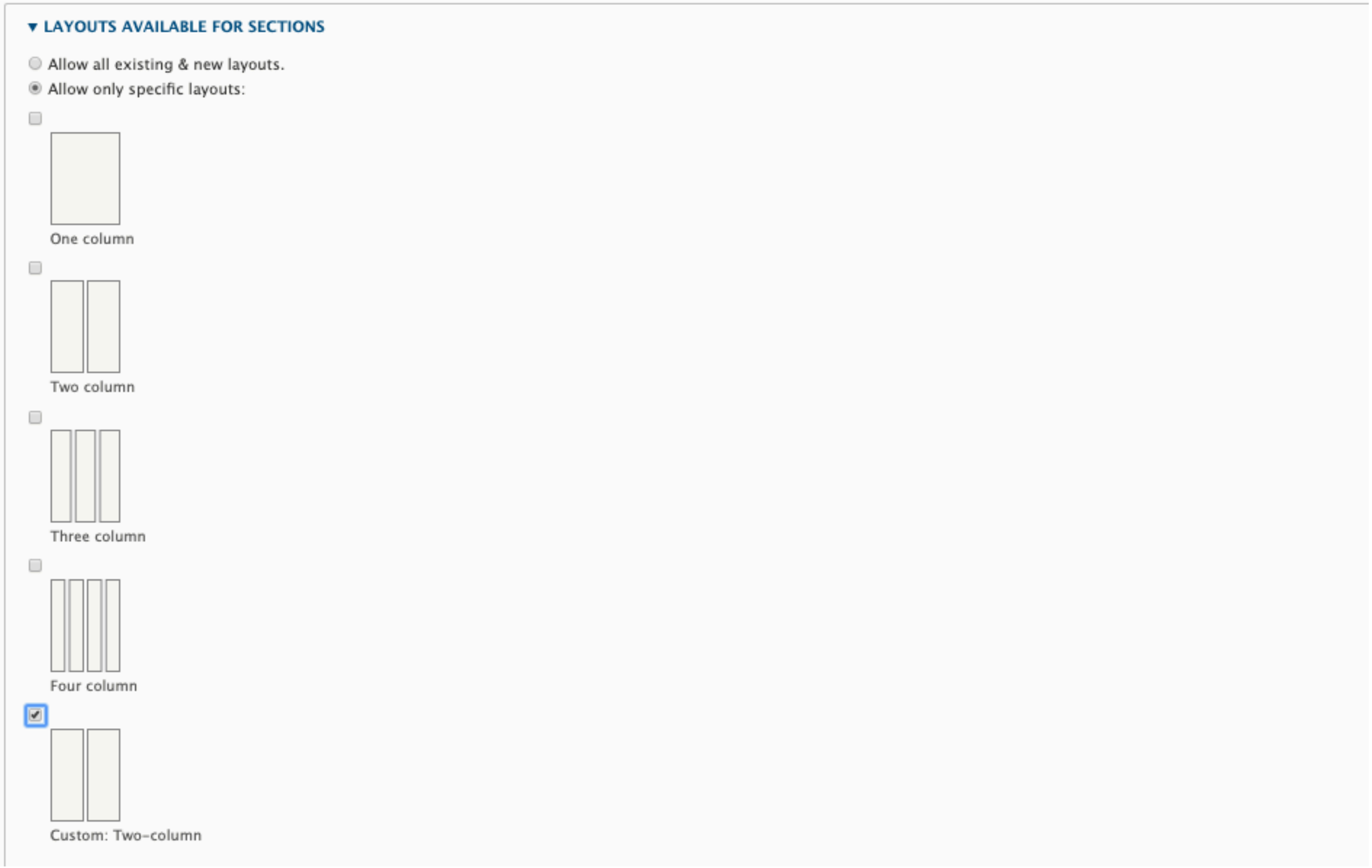
Without Layout Builder Restrictions



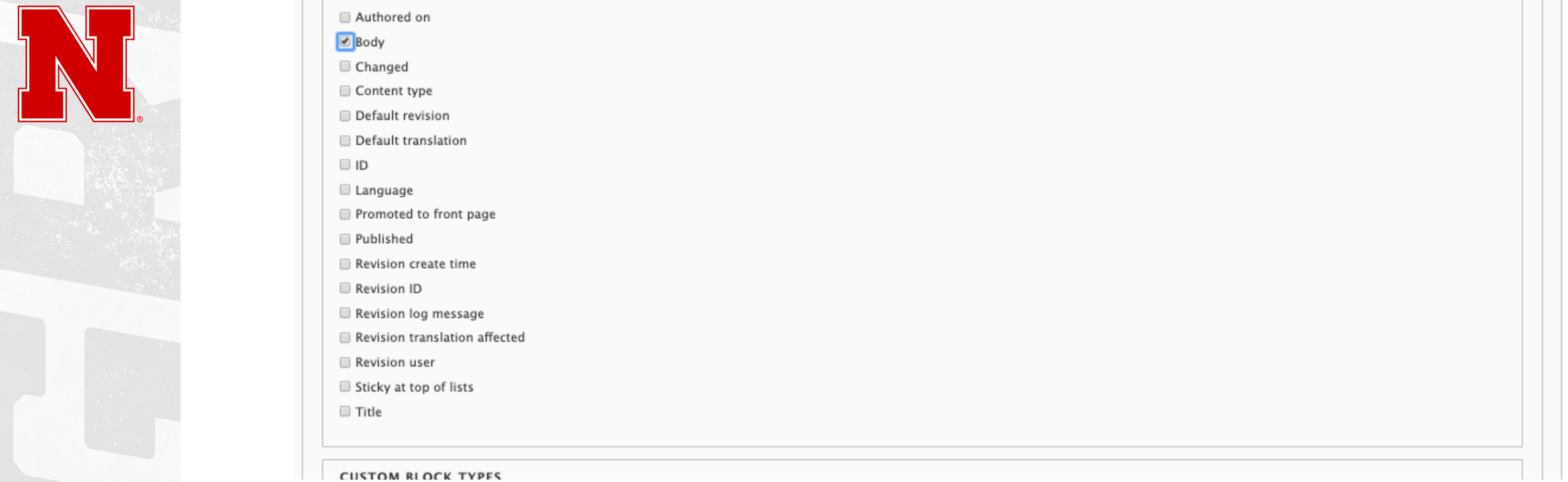
Layout Builder Restrictions

- Any view mode with Layout Builder enabled can be restricted

Create a Layout Builder Restrictions plugin



Create a Layout Builder Restrictions plugin



What does it take to be a plugin?

- At bare minimum, a plugin class
- Probably – configuration and a configuration UI

If you don’t need a UI, consider using hook_plugin_filter_TYPE__CONSUMER_alter() directly instead creating a Layout Builder Restrictions plugin

Step 1: Create a custom module with a plugin class

```
custom_lb_restriction
  src
    Plugin
      LayoutBuilderRestriction
        CustomRestriction.php
  custom_lb_restriction.info.yml
```

CustomRestriction.php

```
<?php

namespace Drupal\custom_lb_restriction\Plugin\LayoutBuilderRestriction;

use Drupal\layout_builder\SectionStorageInterface;
use Drupal\layout_builder_restrictions\Plugin\LayoutBuilderRestrictionBase;
use Drupal\layout_builder_restrictions\Traits\PluginHelperTrait;
use Symfony\Component\DependencyInjection\ContainerInterface;

/**
 * CustomRestriction Plugin.
 *
 * @LayoutBuilderRestriction(
 *   id = "custom_restriction",
 *   title = @Translation("Custom Layout Builder restriction")
 * )
 *
 */
class CustomRestriction extends LayoutBuilderRestrictionBase {

  use PluginHelperTrait;
```

CustomRestriction.php

```
/**
  * Constructs a Drupal\Component\Plugin\PluginBase object.
  *
  * @param array $configuration
  *   A configuration array containing information about the plugin instance.
  * @param string $plugin_id
  *   The plugin_id for the plugin instance.
  * @param mixed $plugin_definition
  *   The plugin implementation definition.
  */
 public function __construct(array $configuration, $plugin_id, $plugin_definition) {
   $this->configuration = $configuration;
   $this->pluginId = $plugin_id;
   $this->pluginDefinition = $plugin_definition;
 }

 /**
  * {@inheritdoc}
  */
 public static function create(ContainerInterface $container, array $configuration) {
   return new static(
     $configuration,
     $plugin_id,
     $plugin_definition
   );
 }
```

CustomRestriction.php

```
 /**
   * {@inheritdoc}
   */
  public function alterBlockDefinitions(array $definitions, array $context) {
    return $definitions;
  }

  /**
   * {@inheritdoc}
   */
  public function alterSectionDefinitions(array $definitions, array $context) {
    return $definitions;
  }

  /**
   * {@inheritdoc}
   */
  public function blockAllowedinContext(SectionStorageInterface $section_storage, $delta_from,
$delta_to, $region_to, $block_uuid, $preceding_block_uuid = NULL) {
    return TRUE;
  }
```

Step 2: Add config and UI

```
custom_lb_restriction
  config
    schema
      custom_lb_restriction.schema.yml
  src
    Form
      Settings.php
    Plugin
      LayoutBuilderRestriction
        CustomRestriction.php
  custom_lb_restriction.info.yml
  custom_lb_restriction.links.menu.yml
  custom_lb_restriction.routing.yml
```

Settings.php

```php
<?php

namespace Drupal\custom_lb_restriction\Form;

use Drupal\Core\Form\ConfigFormBase;
use Drupal\Core\Form\FormStateInterface;

/**
 * Configure example settings for this site.
 */
class Settings extends ConfigFormBase {
```

```php
/**
  * Config settings.
  *
  * @var string
  */
 const SETTINGS = 'custom_lb_restriction.settings';

 /**
  * {@inheritdoc}
  */
 public function getFormId() {
   return 'custom_lb_restriction_settings';
 }

 /**
  * {@inheritdoc}
  */
 protected function getEditableConfigNames() {
   return [
     static::SETTINGS,
   ];
 }
```

Settings.php

```
/**
  * {@inheritdoc}
  */
 public function buildForm(array $form, FormStateInterface $form_state) {
   $config = $this->config(static::SETTINGS);

   $form['disallow_views_blocks'] = [
     '#type' => 'checkbox',
     '#title' => $this->t('Disallow Views blocks on a global basis'),
     '#default_value' => $config->get('disallow_views_blocks'),
     '#description' => $this->t('If a block view is needed, a custom block plugin should be used.'),
   ];

   $form['layouts_restrict_to_custom'] = [
     '#type' => 'checkbox',
     '#title' => $this->t('Only allow Custom layouts'),
     '#default_value' => $config->get('layouts_restrict_to_custom'),
     '#description' => $this->t('Only Custom layouts will be allowed on a global basis.'),
   ];

   return parent::buildForm($form, $form_state);
 }
```

Settings.php

```
/**
  * {@inheritdoc}
  */
public function submitForm(array &$form, FormStateInterface $form_state) {
   // Retrieve the configuration.
   $config = $this->configFactory->getEditable(static::SETTINGS);
   // Set configuration value.
   $config->set('disallow_views_blocks',
     (bool) $form_state->getValue('disallow_views_blocks')
   );
   $config->set('layouts_restrict_to_custom',
     (bool) $form_state->getValue('layouts_restrict_to_custom')
   );
   // Save configuration.
   $config->save();

   parent::submitForm($form, $form_state);
 }
```

custom_lb_restriction.routing.yml

```
custom_lb_restriction.settings:
  path: '/admin/config/content/custom-lb-restriction-settings'
  defaults:
    _form: '\Drupal\custom_lb_restriction\Form\Settings'
    _title: 'Custom Layout Builder Restriction Settings'
  requirements:
    _permission: 'configure layout builder restrictions'
  options:
    _admin_route: TRUE
```

custom_lb_restriction.links.menu.yml

```
custom_lb_restriction.settings:
  title: 'Custom Layout Builder Restriction'
  route_name: custom_lb_restriction.settings
  parent: system.admin_config_content
  description: 'Configure the Custom Layout Builder Restriction module.'
  weight: 20
```

custom_lb_restriction.schema.yml

```
custom_lb_restriction.settings:
  type: config_object
  label: 'Custom Layout Builder Restriction settings'
  mapping:
    disallow_views_blocks:
      type: boolean
      label: 'Disallow Views blocks on a global basis'
    layouts_restrict_to_custom:
      type: boolean
      label: 'Only allow Custom layouts'
```

CustomRestriction.php

```
/**
  * {@inheritdoc}
  */
 public function alterBlockDefinitions(array $definitions, array $context) {
   // Retrieve config.
   $config = \Drupal::service('config.factory')
     ->get('custom_lb_restriction.settings');
   $disallow_views_block = $config
     ->get('disallow_views_blocks');

   if ($disallow_views_block) {
     foreach ($definitions as $id => $definition) {
       if (substr($id, 0, 11) === "views_block") {
         unset($definitions[$id]);
       }
     }
   }

   return $definitions;
 }
```

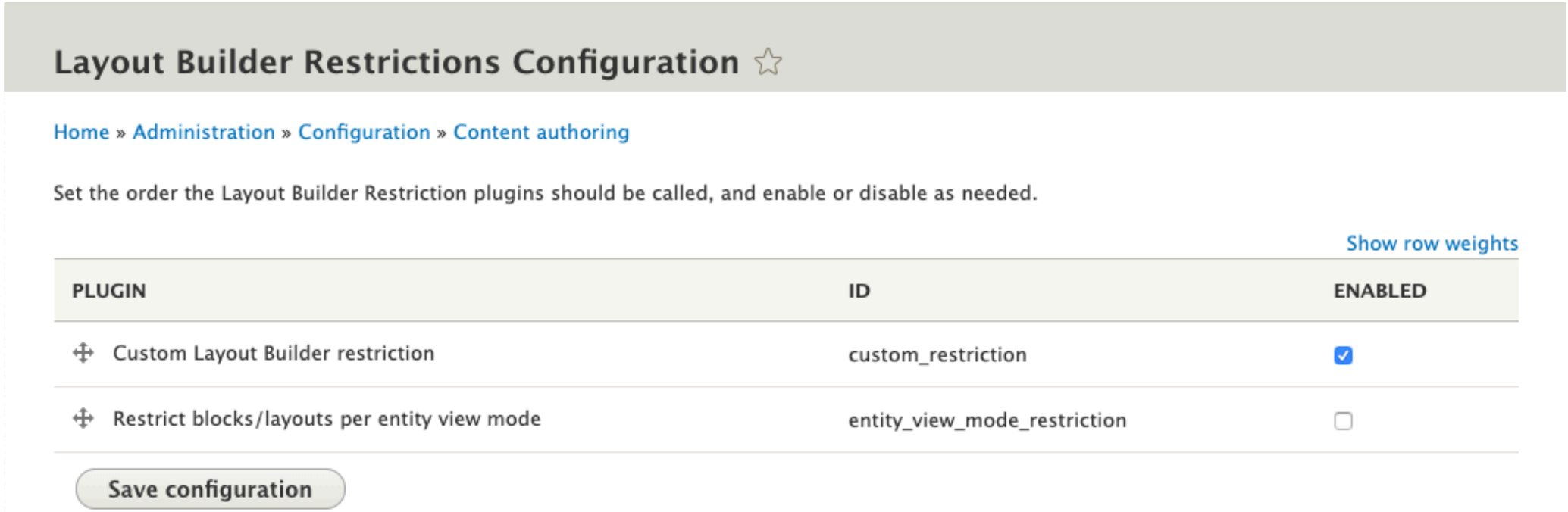
CustomRestriction.php

```
/**
  * {@inheritdoc}
  */
 public function alterSectionDefinitions(array $definitions, array $context) {
   // Retrieve config.
   $config = \Drupal::service('config.factory')
     ->get('custom_lb_restriction.settings');
   $layouts_restrict_to_custom = $config
     ->get('layouts_restrict_to_custom');

   if ($layouts_restrict_to_custom) {
     foreach ($definitions as $id => $definition) {
       if (substr($id, 0, 7) !== "custom_") {
         unset($definitions[$id]);
       }
     }
   }

   return $definitions;
 }
```

CustomRestriction.php

```
/**
  * {@inheritdoc}
  */
 public function blockAllowedinContext(SectionStorageInterface $section_storage,
   $delta_from, $delta_to, $region_to, $block_uuid, $preceding_block_uuid = NULL) {
   return TRUE;
 }
```

Enable the plugin



/admin/config/content/layout-builder-restrictions

Alter Layout Builder Block Forms

- It’s just hook_form_alter(), right?
- Nope

Alter Layout Builder Block Forms

- **Target two form IDs:**
  - layout_builder_add_block
  - layout_builder_update_block
- **Use a ‘process’ function**

Alter Layout Builder Block Forms

- **BUT – this only alter block forms inside Layout Builder**
- **Custom block forms are unaffected (block_content)**

Alter Layout Builder Block Forms

- Target ‘block_content_BUNDLE_edit_form’ and ‘block_content_BUNDLE_form’ forms

Block Form Alter

- hook_block_plugin_form_alter()
- hook_block_type_form_alter()

https://www.drupal.org/project/block_form_alter

hook_block_plugin_form_alter()

```
/**
 * Alter block forms per block plugin.
 *
 * Block forms for the 'block_content' and 'inline_content' plugins must use
 * hook_block_type_form_alter().
 *
 * @param array $form
 *   Nested array of form elements that comprise the form.
 * @param \Drupal\Core\Form\FormStateInterface $form_state
 *   The form state.
 * @param string $plugin
 *   The machine name of the plugin implementing the block.
 */
function hook_block_plugin_form_alter(array &$form, FormStateInterface &$form_state, string $plugin)
{
  if ($plugin == 'webform_block') {
    $form['settings']['redirect']['#default_value'] = TRUE;
    $form['settings']['redirect']['#disabled'] = TRUE;
  }
}
```

hook_block_type_form_alter()

```
/**
 * Alter custom block forms rendered by Block Content and Layout Builder.
 *
 * E.g. Alter block forms for 'block_content' and 'inline_block' plugins.
 *
 * @param array $form
 *   Nested array of form elements that comprise the form.
 * @param \Drupal\Core\Form\FormStateInterface $form_state
 *   The form state.
 * @param string $block_type
 *   The machine name of the custom block bundle.
 */
function hook_block_type_form_alter(array &$form, FormStateInterface &$form_state, string $block_type)
{
  if ($block_type == 'accordion') {
    $form['example_field']['widget'][0]['value']['#default_value'] = 'A better default value';
  }
}
```

Form element name attribute differs

Custom Block:

```
▼<div class="js-form-item form-item js-form-type-textfield form-type-textfield js-form-item-
field-my-field-0-value form-item-field-my-field-0-value">
    <label for="edit-field-my-field-0-value">My Field</label>
    <input class="js-text-full text-full form-text" data-drupal-selector="edit-field-my-field-0-
    value" type="text" id="edit-field-my-field-0-value" name="field_my_field[0][value]" value
    size="60" maxlength="255" placeholder> == $0
  </div>
</div>
```

Inline Block (Layout Builder):

```
▼<div class="js-form-item form-item js-form-type-textfield form-type-textfield js-form-item-
settings-block-form-field-my-field-0-value form-item-settings-block-form-field-my-field-0-value">
    <label for="edit-settings-block-form-field-my-field-0-value--7yv0YeivouY">My Field</label>
    <input class="js-text-full text-full form-text" data-drupal-selector="edit-settings-block-form-
    field-my-field-0-value" type="text" id="edit-settings-block-form-field-my-field-0-value-
    -7yv0YeivouY" name="settings[block_form][field_my_field][0][value]" value size="60" maxlength=
    "255" placeholder> == $0
  </div>
</div>
```

Form element name attribute differs

```
/**
 * Implements hook_block_type_form_alter().
 */
function my_block_type_form_alter(array &$form, FormStateInterface &$form_state, string $block_id) {
  if ($block_id == 'basic') {
    $form['field_my_field']['#states'] = [
      'required' => [
        ':input[name="field_require_my_field[value]"]' => ['checked' => TRUE],
        ':input[name="settings[block_form][field_require_my_field][value]"]' => ['checked' => TRUE],
      ],
    ];
  }
}
```

Relevant Issues on Drupal.org

Add visibility control conditions to blocks within Layout Builder
https://www.drupal.org/project/drupal/issues/2916876

Reorder Layout Builder sections
https://www.drupal.org/project/drupal/issues/3080606

[PP-1] Reverting entity revisions that contain custom blocks erroneously triggers EntityChangedConstraint
https://www.drupal.org/project/drupal/issues/3053881

Relevant Issues on Drupal.org

Multiple select element is inappropriately targeted with CSS in settings tray
https://www.drupal.org/project/drupal/issues/3080698

It's very difficult to alter forms of inline (content blocks) placed via Layout Builder
https://www.drupal.org/project/drupal/issues/3028391

$form['#attributes']['data-drupal-selector'] is set to a random value so can't be used for its intended purpose
https://www.drupal.org/project/drupal/issues/2897377

Questions?