

Layout Builder and Drupal.offCanvas

An intro with extra tricks and no extra modules

DrupalSouth 2022 Brisbane

Joshua Graham

Technical Developer

Digital and Emerging Technology



The better the question. The better the answer.
The better the world works.



About the Speaker



Joshua Graham
Technical Developer
EY Digital | Digital and Emerging Technology

Originally from Wagga Wagga NSW, I graduated From Charles Sturt University with Bachelor of IT before moving to Canberra. I have been developing websites with Drupal for over six years and with the same organisation - then Adelphi Digital now EY.

I have worked various technologies including, GovCMS Drupal 7/8/9, Migrate Plus, Apache SOLR, Docker/Lando, Active Directory and Hyper-V. Websites I have developed include the Defence International Training Centre and State of the Environment Report 2021.

I enjoy having coffee with friends, Saturday parkruns, playing soccer and board games.

Agenda

1. View modes
2. Drupal core field layout
3. Layout builder
4. Why not use Display suite?
5. Field groups & Revisions (a note about that...)
6. Customisation: Adding html classes
7. Customisation: Sections and their editing form
8. Drupal.offCanvas
9. Layout builder & Drupal.offCanvas
10. Demo
11. Questions

Timings

Overview - 26 minutes

Demo - 4 minutes

Questions: 5 minutes

View modes

A view mode is a way in that a specific layout is used to display a content entity in a specific context.

For example:

- ▶ On the homepage you could have a 'news' node showing only a title and a teaser of the body.
- ▶ But on its full page view, show it's title, an image and the full body.

Configurations that can be different per view mode:

1. Field visibility, order, formatter and grouping
2. Template
3. Which preprocess is run
4. And which display plugin is used (etc.)

View mode A



View mode B



<https://www.drupal.org/docs/drupal-apis/entity-api/display-modes-view-modes-and-form-modes>

Entity View Display

The original out-of the box way to display fields in a view mode. It's simple and no-frills.

Good:

- ▶ Can show/hide/reorder fields

Bad

- ▶ Limited to only showing Field UI added configuration fields, can't add a block between instead of fields and can't repeat a field.
- ▶ Can't group fields in an html element without extra contributed modules or a template file override

Limited Workarounds

- ▶ Can't re-order extra fields, but can add current page node's field directly to the whole page's block layout (but only if ctools_blocks is enabled – it comes with pathauto)

Layout builder

Introduced in 8.5 and marked stable in 8.7, Layout builder is a plugin to create and display layouts visually from the administration interface for a view mode.

Features:

- ▶ Can show/hide/reorder fields
- ▶ Can add any “block” as field on display between fields
- ▶ Can add a Drupal core views module block to be able to:
 - ▶ Group fields in a html element.
 - ▶ Show dynamic text etc.
- ▶ Allows a preview of the layout
- ▶ Can override the placement of fields of per entity (must use the same base layout)
- ▶ Can use in the GovCMS distribution

<https://www.drupal.org/project/drupal/releases/8.7.0>

<https://www.drupal.org/docs/drupal-apis/layout-api/how-to-register-layouts>

Display Suite (why not?)

Display Suite is a contributed module which provides a layout plugin for view modes.

Why not – an opinion: It's heavy handed, it's features are quite specific, has issues outstanding and its an extra module to upgrade with Drupal core.

Pros/Cons:

- ▶ Can show/hide/reorder fields
- ▶ Can show dynamic text via token plugin (but has caching issues) *
- ▶ Can add any “block” as a field on display (but node ID context not isn't known.)
- ▶ Can group fields into a html element if also using contrib field_group module
- ▶ No preview of layout
- ▶ Extra field logic is duplicated instead of reused (e.g. book navigation)
- ▶ Deprecated in GovCMS

* Has caching issues if using a Block display plugin without a Lazy Builder (e.g. context module)

<https://www.drupal.org/project/context/issues/2919934> | <https://www.drupal.org/project/ds/issues/2939322> | <https://www.drupal.org/project/ds/issues/3174246>

Layout builder - Set for a node display

Example:

How to configure layout builder for the default view mode for Node:

1. Enable the Layout Builder module
2. Go to the default view mode
3. Check "Use Layout Builder"

Home > Administration > Structure > Content types > Basic page > Manage display




Manage display

Edit Manage fields Manage form display **Manage display**

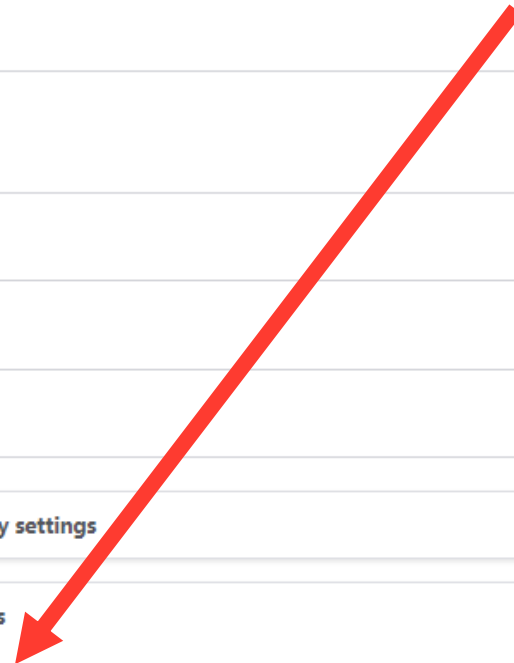
Default Teaser

Content items can be displayed using different view modes: Teaser, Full content, Print, RSS, etc. 7

Here, you can define which fields are shown and hidden when *Basic page* content is displayed in

Field	Label
 Body	- Hidden
 Second Body	Above
 Links	
Disabled	
No field is hidden.	
Custom display settings	
Layout options	
<input type="checkbox"/> Use Layout Builder	

Save



Layout builder - Manage layout

Once enabled, click “Manage layout”

The screenshot shows the 'Manage display' interface in Drupal. At the top, there is a breadcrumb trail: Home > Administration > Structure > Content types > Basic page > Manage display. Below this is the 'Manage display' title with a star icon. A navigation bar contains four tabs: 'Edit', 'Manage fields', 'Manage form display', and 'Manage display' (which is active and underlined). Below the tabs are two sub-tabs: 'Default' and 'Teaser'. A green status message box at the top left says 'Status message: Your settings have been saved.' Below this, a text block explains: 'Content items can be displayed using different view modes: Teaser, Full content, Print, RSS, etc. Teaser Here, you can define which fields are shown and hidden when Basic page content is displayed in each view mode.' A red arrow points from the text 'Here, you can define which fields are shown and hidden' to a grey button labeled 'Manage layout'. Below the button are two expandable sections: 'Custom display settings' (collapsed) and 'Layout options' (expanded). The 'Layout options' section contains two checkboxes: 'Use Layout Builder' (checked) and 'Allow each content item to have its layout customized.' (unchecked). At the bottom right is a blue 'Save' button.

Layout builder – Editing screen

Dummy content:

- ▶ Toggle “Show content preview” to switch between dummy content and field name

Reorder:

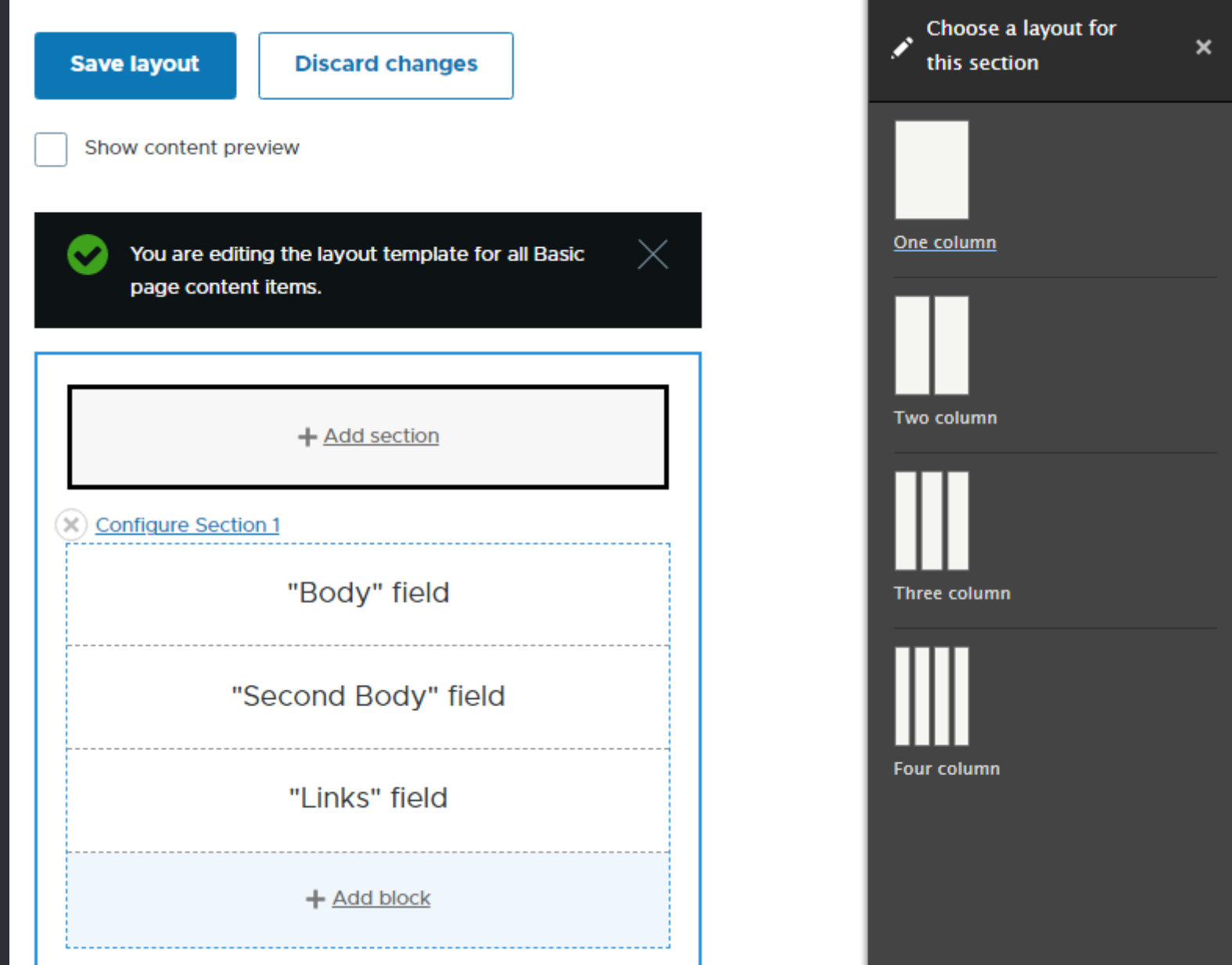
- ▶ Drag and Drop the dotted rectangles to re-order fields

Add section

- ▶ Click “Add section” to add additional layouts to the view mode.
- ▶ Available view modes are shown on the right

Save layout

- ▶ Click “save layout” to save the changes



Configure Section 1

"Image" field

"Body" field

"Tags" field

+ Add block

"Comments" field

+ Add block

+ Add section

Configure section

Column widths

33%/67% ▾

Choose the column widths for this layout.

Administrative label

Update

Field groups & Revisions (a note about that...)

Layout builder supports revisions for top level fields (hurray!)

Sometimes you want a HTML element around two or more elements, commonly referred to as a “field group” for the field_group module that’s is used, but it cannot be used as the field_group module is not compatible with layout builder.

So without wanting to write a preprocess hook or override a twig template.... There are two common options:

1. Use a View block with Content ID for the block context. You also add extra custom html if you use the rewrite option but retrieving the current revision isn’t supported natively.
2. Use the contributed ctools module since you are likely to have pathauto installed anyway and it’s a dependency, to show the same entity in a different view mode in a nested fashion. This supports revisions too. (okay I lied, a small preprocess hook is required to hide the nested title, or you could CSS hide it)

https://www.drupal.org/project/field_group/issues/3176317

Field groups & Revisions (view block setup)

How to setup a view block with content ID context - use the same method as block layout.

=> Add "Content: ID" contextual filter.

Displays

Block* + Add

Add Contextual filter - Content ID

Edit view name/description ▼

Display name: [Block](#)

Duplicate Block ▼

Title

Title: [Layout Builder View](#)

Format

Format: [Unformatted list](#) | [Settings](#)

Show: [Fields](#) | [Settings](#)

Fields

[Content: Title](#)

Filter criteria

[Content: Published \(= Yes\)](#)

Sort criteria

[Content: Authored on \(desc\)](#)

Block settings

Block name: [None](#)

Block category: [Lists \(Views\)](#)

Allow settings: [Items per page](#)

Access: [Permission](#) | [View published content](#)

Header

Add

Footer

Add

No results behavior

Add

Pager

Use pager: [Display a specified number of items](#) | [1 item](#)

More link: [No](#)

Advanced

Contextual filters

[Content: ID](#)

Relationships

Add

Exposed form

Exposed form in block: [No](#)

Exposed form style: [Basic](#) | [Settings](#)

Other

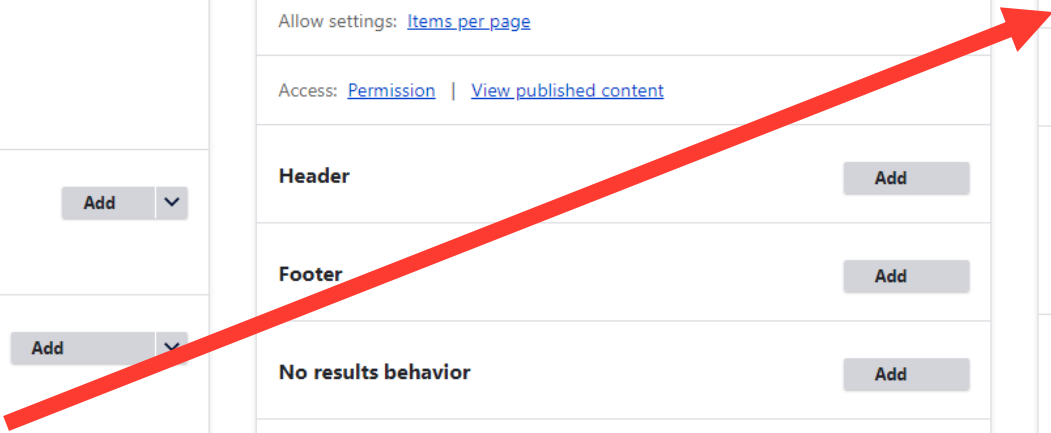
Machine Name: [block_1](#)

Administrative comment: [None](#)

Use AJAX: [No](#)

Hide attachments in summary: [No](#)

Contextual links: [Shown](#)



Field groups & Revisions (view block setup cont.)

Tick "Specify validation criteria" and select "Content" from the dropdown.

Configure contextual filter: Content: ID

^ When the filter value is *NOT* available

- ☐ Display all results for the specified field
- ☐ Provide default value
- ☒ Hide view
- ☐ Display a summary
- ☐ Display contents of "No results found"
- ☐ Display "Access Denied"

Exceptions

☐ Skip default argument for view URL
Select whether to include this default argument when constructing the URL for this view. Skipping default arguments is useful e.g. in the case of feeds.

^ When the filter value *IS* available or a default is provided

- ☐ Override title
- ☒ Specify validation criteria

Validator

Content

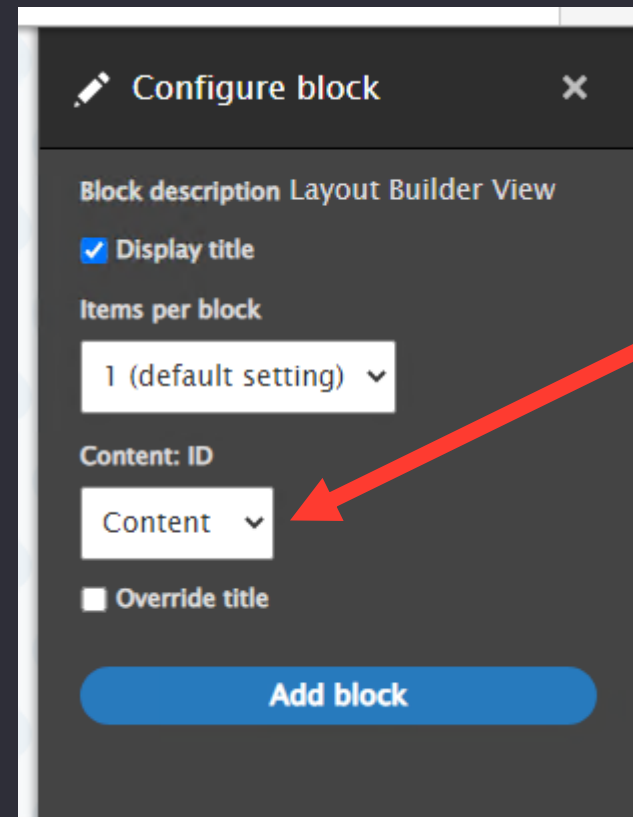
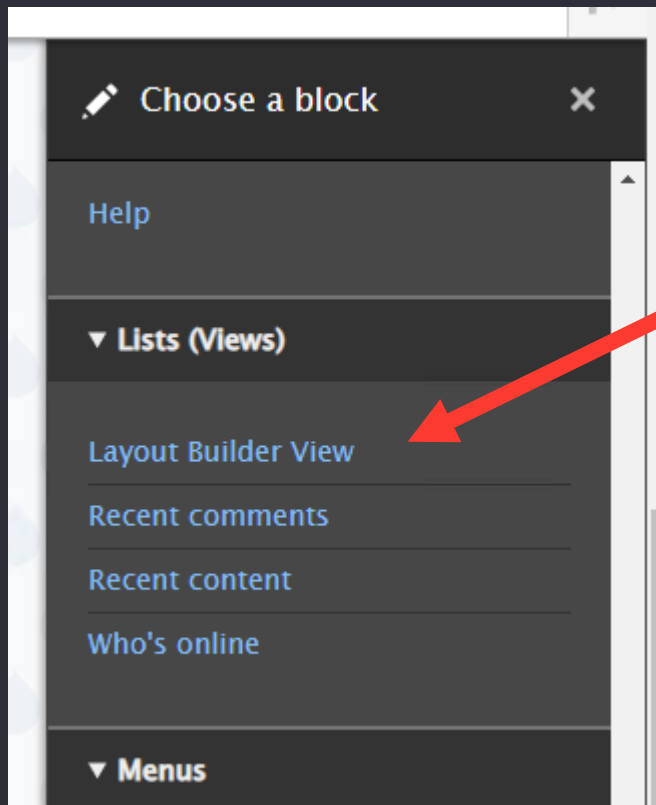
Specify validation criteria - Content

Field groups & Revisions (view block setup cont.)

Then add the views block to the layout builder manage display screen.

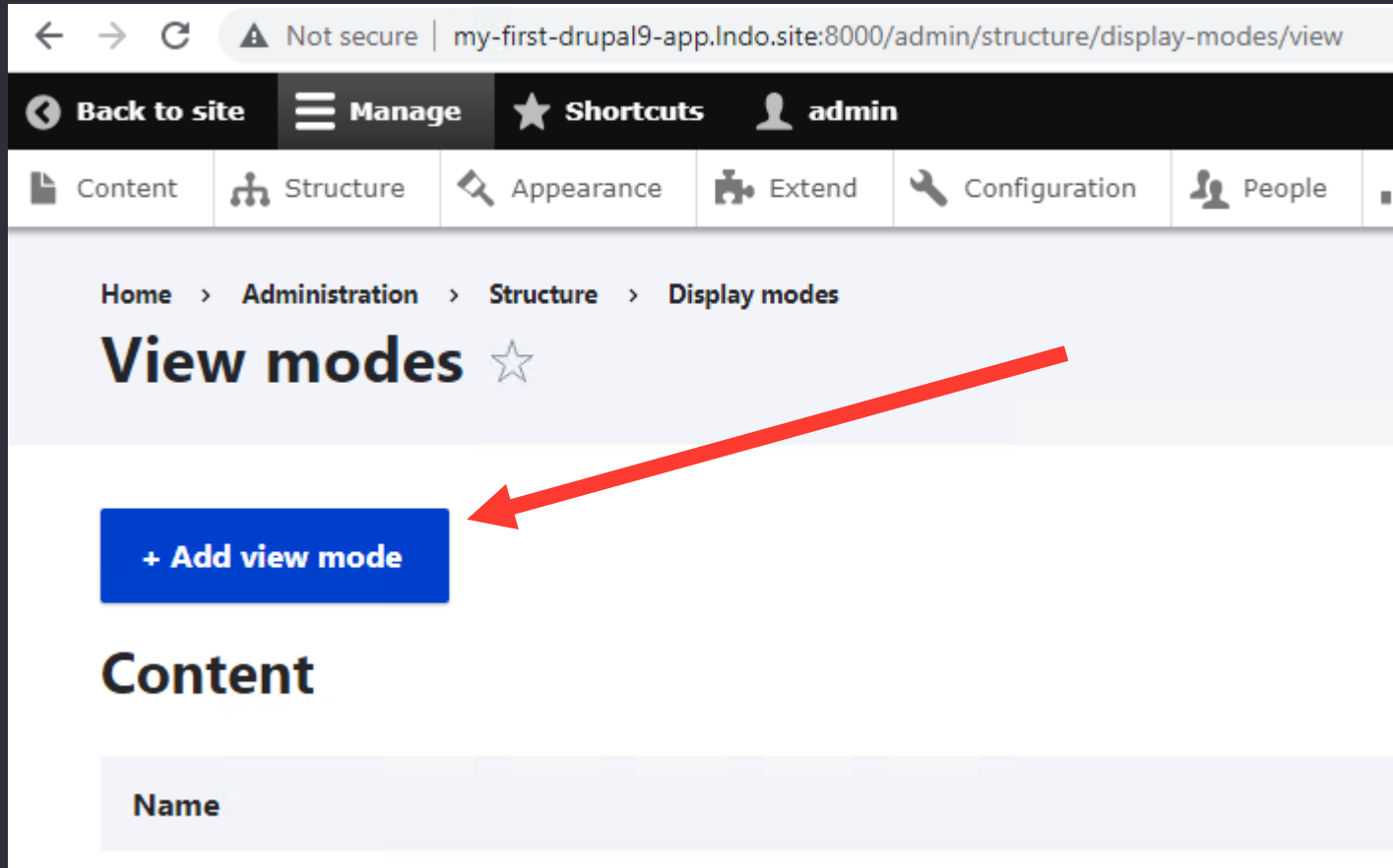
Ensure the context of "Content" is selected to be passed.

My example view is called "Layout Builder view", the name is could be anything though.



Field groups & Revisions (entity view setup)

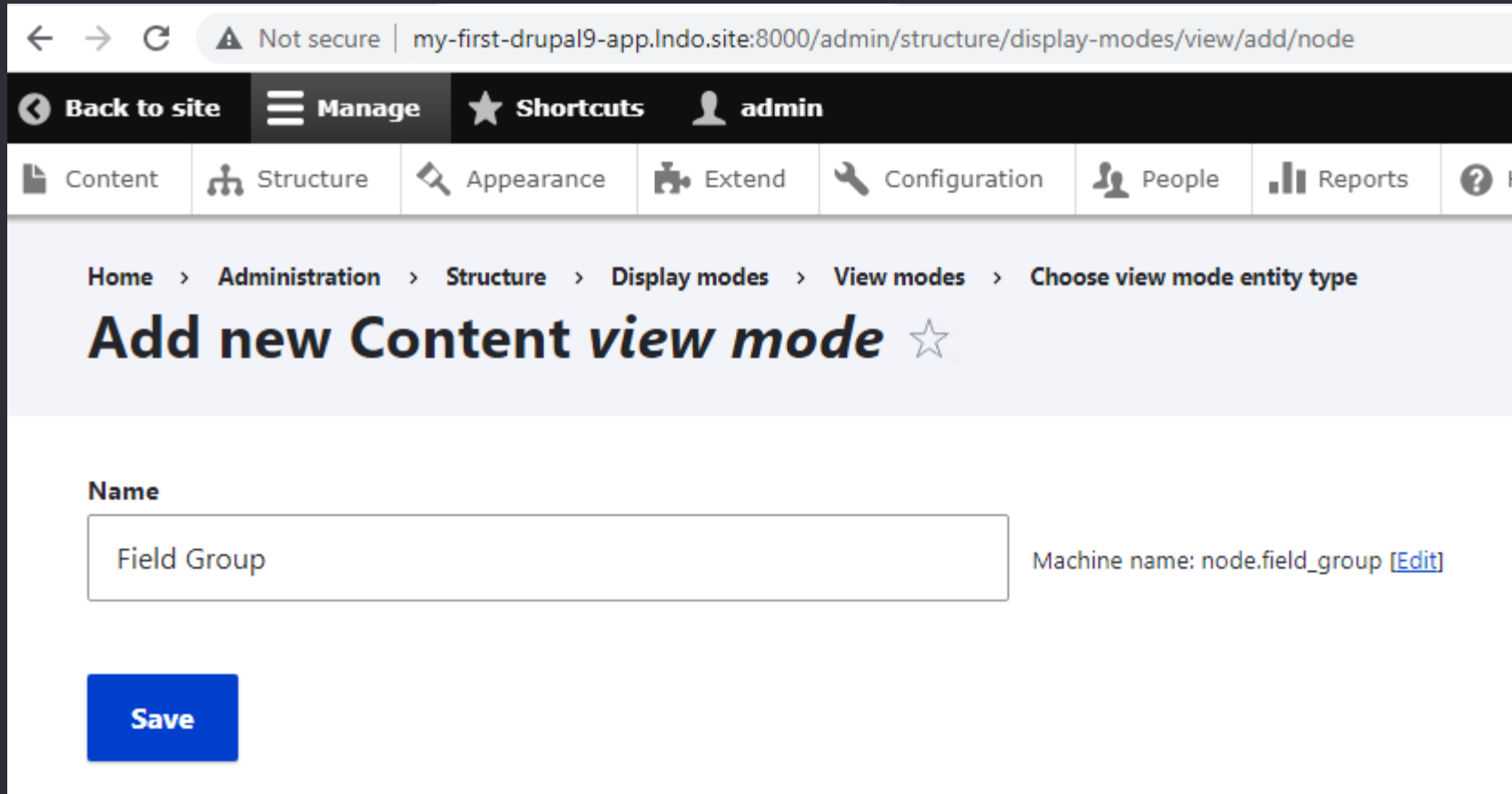
How to setup a field group via ctools module (because you have already pathauto installed right?)
=> First click "Add view mode".



Field groups & Revisions (entity view setup cont.)

Type the name of the new view mode.

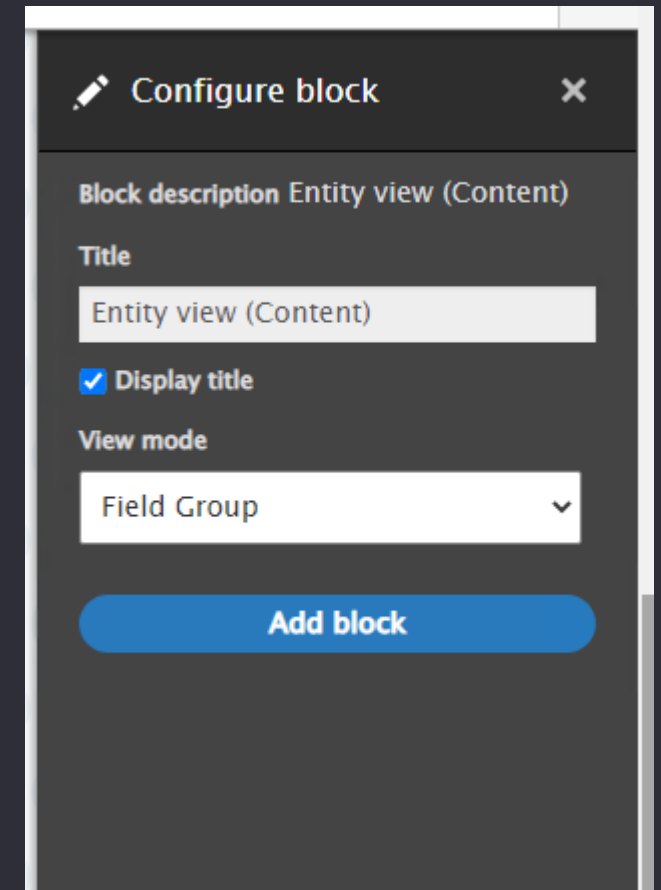
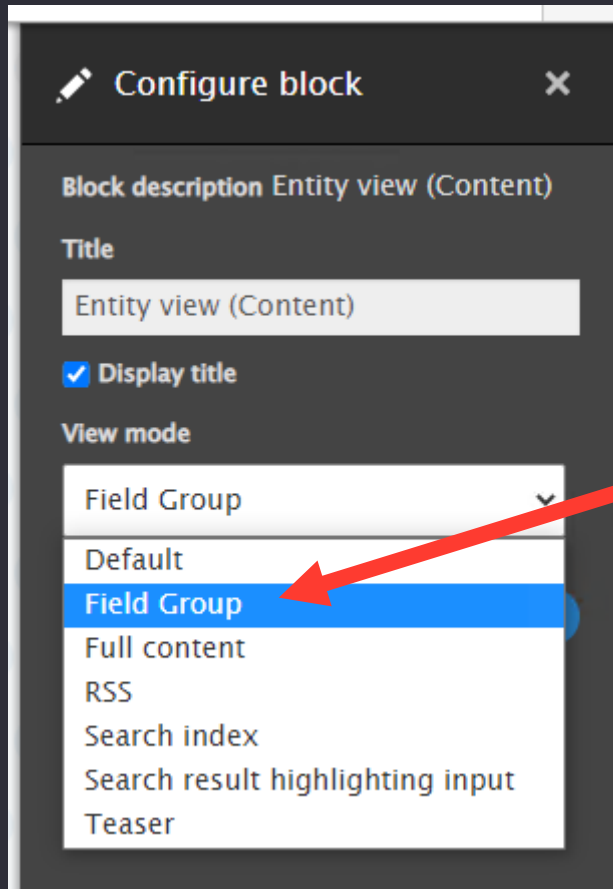
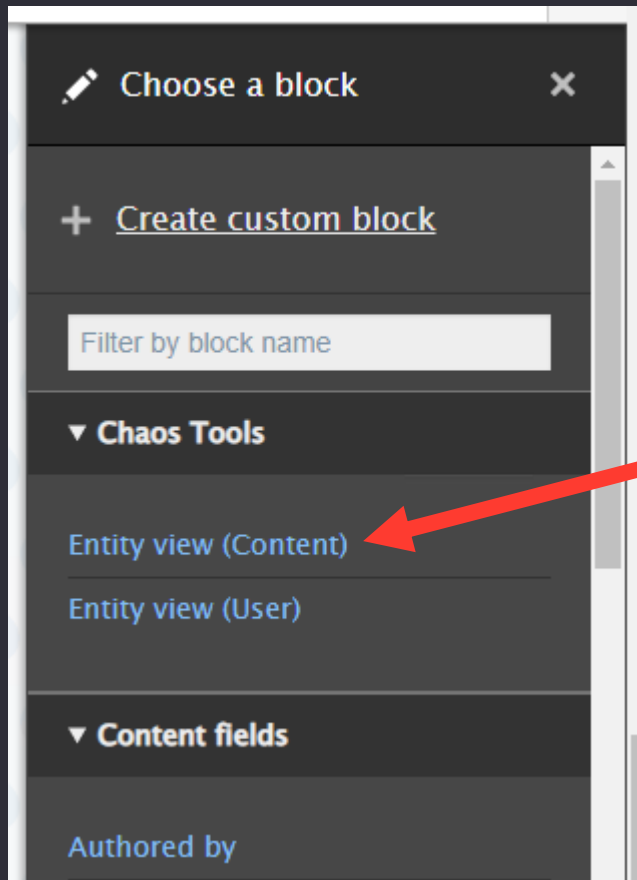
My example view mode is called "Field Group", the name is could be anything though.



The screenshot shows the Drupal 9 admin interface. The browser address bar displays the URL: `my-first-drupal9-app.lndo.site:8000/admin/structure/display-modes/view/add/node`. The top navigation bar includes links for **Back to site**, **Manage**, **Shortcuts**, and the user **admin**. Below this is a horizontal menu with icons and labels for **Content**, **Structure**, **Appearance**, **Extend**, **Configuration**, **People**, **Reports**, and a help icon. The breadcrumb trail reads: **Home > Administration > Structure > Display modes > View modes > Choose view mode entity type**. The main heading is **Add new Content *view mode*** with a star icon. The form contains a **Name** label, a text input field with the value **Field Group**, and a **Machine name** field showing `node.field_group` with an [\[Edit\]](#) link. A blue **Save** button is located at the bottom left of the form.

Field groups & Revisions (entity view setup cont.)

On the layout builder screen, select the Chaos tools (ctools) block “Entity view (content)” and then set the view mode to your new one. (Don’t forget to configure your new nested view mode!)



Field groups & Revisions (entity view setup cont.)

How to Hide duplicate title caused by nesting a node withing a node.

Template: node--my-view-mode.html.twig | node--my-bundle--my-view-mode.html.twig

Preprocess hook: hook_preprocess_node()

```
73 {% extends "node.html.twig" %}
74
75 {# Do not print heading, set page to true. #}
76 {% set page = true %}
```

```
1  .node .node .node__title {
2      display: none;
3  }
```

```
8  /**
9   * Implements hook_preprocess_node().
10  */
11  function my_theme_preprocess_node(&$variables, $hook) {
12      $view_modes_to_hide_title = [
13          'view_mode_to_hide',
14          'another_view_mode_to_hide',
15      ];
16      if (in_array($variables['view_mode'], $view_modes_to_hide_title)) {
17          $variables['page'] = TRUE;
18      }
19  }
```

Customisation: Adding html classes

HTML classes can be added to layout builder such the content type and view mode classes which are missing on each section. Otherwise CSS selectors can get unnecessarily long.

```
1  .node--type-page > .node__content > .layout {  
2      /* CSS Rules */  
3  }
```



```
5  .layout--node--type--page {  
6      /* CSS Rules */  
7  }
```

Customisation: Adding html classes

How to add classes to sections:

Copy: `web/core/modules/layout_discovery/layouts/onecol/layout--onecol.html.twig`

To: `web/themes/[my_theme]/templates/custom/layout/layout--custom--onecol.html.twig`

(Can be in any location in templates folder but path must match `layouts.yml` "path" key location)

Create: `web/themes/[my_theme]/my_front.layouts.yml`

Create: `web/themes/my_theme/src/Plugin/Layout/CustomDefaultLayoutPlugin.php`

(Can be in any location in src folder but namespace must match `layouts.yml` "class" key)

Customisation: Adding html classes (cont.)

Contents: web/themes/[my_theme]/templates/custom/layout/layout--custom--onecol.html.twig

No changes required from the original template.

```
13 {%
14     set classes = [
15         'layout',
16         'layout--onecol',
17     ]
18 %}
19 {% if content %}
20     <div{{ attributes.addClass(classes) }}>
21         <div {{ region_attributes.content.addClass('layout__region', 'layout__region--content') }}>
22             {{ content.content }}
23         </div>
24     </div>
25 {% endif %}
```

Customisation: Adding html classes (cont.)

Contents: web/themes/my_front/my_front.layouts.yml

```
1  layout__custom__onecol:
2    label: 'Custom: One column'
3    path: templates/custom/layout
4    template: layout--custom--onecol
5    class: '\Drupal\my_front\Plugin\Layout\CustomDefaultLayoutPlugin'
6    library: layout_discovery/onecol
7    category: 'Columns: 1'
8    default_region: content
9    icon_map:
10      - [content]
11    regions:
12      content:
13        label: Content
```

Customisation: Adding html classes (cont.)

Contents: web/themes/my_front/src/Plugin/Layout/CustomDefaultLayoutPlugin.php

```
1  <?php
2
3  namespace Drupal\my_front\Plugin\Layout;
4
5  use Drupal\Core\Entity\Display\EntityDisplayInterface;
6  use Drupal\Core\Form\FormStateInterface;
7  use Drupal\Core\Form\SubformStateInterface;
8  use Drupal\Core\Layout\LayoutDefault;
9  use Drupal\Core\Plugin\PluginFormInterface;
10 use Drupal\layout_builder\DefaultsSectionStorageInterface;
11
12 /**
13  * Custom default layout class plugin.
14  */
15 class CustomDefaultLayoutPlugin extends LayoutDefault implements PluginFormInterface {
16     // Code.
17 }
```


Customisation: Adding html classes (cont.)

Contents: web/themes/my_front/src/Plugin/Layout/CustomDefaultLayoutPlugin.php

```
17 ▼ /**
18    * {@inheritdoc}
19    */
20 ▼ public function defaultConfiguration() {
21 ▼    return parent::defaultConfiguration() + [
22        'custom_entity_bundle' => '',
23        'custom_entity_type' => '',
24        'custom_entity_view_mode' => '',
25    ];
26 }
```

Customisation: Adding html classes (cont.)

Contents: web/themes/my_front/src/Plugin/Layout/CustomDefaultLayoutPlugin.php

I added an optional display only form item as the section form does not show which layout is being used.

```
28 ▼ /**
29    * {@inheritdoc}
30    */
31 ▼ public function buildConfigurationForm(array $form, FormStateInterface $form_state) {
32    $form = parent::buildConfigurationForm($form, $form_state);
33    // Optional. Add section name, so we know what we are editing.
34 ▼    $form['custom_display_only_title'] = [
35        '#type' => 'item',
36 ▼        '#title' => $this->t('@title <br /> (%machine_name)', [
37            '@title' => $this->getPluginDefinition()->getLabel(),
38            '%machine_name' => $this->getPluginId(),
39        ]),
40    ];
41    return $form;
42 }
```

Customisation: Adding html classes (cont.)

Contents: web/themes/my_front/src/Plugin/Layout/CustomDefaultLayoutPlugin.php

```
50  /**
51   * {@inheritdoc}
52   */
53  public function validateConfigurationForm(array &$form, FormStateInterface $form_state) {
54      parent::validateConfigurationForm($form, $form_state);
55  }
56
57  /**
58   * {@inheritdoc}
59   */
60  public function submitConfigurationForm(array &$form, FormStateInterface $form_state) {
61      parent::submitConfigurationForm($form, $form_state);
62      $extra_properties = $this->getCustomExtraProperties($form_state);
63      $this->configuration['custom_entity_bundle'] = $extra_properties['bundle'];
64      $this->configuration['custom_entity_type'] = $extra_properties['entity_type'];
65      $this->configuration['custom_entity_view_mode'] = $extra_properties['view_mode'];
66  }
```

Customisation: Adding html classes (cont.)

Contents: web/themes/my_front/src/Plugin/Layout/CustomDefaultLayoutPlugin.php

```
106     protected function getCustomExtraProperties(FormStateInterface $fs) {
107         $return_options = ['bundle' => '', 'entity_type' => '', 'view_mode' => ''];
108         /** @var Drupal\Core\Form\SubformState $complete_form_state */
109         $sub_form_state = ($fs instanceof SubformStateInterface) ? $fs : $fs->getCompleteFormState();
110         $buid_info = $sub_form_state->getBuildInfo();
111         $args = isset($buid_info['args']) ? $buid_info['args'] : [];
112         /** @var Drupal\layout_builder\DefaultsSectonStorageInterface $section_storage */
113         $section_storage = NULL;
114         foreach ($args as $arg) {
115             if ($arg instanceof DefaultsSectionStorageInterface) {
116                 $section_storage = $arg;
117                 break;
118             }
119         }
120     }
```

Customisation: Adding html classes (cont.)

Contents: web/themes/my_front/src/Plugin/Layout/CustomDefaultLayoutPlugin.php

```
120 ▼ if ($section_storage && ($context_values = $section_storage->getContextValues())) {
121 ▼     if (isset($context_values['display'])
122         && ($context_values['display'] instanceof EntityDisplayInterface)) {
123         $return_options['bundle'] = $context_values['display']->getTargetBundle();
124         $return_options['entity_type'] = $context_values['display']->getTargetEntityTypeId();
125     }
126     if (isset($context_values['view_mode'])) {
127         $return_options['view_mode'] = $context_values['view_mode'];
128     }
129 }
130 return $return_options;
131 }
```

Customisation: Adding html classes (cont.)

Contents: web/themes/my_front/src/Plugin/Layout/CustomDefaultLayoutPlugin.php

```
62  /**
63   * {@inheritdoc}
64   */
65  public function build(array $regions) {
66      $build = parent::build($regions);
67      // Extract short variable names.
68      $entity_type = $this->configuration['custom_entity_type'];
69      $bundle = $this->configuration['custom_entity_bundle'];
70      $view_mode = $this->configuration['custom_entity_view_mode'];
71      // Ensure array exists.
72      $build['#attributes'] = $build['#attributes'] ?? [];
73      $build['#attributes']['class'] = $build['#attributes']['class'] ?? [];
74      // Add Classes.
75      $build['#attributes']['class'][] = $this->getPluginDefinition()->getTemplate();
76      $build['#attributes']['class'][] = 'layout';
77      $build['#attributes']['class'][] = "layout--{$entity_type}";
78      $build['#attributes']['class'][] = "layout--{$entity_type}--type--{$bundle}";
79      $build['#attributes']['class'][] = "layout--{$entity_type}--view-mode--{$view_mode}";
80      return $build;
81  }
```

Customisation: Add field html classes

The html classes on the usual fields are not there. How to add them back:

```
8  /**
9   * Implements hook_preprocess_block().
10  */
11  function my_front_preprocess_block(&$variables, $hook) {
12    // Layout builder logic.
13    if (\Drupal::moduleHandler()->moduleExists('layout_builder') && isset($variables['plugin_id'])) {
14      // Add classes to field block.
15      if ((strpos($variables['plugin_id'], 'field_block:') === 0) && isset($variables['content']['#theme'])
16          && ($variables['content']['#theme'] == 'field')) {
17
18        $variables['attributes']['class'][] = "lb--field--name-{$variables['content']['#field_name']}";
19        $variables['attributes']['class'][] = "lb--field--type-{$variables['content']['#field_type']}";
20        $variables['attributes']['class'][] = "lb--field--label-{$variables['content']['#label_display']}";
21      }
22    }
23  }
```

Customisation: Layout builder sections and their form

Additional options such custom extra HTML classes can be added to each layout builder section.

Note: Layouts yml and template twig files are the same as before. PHP class declarations are the same.

Contents: web/themes/my_front/src/Plugin/Layout/CustomDefaultLayoutPlugin.php

```
18 ▼ /**
19    * {@inheritdoc}
20    */
21 ▼ public function defaultConfiguration() {
22     return parent::defaultConfiguration() + [
23         'custom_extra_classes' => '',
24     ];
25 }
```


Customisation: Layout builder sections and their form

Contents: web/themes/my_front/src/Plugin/Layout/CustomDefaultLayoutPlugin.php

```
27 ▼ /**
28    * {@inheritdoc}
29    */
30 ▼ public function buildConfigurationForm(array $form, FormStateInterface $form_state) {
31     $form = parent::buildConfigurationForm($form, $form_state);
32     $configuration = $this->getConfiguration();
33 ▼     $form['custom_extra_classes'] = [
34         '#type' => 'textfield',
35         '#title' => $this->t('Custom: Extra classes'),
36         '#default_value' => $configuration['custom_extra_classes'],
37     ];
38     return $form;
39 }
```

Customisation: Layout builder sections and their form

Contents: web/themes/my_front/src/Plugin/Layout/CustomDefaultLayoutPlugin.php

```
41  /**
42   * {@inheritdoc}
43   */
44  public function validateConfigurationForm(array &$form, FormStateInterface $form_state) {
45      parent::validateConfigurationForm($form, $form_state);
46  }
47
48  /**
49   * {@inheritdoc}
50   */
51  public function submitConfigurationForm(array &$form, FormStateInterface $form_state) {
52      parent::submitConfigurationForm($form, $form_state);
53      $this->configuration['custom_extra_classes'] = $form_state->getValue('custom_extra_classes');
54  }
```

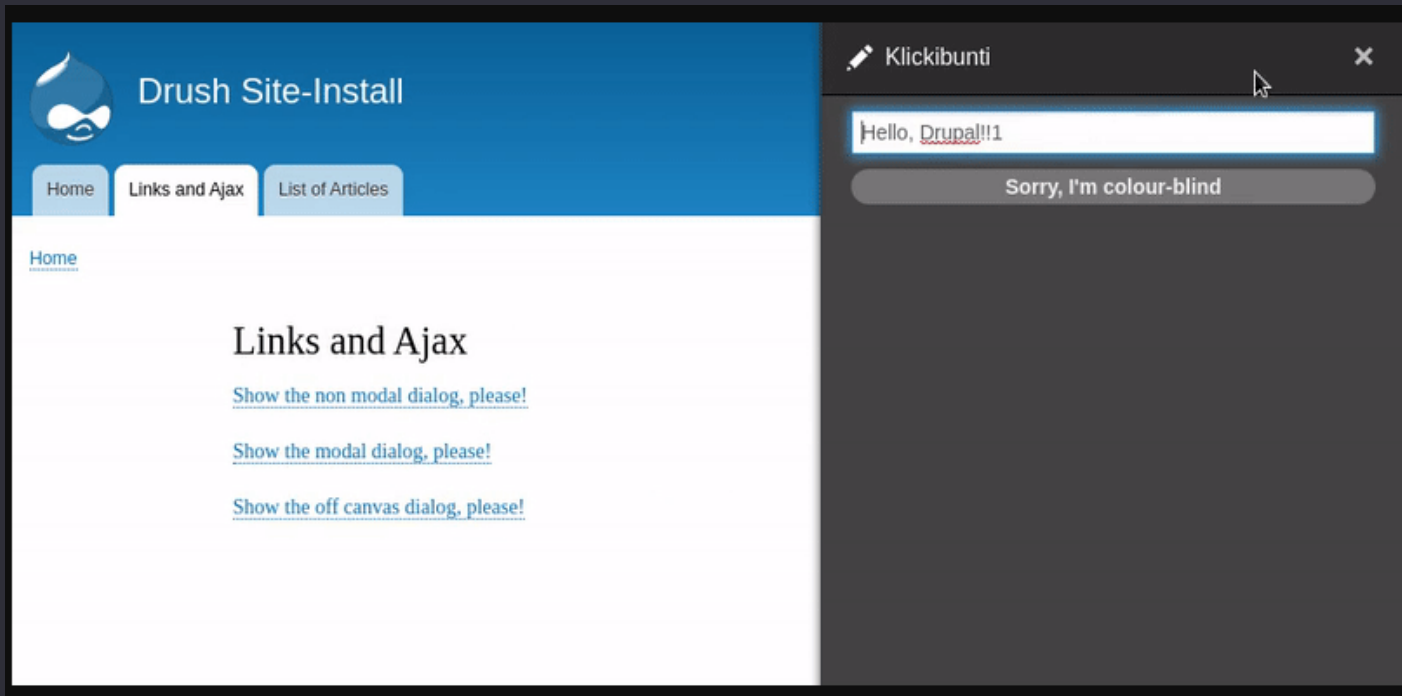
Customisation: Layout builder sections and their form

Contents: web/themes/my_front/src/Plugin/Layout/CustomDefaultLayoutPlugin.php

```
57 ▼ /**
58     * {@inheritdoc}
59     */
60 ▼ public function build(array $regions) {
61     $build = parent::build($regions);
62
63     // Ensure array exists.
64     $build['#attributes'] = $build['#attributes'] ?? [];
65     $build['#attributes']['class'] = $build['#attributes']['class'] ?? [];
66
67     // Add extra classes.
68     if ($this->configuration['custom_extra_classes']) {
69         $build['#attributes']['class'][] = $this->configuration['custom_extra_classes'];
70     }
71     return $build;
72 }
```

Drupal.OffCanvas

Introduced in Drupal 8.5, Drupal.OffCanvas library is a great way to show additional information on a page via off-canvas style sidebar. There three types of Drupal dialogs supported: Modal, Non modal and Off canvas. Modal and non-modal are similar popup style dialogs, The logic to create all three are similar.



<https://dri.es/how-to-use-drupal-8-off-canvas-dialog-in-your-modules>

<https://www.drupal.org/docs/drupal-apis/ajax-api/ajax-dialog-boxes>

Drupal.OffCanvas (a note about views before we get started)

There is an existing bug where a views page title will be returned as a render array then casted to plain text instead of being rendered to plain-text, so we must add a workaround to ensure the title is rendered before the response is retrieved.

```
8 use Drupal\my_front\Render\Element\MyViewElement;
9
10 /**
11  * Implements hook_element_info_alter().
12  */
13 function my_front_element_info_alter(&$types) {
14     if (isset($types['view'])) {
15         $types['view']['#pre_render'] = $types['view']['#pre_render'] ?? [];
16         // Fix title is an array in Off canvas dialog.
17         // Remove when fixed.
18         // @see https://www.drupal.org/project/drupal/issues/2663316
19         $types['view']['#pre_render'][] = MyViewElement::class . '::preRender';
20     }
21 }
22
```

<https://www.drupal.org/project/drupal/issues/2663316>

Drupal.OffCanvas (a note about views before we get started)

Contents: web/themes/my_front/src/Render/Element/MyViewElement.php

```
3 namespace Drupal\my_front\Render\Element;
4
5 use Drupal\Core\Render\BubbleableMetadata;
6 use Drupal\Core\Render\Element\RenderCallbackInterface;
7
8 /**
9  * MyViewElement helper class.
10  */
11 class MyViewElement implements RenderCallbackInterface {
12     // Code.
13 }
```


Drupal.OffCanvas (a note about views before we get started)

Contents: web/themes/my_front/src/Render/Element/MyViewElement.php (cont.)

```
20 public static function preRender($element) {
21     if (isset($element['#title']) && is_array($element['#title'])) {
22         $title_array = $element['#title'];
23         $element['#title'] = \Drupal::service('renderer')->renderPlain($title_array);
24         // Merge metadata such as #attached and #cache.
25         BubbleableMetadata::createFromRenderArray($element)
26             ->merge(BubbleableMetadata::createFromRenderArray($title_array))
27             ->applyTo($element);
28     }
29     return $element;
30 }
```

Drupal.OffCanvas & Layout Builder

Since View blocks can be used in layout builder, it's an easy win to show content via the off canvas dialog.

Example: views rewrite results. Assume 'tid' is a taxonomy id from another field. URL query parameter is there for demonstration purposes.

```
1  {{ attach_library('core/drupal.dialog.ajax') }}
2  {{ attach_library('core/drupal.dialog.off_canvas') }}
3  <a class="use-ajax my-extra-class" href="{ path(
4    'view.my_view.mypage',
5    {'arg_0': tid},
6    {
7      'query': {
8        'my_paramter': url('<current>')|render,
9      }
10   }
11 )
12 }}" data-dialog-type="dialog" data-dialog-renderer="off_canvas">{{ 'My link'|t }}</a>
```

<https://dri.es/how-to-use-drupal-8-off-canvas-dialog-in-your-modules>

<https://www.drupal.org/docs/drupal-apis/ajax-api/ajax-dialog-boxes>

Drupal.OffCanvas & Layout Builder

Example: Override the link and use off canvas in a link field twig template.

```
43 {% extends "field.html.twig" %}
44 {# Ensure no crash on layout builder screen. #}
45 ▼ {% if element['#entity_type'] and element['#object'] %}
46 ▼   {% for key, item in items %}
47 ▼     {% set item = item|merge({
48 ▼       'content': link(
49         item.content['#title'],
50         'base:' ~ path('view.my_view.my_page', {'arg_0': tid}),
51 ▼       create_attribute()
52         .addClass('use-ajax', 'my-extra-class')
53         .setAttribute('data-dialog-type', 'dialog')
54         .setAttribute('data-dialog-renderer', 'off_canvas')
55       )|merge({
56         '#attached': {'library': ['core/drupal.dialog.ajax', 'core/drupal.dialog.off_canvas']}
57       })
58     }) %}
59     {% set items = ( {(key): item} + items ) %}
60   {% endfor %}
61   {% set items = items|reverse %}
62 {% endif %}
```

Drupal.OffCanvas JS tricks

Add HTML classes to the off canvas dialog and other elements in reaction it's activation.
The variable "dialogOptions" will be shown on the next slide.

```
3 (function ($, Drupal) {  
4   Drupal.behaviors.MyThemeAddMyThingDialog = {  
5     attach: function (context, settings) {  
6       $(context).find('a.use-ajax.my-extra-class').once('my-extra-class-processed')  
7         .each(function (element_index, element) {  
8           $.each(Drupal.ajax.instances, function (index, instance) {  
9             if ((!instance.element) || !$(element).is(instance.element)) {  
10              return true;  
11            }  
12            let classSuffixes = ['--mything'];  
13            // See next slide.  
14            let dialogOptions = {};  
15            // @see Drupal.ajax.bindAjaxLinks()  
16            // This built from the element data.  
17            instance.elementSettings.dialog = $.extend(true, instance.elementSettings.dialog, dialogOptions);  
18            // @see Drupal.Ajax().  
19            // @see web/core/misc/ajax.es6.js  
20            instance.options.data.dialogOptions = $.extend(true, instance.options.data.dialogOptions, dialogOptions);  
21          });  
22        });  
23      }  
24    };  
25  })(jQuery, Drupal);
```

Drupal.OffCanvas JS tricks

The variable “dialogOptions”.

```
let dialogOptions = {
  'classes': {
    'ui-dialog': $.map(classSuffixes, (e) => { return 'ui-dialog'.concat(e); }).join(' '),
    'ui-dialog-off-canvas': $.map(classSuffixes, (e) => { return 'ui-dialog-off-canvas'.concat(e); }).join(' '),
    "ui-dialog-titlebar": $.map(classSuffixes, (e) => { return 'ui-dialog-titlebar'.concat(e); }).join(' '),
    "ui-dialog-title": $.map(classSuffixes, (e) => { return 'ui-dialog-title'.concat(e); }).join(' '),
    "ui-dialog-titlebar-close": $.map(classSuffixes, (e) => { return 'ui-dialog-titlebar-close'.concat(e); }).join(' '),
    "ui-dialog-content": $.map(classSuffixes, (e) => { return 'ui-dialog-content'.concat(e); }).join(' '),
    "ui-dialog-buttonpane": $.map(classSuffixes, (e) => { return 'ui-dialog-buttonpane'.concat(e); }).join(' '),
  },
  is_my_thing: true,
  width: 450,
};
```

Drupal.OffCanvas JS tricks

Add a HTML class to the body tag for purposes such as dimming the page.

```
3  (function ($, Drupal) {
4    // Add body class when modal is open.
5    $(window).on('dialog:beforecreate', function (e, dialog, $element, settings) {
6      if (settings.is_my_thing) {
7        $('body').addClass('modal-dialog-open--off-canvas--my-thing');
8      }
9    });
10   // Remove body class when modal is closed.
11   $(window).on('dialog:beforeclose', function (e, dialog, $element) {
12     let settings = $element.dialog('option');
13     if (settings.is_my_thing) {
14       $('body').removeClass('modal-dialog-open--off-canvas--my-thing');
15     }
16   });
17 })(jQuery, Drupal);
```

Drupal.OffCanvas JS tricks

Can also react and change another element according to the sidebar width.

```
3 ▼ (function ($, Drupal) {
4   // Can also change another element.
5 ▼   $(window).on('dialog:aftercreate', function (e, dialog, $element, settings) {
6 ▼     if (settings.is_my_thing) {
7       // @see Drupal.offCanvas.afterCreate().
8       const eventData = { settings, $element, offCanvasDialog: this };
9 ▼       $element.on('dialogContentResize.off-canvas', eventData, function (event) {
10         let settings = event.data.settings;
11         const $element = event.data.$element;
12         const $container = Drupal.offCanvas.getContainer($element);
13         const width = $container.outerWidth();
14         $('<div>.another-element</div>').css('right', width);
15       });
16     }
17   });
18 })(jQuery, Drupal);
```

DEMO!

- ▶ Australian State of the Environment Report 2021
 - ▶ <https://soe.dcceew.gov.au/overview/key-findings>
 - ▶ Click "cite this page" on the right.
- ▶ Simple Local Development Site and code (for any questions)

DrupalSouth 2022 sprint



Friday, 21 Oct 2022



See drupalsouth.org for
more information

Questions?

Leave feedback at drupalsouth.org

Demo code is here:

<https://github.com/silverham/DrupalSouth2022Demo>

EY | Building a better working world

EY exists to build a better working world, helping to create long-term value for clients, people and society and build trust in the capital markets.

Enabled by data and technology, diverse EY teams in over 150 countries provide trust through assurance and help clients grow, transform and operate.

Working across assurance, consulting, law, strategy, tax and transactions, EY teams ask better questions to find new answers for the complex issues facing our world today.

EY refers to the global organization, and may refer to one or more, of the member firms of Ernst & Young Global Limited, each of which is a separate legal entity. Ernst & Young Global Limited, a UK company limited by guarantee, does not provide services to clients. Information about how EY collects and uses personal data and a description of the rights individuals have under data protection legislation are available via ey.com/privacy. EY member firms do not practice law where prohibited by local laws. For more information about our organization, please visit ey.com.

In Consulting, we are building a better working world by transforming businesses through the power of people, technology and innovation.

It's our ambition to become the world's leading transformation consultants.

The diversity and skills of 70,000+ people will help clients realize transformation by putting humans at the center, delivering technology at speed and leveraging innovation at scale.

These core drivers of "Transformation Realized" will create long-term value for people, clients and society.

For more information about our Consulting organization, please visit ey.com/consulting.

© 2022 Ernst & Young, Australia.
All Rights Reserved.

Liability limited by a scheme approved under Professional Standards Legislation.

This communication provides general information which is current at the time of production. The information contained in this communication does not constitute advice and should not be relied on as such. Professional advice should be sought prior to any action being taken in reliance on any of the information. Ernst & Young disclaims all responsibility and liability (including, without limitation, for any direct or indirect or consequential costs, loss or damage or loss of profits) arising from anything done or omitted to be done by any party in reliance, whether wholly or partially, on any of the information. Any party that relies on the information does so at its own risk.

ey.com