
UI Patterns Documentation

Release 1.x

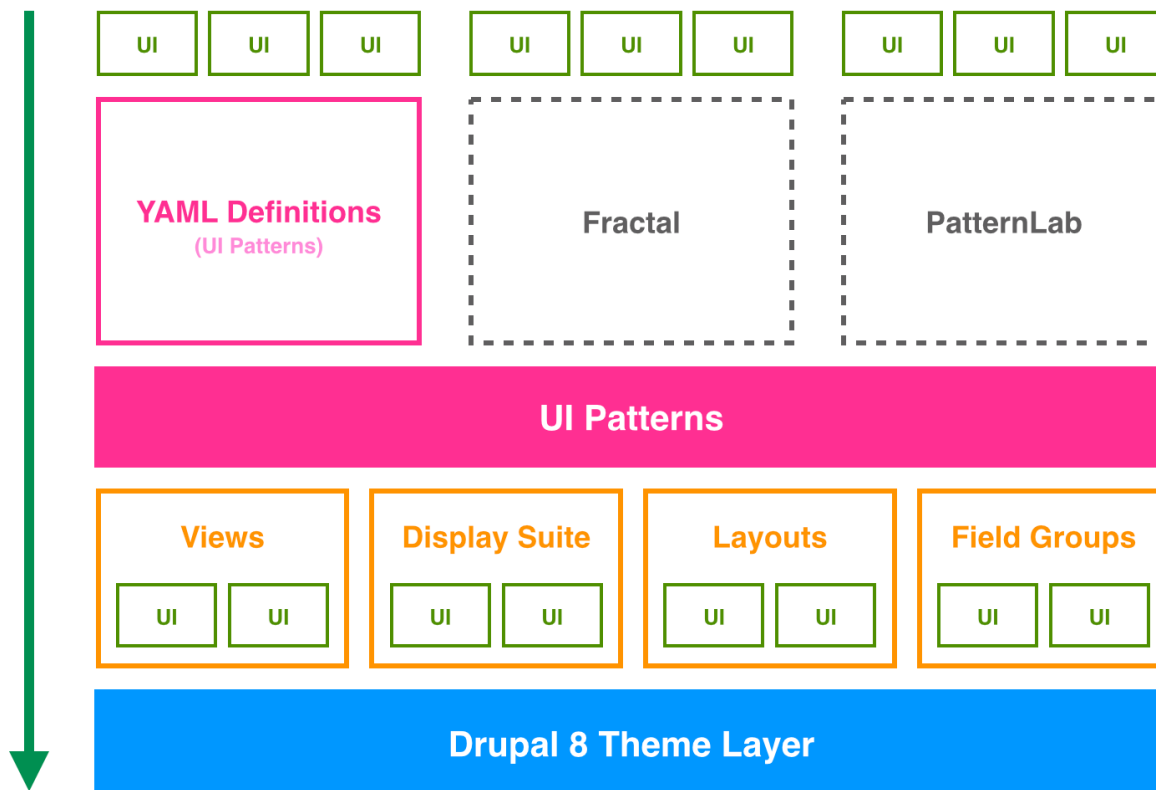
Nuvole Web

Aug 22, 2022

Table of Contents

1	Project overview	3
1.1	Try it out	3

The UI Patterns module allows developers to define self-contained UI patterns as Drupal plugins and use them seamlessly in their [panels](#), [field groups](#) or [Display Suite](#) view modes.



CHAPTER 1

Project overview

The UI Patterns project provides 6 modules:

- **UI Patterns:** the main module, it exposes the UI Patterns system APIs and it does not do much more than that.
- **UI Patterns Library:** allows to define patterns via YAML and generates a pattern library page available at `/patterns` to be used as documentation for content editors or as a showcase for business. Use this module if you don't plan to use more advanced component library systems such as PatternLab or Fractal. [Learn more](#)
- **UI Patterns Field Group:** allows to use patterns to format field groups provided by the [Field group](#) module.
- **UI Patterns Layouts:** allows to use patterns as layouts. This allows patterns to be used on [Display Suite](#) view modes or on [panels](#) out of the box.
- **UI Patterns Display Suite:** allows to use patterns to format [Display Suite](#) field templates.
- **UI Patterns Views:** allows to use patterns as Views row templates.

By the way plugin definitions are handled the UI Patterns module also integrates with tools like [PatternLab](#) or modules like [Component Libraries](#).

1.1 Try it out

Download and install the [Bootstrap Patterns](#) theme on a vanilla Drupal 8 installation to quickly try out the UI Patterns module.

1.1.1 Define your patterns

Patterns can be exposed by both modules and themes by enabling the `ui_patterns_library` module.

Defined patterns will be available at `/patterns`, only accessible by roles having the `access patterns page` permission. Below an example of a pattern library page styled using the [Bootstrap Patterns](#) theme:

Showcase

Demo site for the **UI Patterns** module. build passing

UI Patterns

- Carousel
- Jumbotron
- Modal
- Metadata
- Blockquote

Carousel

[View Carousel >](#)

A slideshow component for cycling through elements, like a carousel.

ID	Field	Description
slides	Slides	Each slide is a collection of title, subtitle and slide image.

Pattern definitions

To define your patterns simply create a YAML file named `MY_MODULE.ui_patterns.yml` or `MY_THEME.ui_patterns.yml` and list them using the following format:

```
blockquote:
  label: Blockquote
  description: Display a quote with attribution information.
  variants:
    default:
      label: Default
      description: An ordinary quote.
    highlighted:
      label: Highlighted
      description: A special quote.
  fields:
    quote:
      type: text
      label: Quote
      description: Quote text.
      preview: Life is like riding a bicycle. To keep your balance, you must keep_↵
↵moving.
```

(continues on next page)

(continued from previous page)

```

attribution:
  type: text
  label: Attribution
  description: Quote attribution.
  preview: Albert Einstein
libraries:
  - MY_MODULE/module_library_one
  - MY_MODULE/module_library_two
  - pattern_library_one:
    css:
      component:
        css/my_component.css: {}
        http://example.com/external.min.css: { type: external, minified: true }
  - pattern_library_two:
    js:
      js/library_two.js: {}
    dependencies:
      - core/jquery

```

Let's break this down:

id The root of a new pattern definition (blockquote in the example above). It must contain only lowercase characters, numbers, underscores and hyphens (i.e. it should validate against `[^a-z0-9_-]+`).

label Pattern label, used on pattern library page.

description Pattern description (optional), used on pattern library page.

fields Hash defining the pattern fields (optional). Each field can have the following properties:

type Field type, can be `text`, `numeric`, etc. at the moment only used for documentation purposes. Optional.

label Field label, used on pattern library page.

description Field description, used on pattern library page. Optional.

preview Preview content, used on pattern library page. It can be either a string or a Drupal render array, in which case we can use keys like `type: processed_text` or `theme: image`. Optional.

variants Hash defining the pattern variants (optional). Each variant can have the following properties:

label Variant label, used on pattern library page.

description Variant description, used on pattern library page. Optional.

libraries List of libraries to be loaded when rendering the pattern (optional). UI patterns are supposed to be self-contained so they should define along all needed libraries.

Once the pattern is defined the module will expose them as standard Drupal theme definitions.

For example, given the `my_pattern` pattern ID then a theme function `pattern_my_pattern` is created and, consequently, the module will look for a template file called `pattern-my-pattern.html.twig`.

The Twig template can be placed either under `MY_MODULE/templates`, if the pattern is exposed by a module, or under `MY_THEME/templates`, if it is exposed by a theme. As expected themes will override templates exposed by modules.

For example, consider the following Twig template file `pattern-blockquote.html.twig` for the `blockquote` pattern defined above:

```
<blockquote>
  <p>{{ quote }}</p>
  <footer>{{ attribution }}</footer>
</blockquote>
```

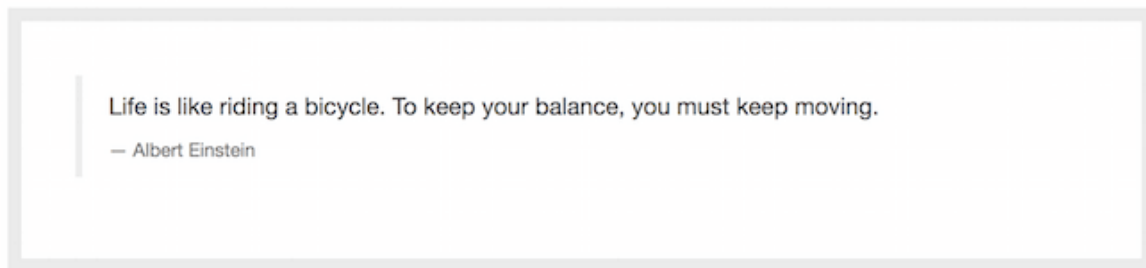
The pattern will be rendered as shown below (styled using the [Bootstrap](#) theme):

Blockquote

[View Blockquote >](#)

Display a quote with attribution information.

Field	Label	Type	Description
quote	Quote	text	Quote text.
attribution	Attribution	text	Quote attribution.



Remember: we can always visit the `/pattern` page in order to have access to a full preview of all our patterns.

Organize your patterns in sub-folders

Patterns can be defined using a single `NAME.ui_patterns.yml` file. However, in case of sites with a large number of patterns, this might quickly becomes difficult to manage.

If that's the case pattern definitions can also be organised in sub-folders, as shown below:

```
.
├── templates
│   └── patterns
│       ├── button
│       │   ├── button.ui_patterns.yml
│       │   └── pattern-button.html.twig
│       ├── media
│       │   ├── media.ui_patterns.yml
│       │   └── pattern-media.html.twig
│       └── ...
│           └── pattern-jumbotron.html.twig
├── ui_patterns_test_theme.info.yml
└── ui_patterns_test_theme.ui_patterns.yml
```

Note: the example above is taken by the actual test target site that is used to test the module itself: have a look at `./tests/README.md` and at `./tests/target/custom` for working examples on how to use the UI Patterns

module.

Expose pattern assets as libraries

In case you wish to bundle your assets within the pattern directory you can define libraries with the alternative syntax below:

```
blockquote:
  label: Blockquote
  ...
  libraries:
    ...
    - pattern_library_one:
      css:
        component:
          css/my_component.css: {}
          http://example.com/external.min.css: { type: external, minified: true }
    - pattern_library_two:
      js:
        js/library_two.js: {}
```

Libraries defined as above will be automatically loaded when the pattern is rendered. They are also exposed as ordinary Drupal libraries as follows: `ui_patterns/PATTERN_ID.LIBRARY_NAME`

For example, the two local libraries above can be attached to your render arrays in the following way:

```
<?php
$build['#attached']['library'][] = 'ui_patterns/blockquote.pattern_library_one';
$build['#attached']['library'][] = 'ui_patterns/blockquote.pattern_library_two';
```

Override patterns behavior

The default behavior can be changed by using the following properties in you pattern definitions:

theme hook If specified it overrides the default `pattern_[id]` theme hook with the provided value; the template file will change accordingly.

template If specified it overrides only the template file keeping the default `pattern_[id]` theme hook.

use If specified it will use a stand-alone Twig file as template. The value supports [Twig namespaces](#), so the following notations are valid examples:

```
use: "@my_module/templates/my-template.html.twig"
```

```
use: "@molecules/media/media-block.twig"
```

The possibility of using stand-alone Twig templates allows for a swift integration with tools like [PatternLab](#) or modules like [Component Libraries](#).

Attention: always remember to double-quote `use:` values or some YAML parsers (including PatternLab's) will complain.

1.1.2 Use patterns with Field Groups

Patterns can be used to style entities' [field groups](#) thanks to the `ui_patterns_field_group` module.

For example, say we want to show some metadata associated with an article, such as author, post date and tags.

After enabling the module we create a new field group of type **Pattern** and drag all fields you want to use in that group, as shown below:

LABEL	REGION	NAME	FIELD	WIDGET
Content				
✚ Metadata	Content		Pattern	Pattern: Metadata
✚ Author	Content	- Hidden -	Author	Field template: default
✚ Post date	Content	- Hidden -	long	Field template: default
✚ Tags	Content	- Hidden -	Label	No link Field template: default

Once all fields are in place we access the field group settings and choose the **Metadata** pattern. At this point we map the fields to the pattern destination fields and save our settings:

Field group format: pattern_formatter

Field group label

Metadata

Pattern *

Metadata

[Show row weights](#)

SOURCE	PLUGIN	DESTINATION
✚ Tags	Fields	Categories
✚ Author	Display Suite	Author
✚ Post date	Display Suite	Publication date

Update Cancel

Articles will now always use the **Metadata** pattern to style that field group, as shown below:

Pellentesque habitant morbi tristique senectus

Author:	admin
Publication date:	Wednesday, January 11, 2017 - 19:12
Categories:	Tag 1 Tag 2

Ut a nisl id ante tempus hendrerit. Nullam cursus lacinia erat. Sed augue ipsum, egestas nec, vestibulum et, malesuada adipiscing, dui. Proin sapien ipsum, porta a, auctor quis, euismod ut, mi. Praesent venenatis metus at tortor pulvinar varius.

Nam pretium turpis et arcu. Nullam vel sem. Morbi mollis tellus ac sapien. Donec mi odio, faucibus at, scelerisque quis, convallis in, nisi. Ut tincidunt tincidunt erat.

Suspendisse eu ligula. Donec vitae sapien ut libero venenatis faucibus. Donec sodales sagittis magna. Proin magna. Proin sapien ipsum, porta a, auctor quis, euismod ut, mi.

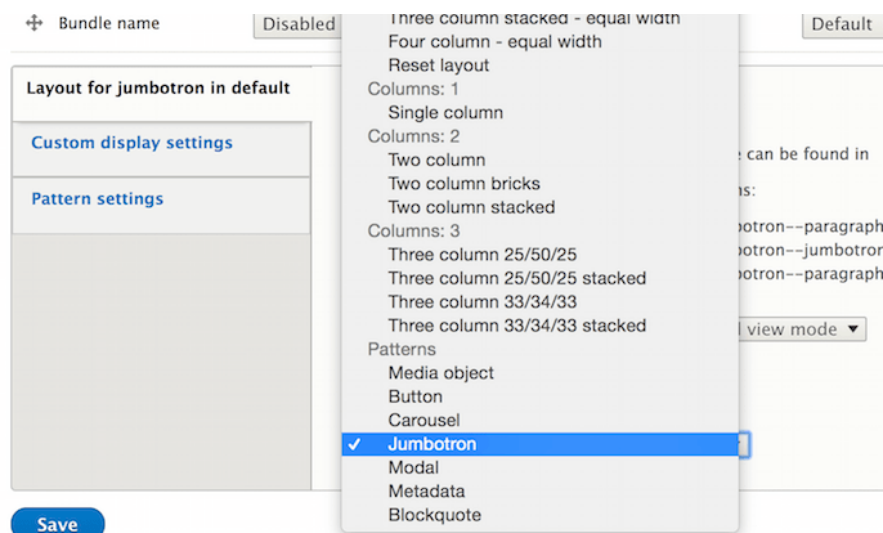
1.1.3 Use patterns as layouts

Patterns can be used as layouts thanks to the `ui_patterns_layouts` module.

Once exposed as layouts patterns can be used to arrange fields on entities like nodes, [paragraphs](#), etc. or to place blocks on a page using [Panels](#).

In the example below we will style a **Jumbotron** paragraph using the Jumbotron paragraph.

Once on the paragraph **Manage display** page we choose the **Jumbotron** pattern as layout:



After doing that the pattern fields will be exposed as layout regions, so given the following definition:

```
jumbotron:
  label: Jumbotron
  description: A lightweight, flexible component that can optionally extend the
  ↳entire viewport to showcase key content on your site.
  fields:
    title:
      type: text
      label: Title
      description: Jumbotron title.
      preview: Hello, world!
    subtitle:
      type: text
      label: Description
      description: Jumbotron description.
      preview: This is a simple hero unit, a simple jumbotron-style component for
  ↳calling extra attention to featured content or information.
```

We will get the following layout regions:

[+ Add group](#)

LABEL	REGION	NAME			
Title					
✚ Title	Title ▼	Above ▼			
Description					
✚ Subtitle	Description ▼	Above ▼			

We can now arrange the paragraph fields on the layout and save our settings.

The paragraph below:

[Show row weights](#)

PARAGRAPHS

✚ Type: Jumbotron
 Title

This is my Jumbotron title!

 Subtitle

It is a long established fact that a reader will be distracted by the readable content of a page when looking at its layout.

[Remove](#)

[Add Jumbotron](#)

will be now styled using the **Jumbotron** pattern as follows:

This is my Jumbotron title!

It is a long established fact that a reader will be distracted by the readable content of a page when looking at its layout.

1.1.4 Use patterns with Field templates

Patterns can be used as Display Suite field templates by enabling the `ui_patterns_ds` module. This opens the following interesting possibilities:

- Link fields can be styled as buttons by mapping their URL and link titles to specific pattern destinations.
- Image fields can be styled as an “image with caption” by mapping a formatted image and title to specific pattern destinations.

Let’s see how to implement the first example having the following pattern definition:

```
button:
  label: Button
  description: A simple button.
  fields:
    title:
      type: text
      label: Label
      description: The button label
      preview: Submit
    url:
      type: text
      label: URL
      description: The button URL
      preview: http://example.com
```

On the entity display setting page we access the link field setting by clicking on the gear icon:



Then, after selecting the **Pattern** field template and the **Button** pattern, we map the link field columns to the pattern’s fields defined above:

Choose a Field Template

Pattern ▼

Pattern *
Button ▼

[Show row weights](#)

SOURCE	PLUGIN	DESTINATION
⛶ Links	Field template	- Hidden - ▼
⛶ Links: uri	Field template	URL ▼
⛶ Links: title	Field template	Label ▼
⛶ Links: options	Field template	- Hidden - ▼

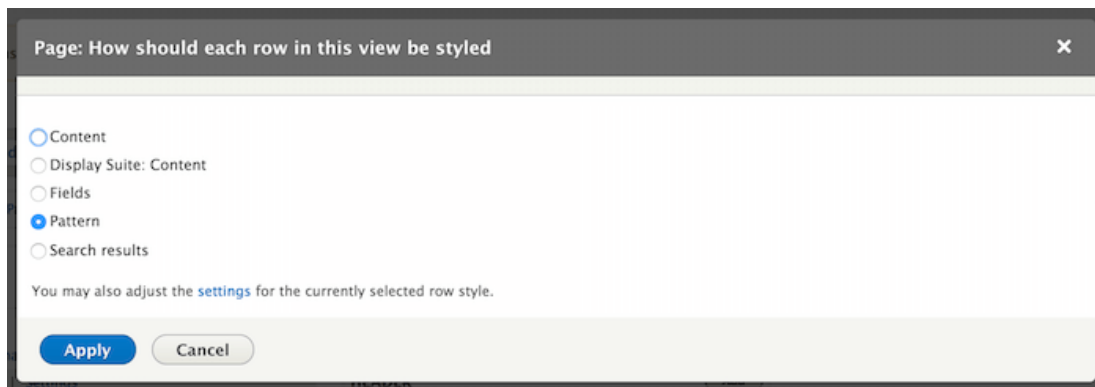
Update Cancel

Our multi-valued link field will then be formatted as follow:



1.1.5 Use patterns with Views

Patterns can be used as Views row templates thanks to the `ui_patterns_views` module, which exposes a **Patterns** row style plugin.



After choosing the **Pattern** row style plugin we can map the current Views display fields to the chosen pattern destinations, as shown below:

Page: Row style options

☐ Provide default field wrapper elements
If not checked, fields that are not configured to customize their HTML elements will get no wrappers at all for their field, label and field + label wrappers. You can use this to quickly reduce the amount of markup the view provides by default, at the cost of making it more difficult to apply CSS.

☐ Hide empty fields
Do not display fields, labels or markup for fields that are empty.

Pattern *
Media object

Show row weights

SOURCE	PLUGIN	DESTINATION
Content: Title	Views row	Title
Content: Body	Views row	Body

Apply

Cancel

Views rows will now be styled using the selected pattern.

1.1.6 Developer documentation

Render patterns programmatically

Patterns can be rendered programmatically by using the following syntax:

```
<?php
$elements['quote'] = [
  '#type' => 'pattern',
  '#id' => 'blockquote',
  '#fields' => [
    'quote' => 'You must do the things you think you cannot do.',
    'attribution' => 'Eleanor Roosevelt'
  ]
];

\Drupal::service('renderer')->render($elements);
```

The code above will produce the following result:

You must do the things you think you cannot do.

— Eleanor Roosevelt

To render a specific pattern variant use:

```
<?php
$elements['quote'] = [
  '#type' => 'pattern',
  '#id' => 'blockquote',
  '#variant' => 'highlighted',
```

(continues on next page)

(continued from previous page)

```
'#fields' => [
  'quote' => 'You must do the things you think you cannot do.',
  'attribution' => 'Eleanor Roosevelt'
]
];
```

It is also possible to just render a pattern preview as displayed on the patterns overview page in the following way (since fields are already bundled within the pattern definition we don't need to re-declare them here):

```
<?php
$elements['quote'] = [
  '#type' => 'pattern_preview',
  '#id' => 'blockquote',
];

\Drupal::service('renderer')->render($elements);
```

Rendering the code above will produce the following output:

```
Life is like riding a bicycle. To keep your balance, you must keep moving.
— Albert Einstein
```

To render a specific pattern preview variant use:

```
<?php
$elements['quote'] = [
  '#type' => 'pattern_preview',
  '#id' => 'blockquote',
  '#variant' => 'highlighted',
];
```

Render patterns using Twig functions

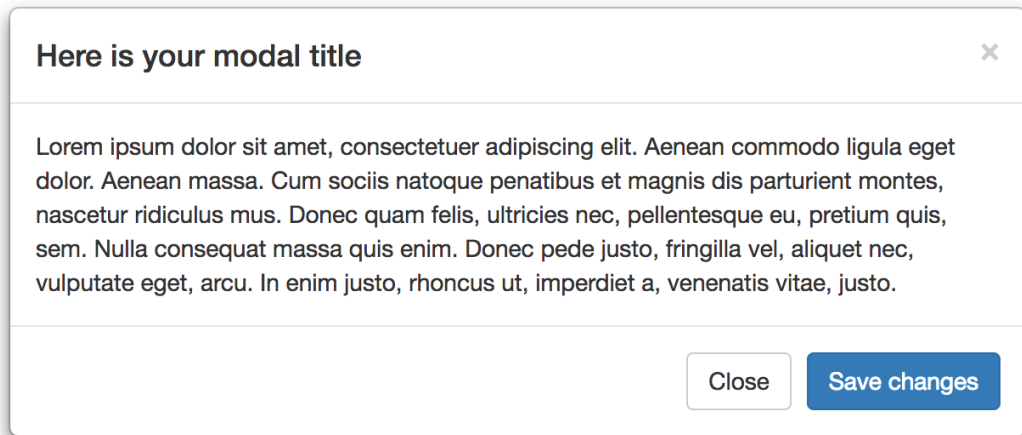
The UI Patterns module also exposes two Twig functions to easily render patterns into your Twig templates.

The following two calls:

```
{{ pattern('button', {title: 'Link title', url: 'http://example.com'}) }}
{{ pattern_preview('modal') }}
```

Will print:

Link title



To render pattern variants use:

```
{{ pattern('button', {title: 'Link title', url: 'http://example.com'}, 'highlighted') }}
{{ pattern_preview('modal', 'highlighted') }}
```

Since patterns are rendered using the render element described above all libraries and preprocess hooks will be ran when using Twig functions.

Working with pattern suggestions

Modules that want to add theme hook suggestions to patterns can do that by implementing the following hook:

```
<?php
/**
 * Provide hook theme suggestions for patterns.
 *
 * @see ui_patterns_theme_suggestions_alter()
 */
function hook_ui_patterns_suggestions_alter(array &$suggestions, array $variables, PatternContext $context) {
    if ($context->isOfType('views_row')) {
        $hook = $variables['theme_hook_original'];
        $view_name = $context->getProperty('view_name');
        $display = $context->getProperty('display');

        $suggestions[] = $hook . '__views_row__' . $view_name;
        $suggestions[] = $hook . '__views_row__' . $view_name . '__' . $display;
    }
}
```

The hook above is a `hook_theme_suggestions_alter()` specifically designed for patterns. The hook is invoked with a `PatternContext` object that describes information on where the current pattern is being used.

Pattern suggestions can, for example, allow developers to use alternative pattern templates in specific contexts or to “massage” data before it sent to the pattern by implementing fine-grained preprocess hooks.

The following suggestions are automatically exposed by the project's sub-modules:

```
<?php

// Suggestions for patterns used as layouts.
// @see ui_patterns_layouts_ui_patterns_suggestions_alter()
$suggestions[] = $hook . '__layout';
$suggestions[] = $hook . '__layout__' . $entity_type;
$suggestions[] = $hook . '__layout__' . $entity_type . '__' . $bundle;
$suggestions[] = $hook . '__layout__' . $entity_type . '__' . $view_mode;
$suggestions[] = $hook . '__layout__' . $entity_type . '__' . $bundle . '__' . $view_
    mode;
$suggestions[] = $hook . '__layout__' . $entity_type . '__' . $entity_id;

// Suggestions for patterns used as Display Suite field templates.
// @see ui_patterns_ds_ui_patterns_suggestions_alter()
$suggestions[] = $hook . '__ds_field_template';
$suggestions[] = $hook . '__ds_field_template__' . $field_name;
$suggestions[] = $hook . '__ds_field_template__' . $field_name . '__' . $entity_type;
$suggestions[] = $hook . '__ds_field_template__' . $field_name . '__' . $entity_type .
    __' . $bundle;
$suggestions[] = $hook . '__ds_field_template__' . $field_name . '__' . $entity_type .
    __' . $view_mode;
$suggestions[] = $hook . '__ds_field_template__' . $field_name . '__' . $entity_type .
    __' . $bundle . '__' . $view_mode;

// Suggestions for patterns used as field groups templates.
// @see ui_patterns_field_group_ui_patterns_suggestions_alter()
$suggestions[] = $hook . '__field_group';
$suggestions[] = $hook . '__field_group__' . $group_name;
$suggestions[] = $hook . '__field_group__' . $group_name . '__' . $entity_type;
$suggestions[] = $hook . '__field_group__' . $group_name . '__' . $entity_type . '__' .
    $bundle;
$suggestions[] = $hook . '__field_group__' . $group_name . '__' . $entity_type . '__' .
    $view_mode;
$suggestions[] = $hook . '__field_group__' . $group_name . '__' . $entity_type . '__' .
    $bundle . '__' . $view_mode;

// Suggestions for patterns used as Views row templates.
// @see ui_patterns_views_ui_patterns_suggestions_alter()
$suggestions[] = $hook . '__views_row';
$suggestions[] = $hook . '__views_row__' . $view_name;
$suggestions[] = $hook . '__views_row__' . $view_name . '__' . $display;
```

When rendering a variant the following suggestions will be available too:

```
<?php

// Suggestions for patterns used as layouts.
// @see ui_patterns_layouts_ui_patterns_suggestions_alter()
$suggestions[] = $hook . '__variant__' . $variant . '__layout';
$suggestions[] = $hook . '__variant__' . $variant . '__layout__' . $entity_type;
$suggestions[] = $hook . '__variant__' . $variant . '__layout__' . $entity_type . '__' .
    $bundle;
$suggestions[] = $hook . '__variant__' . $variant . '__layout__' . $entity_type . '__' .
    $view_mode;
$suggestions[] = $hook . '__variant__' . $variant . '__layout__' . $entity_type . '__' .
    $bundle . '__' . $view_mode;
```

(continues on next page)

(continued from previous page)

```

$suggestions[] = $hook . '__variant_' . $variant . '__layout__' . $entity_type . '__' .
↳. $entity_id;

// Suggestions for patterns used as Display Suite field templates.
// @see ui_patterns_ds_ui_patterns_suggestions_alter()
$suggestions[] = $hook . '__variant_' . $variant . '__ds_field_template';
$suggestions[] = $hook . '__variant_' . $variant . '__ds_field_template__' . $field_
↳name;
$suggestions[] = $hook . '__variant_' . $variant . '__ds_field_template__' . $field_
↳name . '__' . $entity_type;
$suggestions[] = $hook . '__variant_' . $variant . '__ds_field_template__' . $field_
↳name . '__' . $entity_type . '__' . $bundle;
$suggestions[] = $hook . '__variant_' . $variant . '__ds_field_template__' . $field_
↳name . '__' . $entity_type . '__' . $view_mode;
$suggestions[] = $hook . '__variant_' . $variant . '__ds_field_template__' . $field_
↳name . '__' . $entity_type . '__' . $bundle . '__' . $view_mode;

// Suggestions for patterns used as field groups templates.
// @see ui_patterns_field_group_ui_patterns_suggestions_alter()
$suggestions[] = $hook . '__variant_' . $variant . '__field_group';
$suggestions[] = $hook . '__variant_' . $variant . '__field_group__' . $group_name;
$suggestions[] = $hook . '__variant_' . $variant . '__field_group__' . $group_name .
↳'__' . $entity_type;
$suggestions[] = $hook . '__variant_' . $variant . '__field_group__' . $group_name .
↳'__' . $entity_type . '__' . $bundle;
$suggestions[] = $hook . '__variant_' . $variant . '__field_group__' . $group_name .
↳'__' . $entity_type . '__' . $view_mode;
$suggestions[] = $hook . '__variant_' . $variant . '__field_group__' . $group_name .
↳'__' . $entity_type . '__' . $bundle . '__' . $view_mode;

// Suggestions for patterns used as Views row templates.
// @see ui_patterns_views_ui_patterns_suggestions_alter()
$suggestions[] = $hook . '__variant_' . $variant . '__views_row';
$suggestions[] = $hook . '__variant_' . $variant . '__views_row__' . $view_name;
$suggestions[] = $hook . '__variant_' . $variant . '__views_row__' . $view_name . '__'
↳. $display;

```

Expose source field plugins

When using a pattern on a view or an entity display form we are provided with a set of possible patterns source fields that we can map to our pattern destination fields. Available source fields depends on the context in which a pattern is being configured.

Pattern source fields are provided by plugins of type `@UiPatternsSource`.

For example, when a pattern is used as a Views row template then the `UiPatternsSourceManager` collects all plugins annotated with `@UiPatternsSource` and tagged by `views_row`. A context array describing the current view is then passed to each of the `@UiPatternsSource` plugins.

In the example below we can see the actual implementation of such a system:

```

<?php

namespace Drupal\ui_patterns_views\Plugin\UiPatterns\Source;

use Drupal\ui_patterns\Plugin\PatternSourceBase;

```

(continues on next page)

(continued from previous page)

```

/**
 * Defines Views row pattern source plugin.
 *
 * @UiPatternsSource(
 *   id = "views_row",
 *   label = @Translation("Views row"),
 *   provider = "views",
 *   tags = {
 *     "views_row"
 *   }
 * )
 */
class ViewsRowSource extends PatternSourceBase {

  /**
   * {@inheritdoc}
   */
  public function getSourceFields() {
    $sources = [];
    /** @var \Drupal\views\ViewExecutable $view */
    $view = $this->getContextProperty('view');
    foreach ($view->display_handler->getFieldLabels() as $name => $label) {
      $sources[] = $this->getSourceField($name, $label);
    }
    return $sources;
  }
}

```

At the moment the available source plugin tags are the following:

- `entity_display`: provided by the `ui_patterns` module and triggered on an entity display configuration page.
- `ds_field_template`: provided by the `ui_patterns_ds` module and triggered when setting up a field template on an entity display configuration page.
- `views_row`: provided by the `ui_patterns_views` module and triggered on a Views row setting pane.
- `test`: provided by the `ui_patterns_test` module and used in tests.

Alter pattern configuration forms

You can alter UI Patterns configuration forms by implementing `hook_ui_patterns_display_settings_form_alter()`.

For example, the following implementation adds a CSS class input field to the pattern configuration:

```

<?php

/**
 * Implements hook_ui_patterns_display_settings_form_alter().
 *
 * Add a css class name configuration option.
 */
function my_module_ui_patterns_display_settings_form_alter(array &$form, array
  ↪ $configuration) {

```

(continues on next page)

(continued from previous page)

```

$setting_value = isset($configuration['class_name']) ? $configuration[$key] : '';
$form['class_name'] = [
  '#type' => 'input',
  '#title' => t('Class name'),
];
}

```

This hook alter forms that are built using the `PatternDisplayFormTrait` trait, meaning:

- Display Suite field templates
- Field groups
- Views

If you want to alter an entity layout form that uses UI Patters for its layout use `hook_ui_patterns_layouts_display_settings_form_alter()` instead, for example:

```

<?php

/**
 * Implements hook_ui_patterns_layouts_display_settings_form_alter().
 *
 * Add a css class name configuration option.
 */
function hook_ui_patterns_layouts_display_settings_form_alter(array &$form,
↳ PatternDefinition $definition, array $configuration) {
  $class_name = isset($configuration['class_name']) ? $configuration['class_name'] :
↳ '';
  $form['class_name'] = [
    '#type' => 'input',
    '#title' => t('Class name'),
  ];
}

```

1.1.7 Working with tests

UI Patterns is tested using `PHPUnit` in `Drupal 8`.

To build the test site just run:

```
$ composer install
```

After that you'll find a fully functional `Drupal 8` site under `./build`, thanks to the integration with the [OpenEuropa Task Runner](#).

To run tests use:

```
$ ./vendor/bin/phpunit
```

1.1.8 Working with coding standards

UI Patterns coding standards checks are ran using `GrumPHP`.

```
$ ./vendor/bin/grumphp run
```

1.1.9 Docker Compose

UI Patterns ships with a `docker-compose.yml` file which can be used to streamline local development and tests execution.

Setup Docker Compose by copying `docker-compose.yml` to `docker-compose.override.yml` and replace `${TRAVIS_PHP_VERSION}` with the desired PHP version (either “5.6” or “7.1”).

After that run:

```
$ docker-compose up -d
$ docker-compose exec -u www-data php composer install
$ docker-compose exec -u www-data php ./vendor/bin/run drupal:site-setup
$ docker-compose exec -u www-data php ./vendor/bin/run drupal:site-install
$ docker-compose exec -u www-data php chown -R www-data:www-data build
```

You’ll then have a fully functional test site at <http://localhost:8080>.

To run all tests use:

```
$ docker-compose exec -u www-data php ./vendor/bin/grumphp run
$ docker-compose exec -u www-data php ./vendor/bin/phpunit
```