



APRIL 15<sup>TH</sup> - 18<sup>TH</sup> 2025

**SDC & Form API, better late than  
never**

# Florent Torregrosa

[Grimreaper](#)

Drupaler since 2011 (D6), professional since 2013. Tech expert at [Smile](#)

Community contributions:

- UI Suite ecosystem since 2021
- Active maintainer of 30+ projects
- Contrib and core patches
- French translation moderator
- Events co-organizer
- Former French Drupal association board member





# Current situation

# Form API , since [Fall 2005](#) (Drupal 4)

Nested arrays, describing a form structure.

Declarative. Easy to type. Easy nesting.

Easy alteration.

Return value of:

- `FormInterface::buildForm()`
- `PluginFormInterface::buildConfigurationForm()`
- `WidgetInterface::settingsform()`
- `WidgetInterface::formElement()`
- ...

```
return [
  'phone_number' => [
    '#type' => 'tel',
    '#title' => t('Your phone
number'),
  ],
  'actions' => [
    '#type' => 'actions',
    'submit' => [
      '#type' => 'submit',
      '#value' => t('Save'),
      '#button_type' => 'primary',
    ]
  ]
];
```

# Render API , since Summer 2006 (Drupal 5)

Inspired by the Form API.

Return value of:

- `LayoutInterface::build()`
- `FormatterInterface::view()`
- `FormatterInterface::viewElements()`
- `BlockPluginInterface::build()`
- ...

You can put (most) render elements in a form.

Building \$node->body with arrays like FAPI for viewing



**eaton** Credits commented 26 July 2006 at 07:04

#1

RobRoy, I've been giving some thought to this and if it's done right, I think it can be a huge improvement. I'll brain-dump my thinking on the subject so far and perhaps we'll come up with something that works.

1. The solution should allow module-provided data to be kept separate from rendered HTML
2. The solution should preserve the original body and teaser values from the database (if any) for use by themers
3. The solution should allow individual pieces of node data or the entire assembled node to be filtered and/or processed for viewing
4. The fully rendered teaser and body should still be stored in \$node->teaser and \$node->body once everything has been processed.
5. The solution should mirror existing Drupal systems (notably FormAPI) where possible. A new 'style' of data structures is bad.

Closed (fixed)

Project:	Drupal core
Version:	5.x-dev
Component:	node system
Priority:	Normal
Category:	Task
Assigned:	eaton
Reporter:	RobRoy
Created:	18 Jul 2006 at 20:41 CEST
Updated:	8 Dec 2006 at 22:46 CET

```
return [
  "#theme" => "item_list",
  "#items" => [
    [ "value" =>
      [
        "#theme" => "image",
        "#uri" => "/path/to/image"
      ]
    ]
  ]
]
```



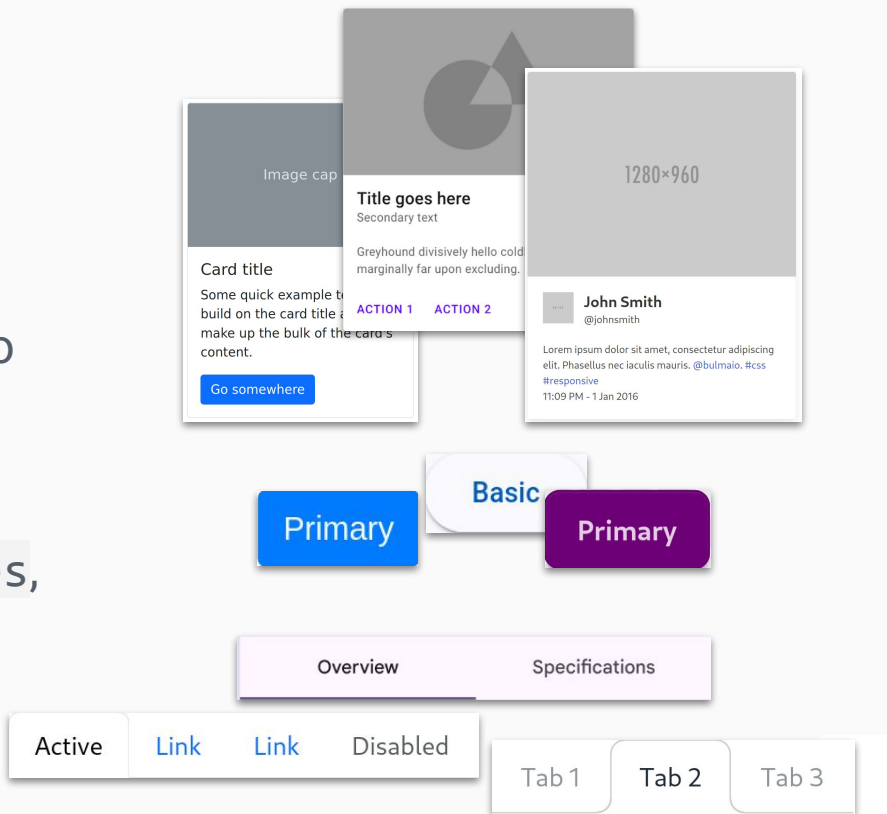
# SDC, since Spring 2023 (Drupal 10)

## The *front devs friendly* Render API.

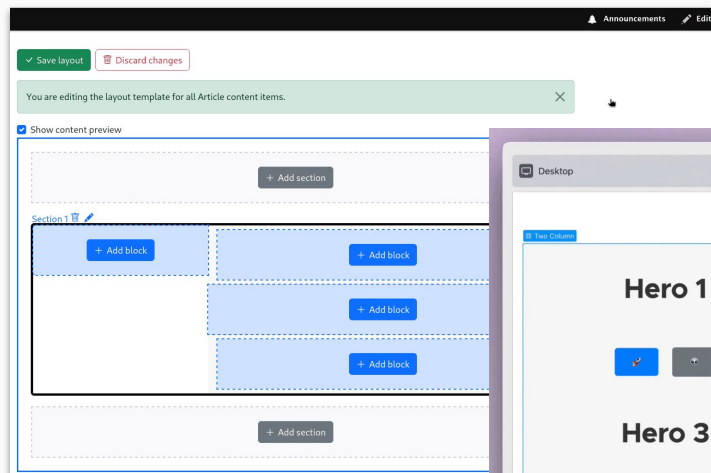
No PHP needed. UI focused & reusable components.

Good stuff inside: Twig templating, plugin discovery, render element, no preprocesses...

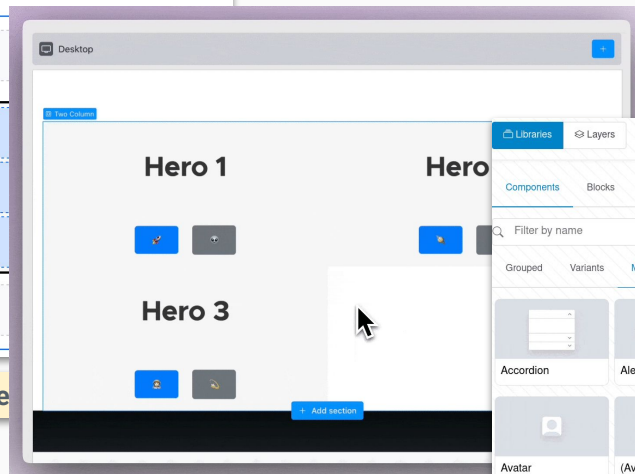
SDC can replace most of render elements (table, status\_messages, pager...) and hook themes (breadcrumb, progress\_bar, links...).



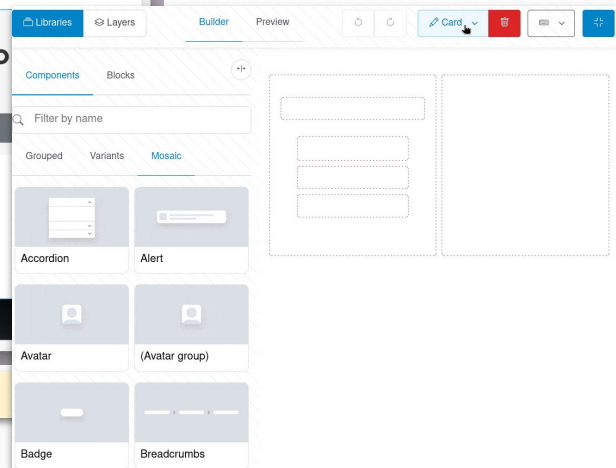
# SDC is changing the display building



Layout Builder + UI Patter



Experience Builder



UI Suite's Display Builder

A close-up photograph of several chilled beer bottles in a metal tray. The bottles are condensation-covered and filled with a golden beer. The caps are white with blue and red text. The background is dark and out of focus, with warm light reflecting off the bottles and the tray. The text "Problem 1: Using SDC in a Form" is overlaid in the center in a bold, white, sans-serif font.

# Problem 1: Using SDC in a Form



# Components are made of slots and props

**Slots:** “areas” for free renderables only, like other components.

**Props:** strictly typed data only, for some UI logic in the template.

```
name: Card
description: "Cards are used to group and..."
group: "Data display"
slots:
  image:
    title: Image
  title:
    title: Title
  text:
    title: Text
  actions:
    title: Actions
props:
  type: object
  properties:
    heading_level: {}
    image_bottom: {}
```



**Shoes!**

If a dog chews shoes whose shoes does he choose?

Buy Now

# Renderables are made of children & properties

**Properties:** Prefixed with “#”, data used in the render logic.

**Children:** Not prefixed, free renderables.

```
$build['my_details'] = [  
  '#type' => 'details',  
  '#title' => 'Example',  
  '#open' => TRUE,  
  'content' => [  
    '#markup' => 'Example content',  
  ],  
];
```

*Does it ring a bell?*



## We can insert a form inside a component slot

```
return [  
  '#type' => 'component',  
  '#component' => 'core_sdc_form:accordion',  
  '#slots' => [  
    'title' => 'Form in component accordion',  
    'content' => $this->formBuilder()->getForm(...),  
  ]  
];
```

Form in component accordion

Textfield in component

Submit

# ✗ But we can't insert a component in a form

```
$form['normal'] = [  
  '#type' => 'textfield',  
  '#title' => 'Normal form element',  
];  
$form['component'] = [  
  '#type' => 'component',  
  '#component' => 'core_sdc_form:accordion',  
  '#slots' => [  
    'title' => 'Form in component accordion',  
    'content' => [  
      'form_element_in_component' => [  
        '#type' => 'textfield',  
        '#title' => 'Form element in  
component',  
      ],  
    ],  
  ],  
];
```

FormBuilder::doBuildForm()  
expects children, not #slots  
property.

Normal form element

Test 1

Form in component accordion

Form element in component

Displayed as expected but data not  
submitted!

# Step 1: slots as render element children

```
$form['component'] = [  
  '#type' => 'component',  
  '#component' => 'foo:accordion',  
  '#slots' => [  
    'title' => 'Accordion title',  
    'content' => [  
      'form_element_in_component' => [  
        '#type' => 'textfield',  
        '#title' => 'Hi there',  
      ],  
    ],  
  ],  
];
```

```
$form['component'] = [  
  '#type' => 'component',  
  '#component' => 'foo:accordion',  
  'title' => 'Accordion title',  
  'content' => [  
    'form_element_in_component' => [  
      '#type' => 'textfield',  
      '#title' => 'Hi there',  
    ],  
  ],  
];
```



# Step 1: slots as render element children

In `ComponentElement::preRenderComponent()`, let's put the children (already processed by the form builder) into the slots.

```
// Handle children as slots.
$children = Element::children($element, TRUE);
foreach ($children as $key) {
    $element['#slots'][$key] = $element[$key];
}
```

- ✓ Form API is able to go through the render tree.
- ✓ Component element has its slots.

## Step 2: Let's leverage #name property

We still don't get the values when submitting the form.

Mismatch between slots hierarchy and names generated by FormBuilder from #parents

Let's use #name explicitly instead

```
'content' => [  
  '#type' => 'component',  
  '#component' => 'core_sdc_form:card_body',  
  'content' => [  
    [  
      '#type' => 'textfield',  
      '#title' => $this->t( string: 'Textfield in content'),  
      '#name' => 'content',  
    ],  
    [  
      '#type' => 'select',  
      '#title' => $this->t( string: 'Select in content'),  
      '#name' => 'content_select',  
      '#options' => [  
        'option_1' => $this->t( string: 'Option 1'),  
        'option_2' => $this->t( string: 'Option 2'),  
      ],  
      '#empty_option' => $this->t( string: 'Empty option'),  
    ],  
  ],  
],
```

## Step 2: Let's leverage #name property

In FormBuilder::handleInputElement()

```
$input = NestedArray::getValue($form_state->getUserInput(), $element['#parents'], $input_exists);  
if (!empty($element['#name'])) {  
    $input = NestedArray::getValue($form_state->getUserInput(), [$element['#name']], $input_exists);  
}  
else {  
    $input = NestedArray::getValue($form_state->getUserInput(), $element['#parents'], $input_exists);  
}
```

In FormStateValuesTrait::setValueForElement()

```
public function setValueForElement(array $element, $value) {  
    if (!empty($element['#name'])) {  
        return $this->setValue($element['#name'], $value);  
    }  
  
    return $this->setValue($element['#parents'], $value);  
}
```



A photograph of a warehouse filled with numerous metal beer kegs. The kegs are stacked in rows, with some resting on wooden pallets. The kegs are silver-colored and have "ANHEUSER-BUSCH, INC." embossed on them. The background shows industrial equipment and a forklift, suggesting a brewery or distribution center.

# **Problem 2: We can't define form element with SDC**

# Why is it important?

Form elements are (long) PHP classes instead of Twig + YAML.

It keeps the front dev away and are hard to manage.

Let's move those to the theming scope.

However, SDC components are stateless.

```
use Drupal\Core\Form\FormStateInterface;
use Drupal\Core\Render\Attribute\FormElement;
use Drupal\Core\Render\Element;

/**
 * Provides a form element for a submit button with an image.
 */
#FormElement('image.button')
class ImageButton extends Submit {

  /**
   * @inheritdoc
   */
  public function getInfo() {
    $info = parent::getInfo();
    unset($info['name']);

    return [
      'return_value' => TRUE,
      'has_garbage_value' => TRUE,
      'src' => NULL,
      'theme_wrappers' => ['input_image_button'],
    ] + $info;
  }

  /**
   * @inheritdoc
   */
  public static function valueCallback($element, $input, FormStateInterface $form_state) {
    if ($input !== FALSE) {
      if (empty($input)) {
        // If we're dealing with theme wrappers, we're lucky. It will
        // return a proper value.
        return $element['#return'];
      }
      else {
        // Unfortunately, in IE we never get a proper value for
        // form element. Instead, we get the coordinates of the
        // button. We'll find this element in the form state.
        // In the same spot for its name, with 'input' as prefix.
        $input = $form_state->getUserInput();
        foreach (explode('.', $element['#name']) as $part) {
          // Chop off the last part that may exist.
          if (strpos($part, $element['#name']) === 0) {
            $element_name = substr($element['#name'],
              strlen($part) + 1);
          }

          if (!isset($input[$element_name])) {
            if (isset($input[$element_name . '_button'])) {
              return $input[$element_name . '_button'];
            }
            return NULL;
          }
          $input = $input[$element_name];
        }
        return $element['#return_value'];
      }
    }
  }

  /**
   * @inheritdoc
   */
  public static function preRenderButton($element) {
    $element['#attributes']['type'] = 'image';
    Element::setAttribute($element, ['id', 'name', 'value']);

    $element['#attributes']['src'] = \Drupal::service('file_url_generator')->generateString($element['#src']);
    if (empty($element['#title'])) {
      $element['#attributes']['alt'] = $element['#title'];
      $element['#attributes']['title'] = $element['#title'];
    }

    $element['#attributes']['class'][] = 'image-button';
    if (empty($element['#button_type'])) {
      $element['#attributes']['class'][] = 'image-button-' . $element['#button_type'];
    }
    $element['#attributes']['class'][] = 'js-form-submit';
    $element['#attributes']['class'][] = 'form-submit';

    if (empty($element['#attributes']['disabled'])) {
      $element['#attributes']['class'][] = 'is-disabled';
    }

    return $element;
  }
}
```



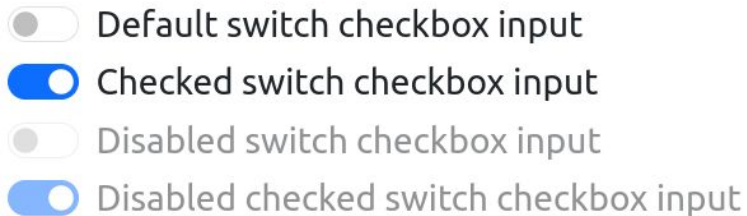
# The missing components

In a design system, (around) 20% of components are form elements.

Checkboxes, date pickers, sliders, input groups...

Today, they are skipped by the SDC authors.

Example: [Bootstrap Switches](#)



Example: [Material Search](#)



Hinted search text



# Step 1: Change in Render pipeline

A form element is a render element with 2 extra render properties:

- `#input` = True, added by ElementInfoManager if the element implements FormElementInterface
- `#name`, added by the form builder from the parent when missing

So:

- ComponentElement must implements FormElementInterface
- We can leverage the `#name` property expectations already discussed

# From an user point of view

A SDC form element is a SDC renderable with **#name**

Example:

```
[  
  "#type" => "component",  
  "#component" => "ui_suite_bootstrap:date",  
  "#name" => "foo",  
  "#default_value" => "2025-04-15",  
  "#slots" => [],  
  "#props" => [],  
]
```

## Step 2: Twig template additions

We inject a new variable with data needed for form processing:

- `form_state.name` (string)
- `form_state.value` (mixed)
- `form_state.required` (boolean)

More may be added later.

```
{% set id = id|default('mytextfield-' ~
random()) %}
{% set input_attributes =
create_attribute({
  type: 'text',
  name: form_state.name,
  id: id,
  value: form_state.value,
}) %}
<div>{{ attributes.addClass('mb-3') }}>
  <label class="form-label" for="{{ id }}">
    {{ label }}
  </label>
  <input{{ input_attributes }}>
</div>
```

# Input or wrapper template?

In general a Drupal form element uses 2 templates:

- The wrapper (`form-element.html.twig` for half of them) with label, errors & description
- The input (`input.html.twig`, `textarea.html.twig`, `select.html.twig`...)

After studying design systems (Bootstrap, Material, DSFR...), we decided to target wrappers.

But still challengeable.



A close-up photograph of two hands holding beer glasses. The hand on the left holds a tall, slender glass filled with a golden beer topped with a thick white head of foam. The hand on the right holds a shorter, wider glass filled with a dark, stout-like beer, also topped with foam. The person holding the glasses has blue nail polish. The background is a blurred bar interior with warm, glowing lights and the faint outlines of other patrons.

**What will be the benefits?**

# Benefit 1: Form API simplification

37 form elements in Core

4

**with only applicative logic** (no rendering)

Item, LanguageSelect, Submit & Value

 **Not suitable for SDC**

26

Do we split them?

- UI logic (#theme, #theme\_wrapper & #pre\_render) to SDC
- App logic (#process, ::valueCallback() & #element\_validate) kept in PHP classes

7

**without applicative logic**

Button, Date, Hidden, ImageButton, Search, Tel & VerticalTabs

 **Convertible to SDC**

# Benefit 2: Front dev empowerment

## The *front devs friendly* Form API.

No PHP involved. UI focused & reusable form components.

Part of the design system implementation.

Business agnostic. Sharable in a Drupal theme.

Testable out of context from a component library.

### Bootstrap radios ☆

ID: core\_sdc\_form:myradios

Status: experimental

Name	Label	Type	Description
id	ID	String	
options	Options	Unknown	⚠️ ("type":"object")

#### Default

- ☐ One
- ☐ Two
- ☐ Three

### Bootstrap select ☆

ID: core\_sdc\_form:myselect

Status: experimental

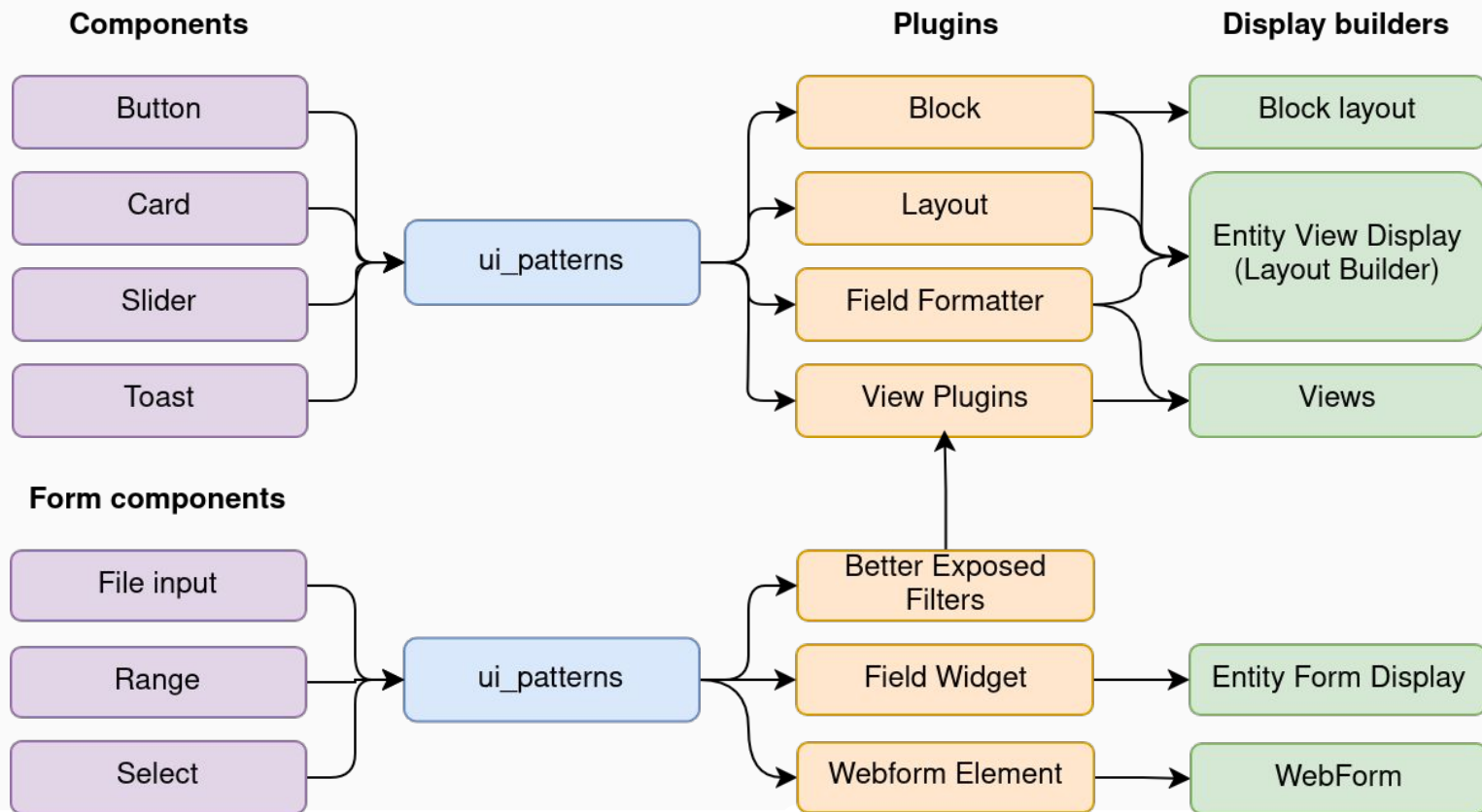
Name	Label	Type	Description
label	Label	Slot	
id	ID	String	
options	Options	Unknown	⚠️ ("type":"object")

#### Default

My Bootstrap select

One

## Benefit 3: Form building with SDC



# Work in progress

Core issues:

- #3494634: [Compatibility between SDC and the Form API](#)
- #3508641: [Define form elements from SDC](#)

Next steps:

- Test submit button with component
- Error message handling: transmit the state into the element & message if no #title
- Handle required on checkboxes and radios (backend side) (composite elements)
- Form Ajax API & Form states API
- ComponentElement: remove #slots usage completely
- Replacing existing form elements
- Handle #tree
- ...



A close-up photograph of a beer tap. The tap has a black handle and a silver-colored body. A circular label with the word 'Stampramen' in a script font is visible on the right side of the tap. The background is a soft, out-of-focus bokeh of warm, golden-yellow and blue lights, suggesting a bar or restaurant setting at night.

**Questions?**