Hooks & Events Overview

How Complex Systems Communicate

Jonathan Daggerhart

Architect

daggerhart

daggerhart

daggerhart

daggerhart.com

DRUPAL CAMP ASHEVILLE 2020

July 10-12, 2020
Asheville, NC
www.drupalasheville.com

Hooks & Events: What We'll Cover

- Event Systems in General
  - What problems does it solve?
  - Parts of an event system

- Exploration of popular Event Systems
  - Hooks in Drupal 7 & 8
  - Events in Drupal 8
  - WordPress Hooks
  - JavaScript Events

What is an Event System?

Patterns

- Mediator (centralized)
- Observer (distributed)

The implementation of a programming pattern that allows smaller components of a complicated framework to communicate with each other, modify shared data, and otherwise react to changes performed on the system.

HOOK42

What problem does an Event System solve?

The goal of an Event System is to:

- Prevent tight coupling between components
- Allow for communicating changes throughout components
- Allows modifications of the data of any component by almost any other component

It does this by acting as a mediator between disparate parts of the system

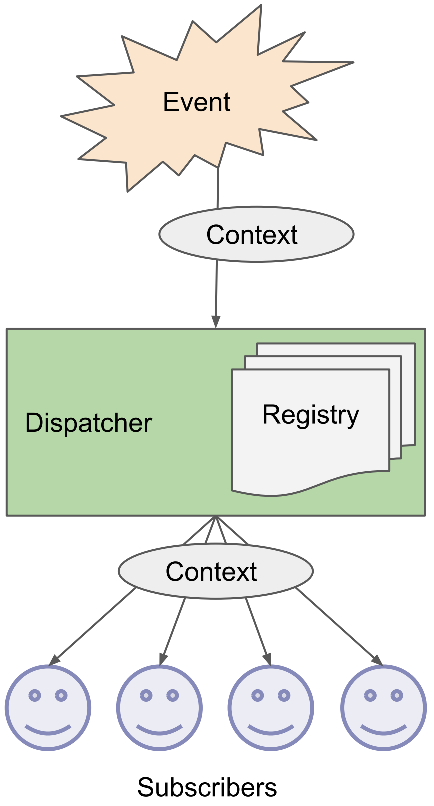
Sure… but what does *that* mean?

Imagine a system without hooks/events …

- Each module would have to explicitly update the components of each other module they want to interact with
- Component interactions would cause changes to other components that result in conflicts and errors
- Themes would have to override the entire output of all modules
- Dogs and cats living together… **Total chaos!**

Parts of an Event System

- **Event**
  A specific thing that happened
- **Context**
  Details about the event
- **Subscriber (aka, Listener)**
  Component that wants to know about an event occurrence
- **Registry**
  List of subscribers per event
- **Dispatcher**
  Delivers event context to subscribers



Newspapers: An Event System

(An Analogy)

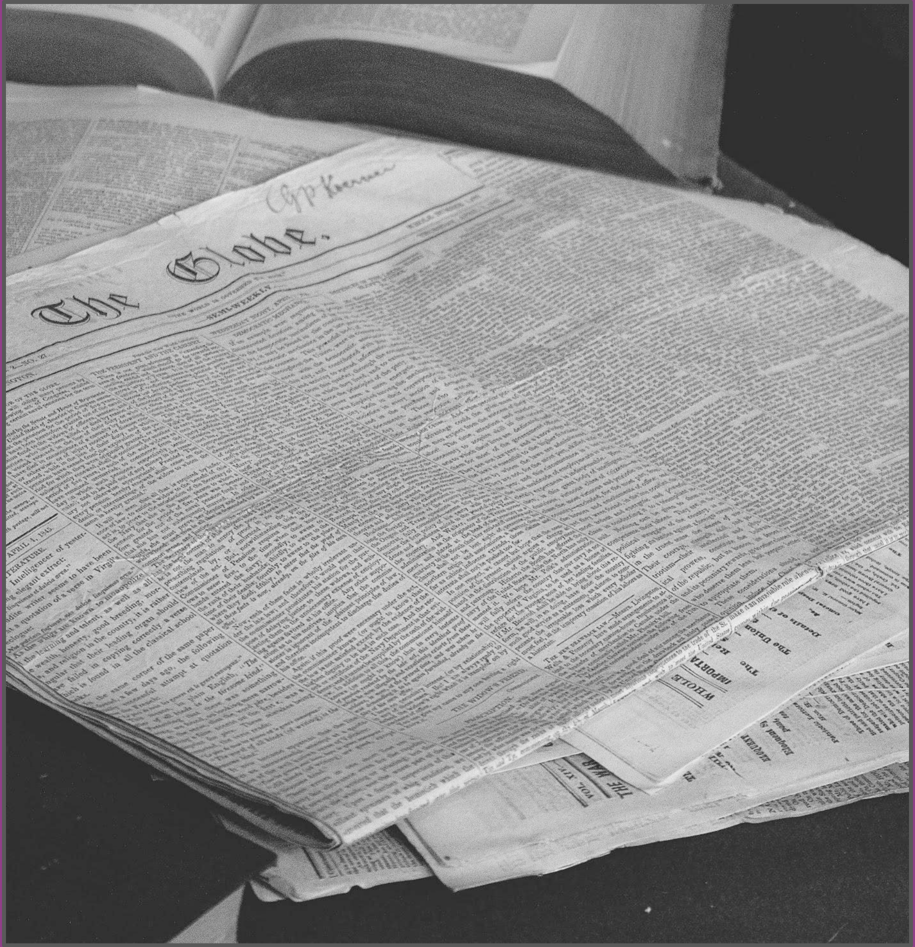
Event

The latest newspaper issue is hot off the press!

HOOK42

Event Context

A single issue of the newspaper containing stories, opinions, comics, etc



HOOK42

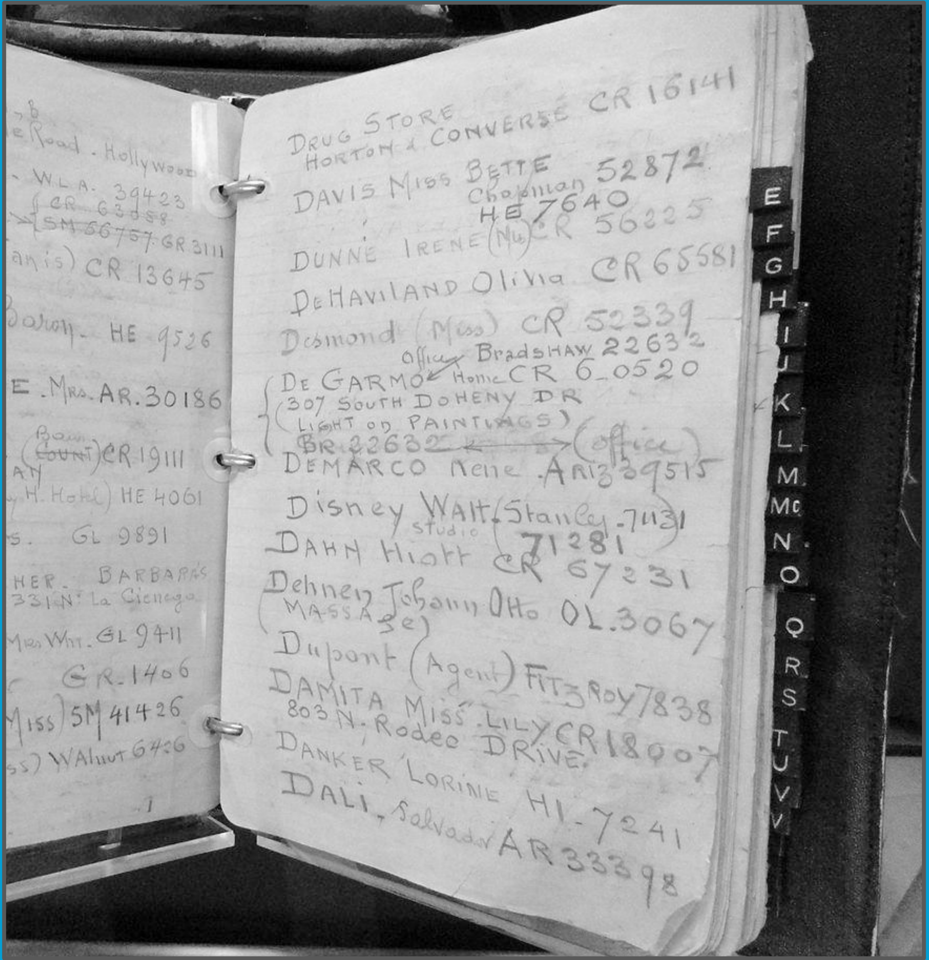
Event Subscribers

The homes that have paid for this edition of the newspaper

HOOK42

Event Registry

List of all homes that subscribe to this edition of the newspaper



…e Road . Hollywood
- W.L.A. 39423
~~CR 63058~~
~~SM 66757~~ GR 3111
…anis) CR 13645
Baron - HE 9526
E . Mrs. AR.30186
(~~COUNT~~ Baron) CR 19111
AN
…H. Hotel) HE 4061
…s. GL 9891
HER. BARBARA'S
331 N. La Cienega
Mrs Wm. GL 9411
GR.1406
Miss) SM 41426
…ss) WAlnut 6426

DRUG STORE
HORTON & CONVERSE CR 16141
DAVIS Miss BETTE 52872
Chapman HE 7640
DUNNE IRENE (Mrs) CR 56225
DeHAVILAND Olivia CR 65581
Desmond (Miss) CR 52339
office Bradshaw 22632
DE GARMO Home CR 6-0520
(307 SOUTH DOHENY DR
(LIGHT on PAINTINGS)
BR 22632 (office)
DEMARCO Rene ARiz 39515
Disney WAlt (Stanley 71131
studio 71281
DAHN Hiott CR 67231
Dehnen Johann Otto OL.3067
(MASSAGE)
Dupont (Agent) FITzRoy 7838
DAMITA Miss LILY CR 18007
803 N. Rodeo DRIVE
DANKER LORINE HI-7241
DALI, Salvador AR 33398

E F G H I J K L M Mc N O Q R S T U V V

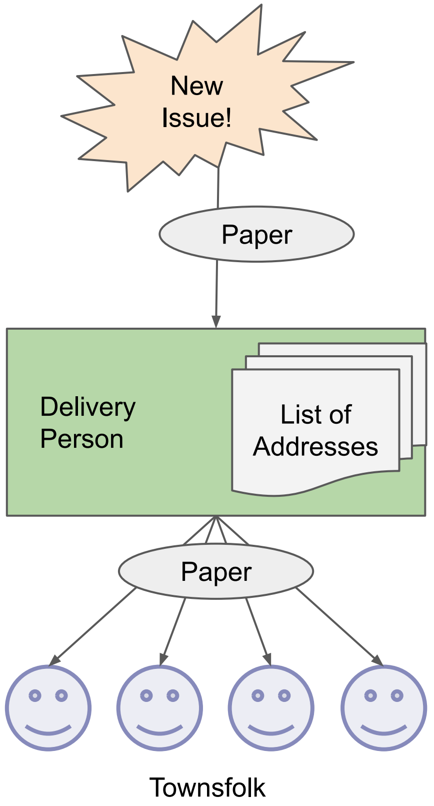
Event Dispatcher

Lil' Timmy

HOOK42

Overview: Newspaper as an Event System

- **Event**
  New issue of the paper comes out
- **Context**
  The issue: stories, opinions, comics, etc
- **Subscriber (aka, Listener)**
  Sleepy townsfolk, making a cup of coffee
- **Registry**
  List of newspaper subscribers
- **Dispatcher**
  Kid on a bike w/ bag of newspapers



Exploration of Event Systems

- Drupal Hooks
- WordPress Hooks
- Drupal 8 Events
- JavaScript

Drupal Hooks

Event, Subscriber, & Context

Drupal hooks are functions with specific names.

```
<module>_<hook name>()
<module>_<hook name>_alter()
```

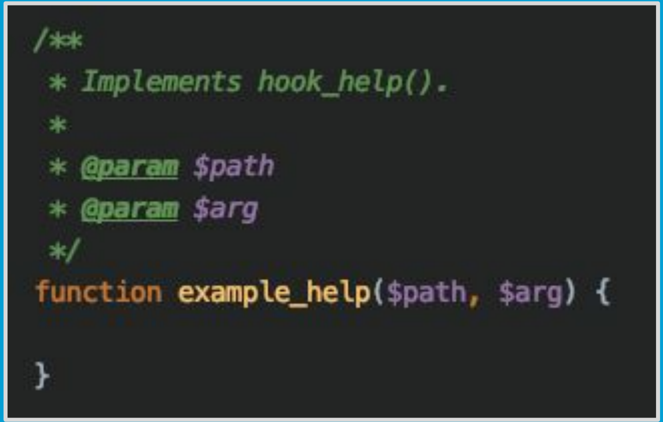
Event - "help"

Subscriber

The function named "**example_help**"

Context

All of the function parameters. In this case, "**$path**" and "**$arg**"



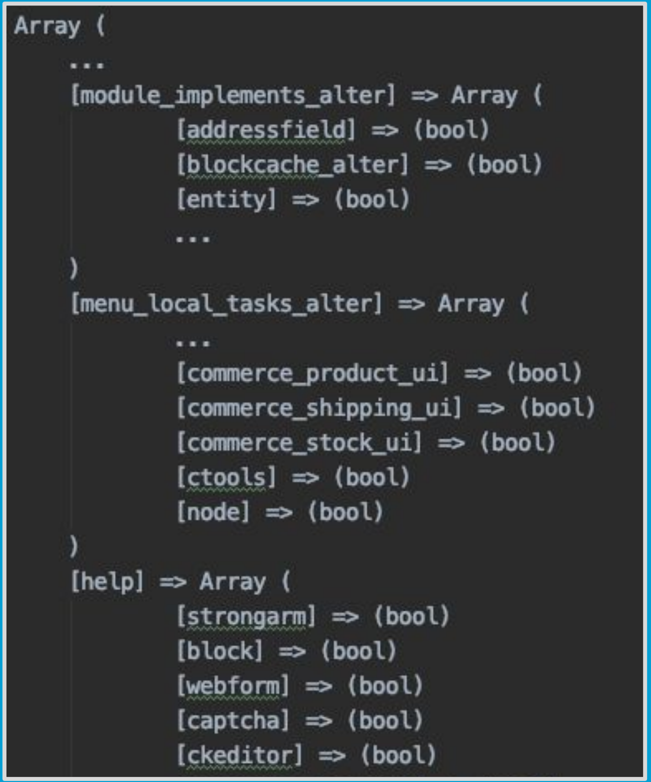
```
/**
 * Implements hook_help().
 *
 * @param $path
 * @param $arg
 */
function example_help($path, $arg) {

}
```

HOOK42

Event Registry

Drupal hooks registered as a serialized array in the **cache_bootstrap** table. This is why we must clear the site cache when adding new hooks.

```
SELECT data
FROM   cache_bootstrap
WHERE  cid='module_implements';
```



```
Array (
    ...
    [module_implements_alter] => Array (
            [addressfield] => (bool)
            [blockcache_alter] => (bool)
            [entity] => (bool)
            ...
    )
    [menu_local_tasks_alter] => Array (
            ...
            [commerce_product_ui] => (bool)
            [commerce_shipping_ui] => (bool)
            [commerce_stock_ui] => (bool)
            [ctools] => (bool)
            [node] => (bool)
    )
    [help] => Array (
            [strongarm] => (bool)
            [block] => (bool)
            [webform] => (bool)
            [captcha] => (bool)
            [ckeditor] => (bool)
```

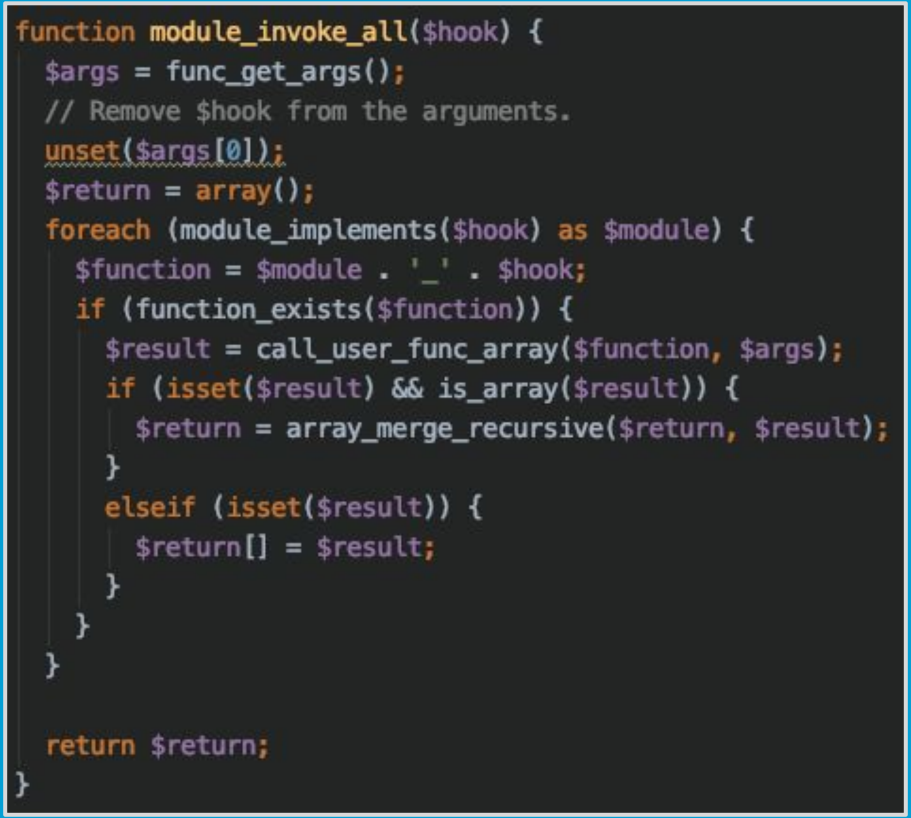
HOOK42

Event Dispatcher

module_invoke_all( ) looks in the registry for all subscribers to a hook, then calls each and provides the event context.

HOOK42

```
module_invoke_all('help', $path, $arg);
```



```
function module_invoke_all($hook) {
  $args = func_get_args();
  // Remove $hook from the arguments.
  unset($args[0]);
  $return = array();
  foreach (module_implements($hook) as $module) {
    $function = $module . '_' . $hook;
    if (function_exists($function)) {
      $result = call_user_func_array($function, $args);
      if (isset($result) && is_array($result)) {
        $return = array_merge_recursive($return, $result);
      }
      elseif (isset($result)) {
        $return[] = $result;
      }
    }
  }

  return $return;
}
```

WordPress Hooks

Event, Subscriber, & Context

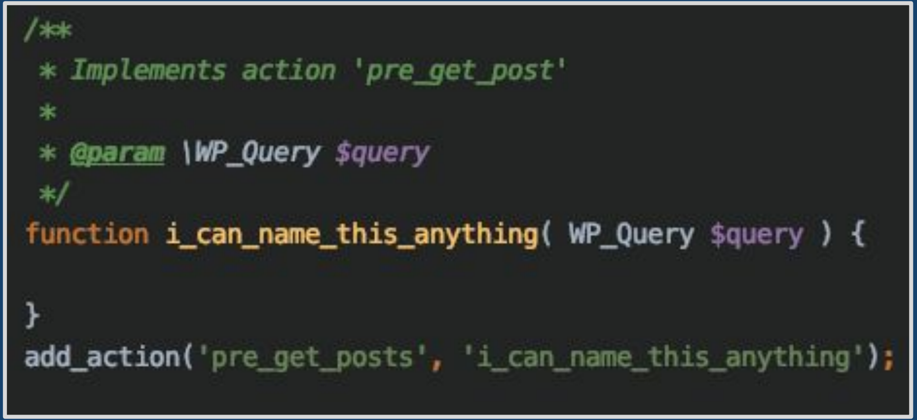
Event - “pre_get_post”

Subscriber

The function named “**i_can_name_this_anything**”

Context

All of the function parameters. In this case, “**$query**”



```
/**
 * Implements action 'pre_get_post'
 *
 * @param \WP_Query $query
 */
function i_can_name_this_anything( WP_Query $query ) {

}
add_action('pre_get_posts', 'i_can_name_this_anything');
```

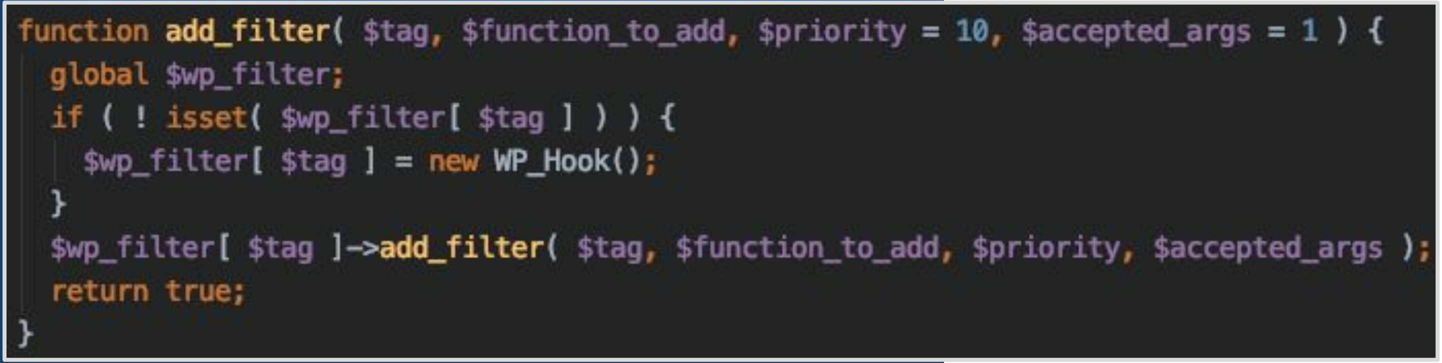
WordPress hooks are functions, methods, and closures that we register with these functions:

`add_action`( )
`add_filter`( )

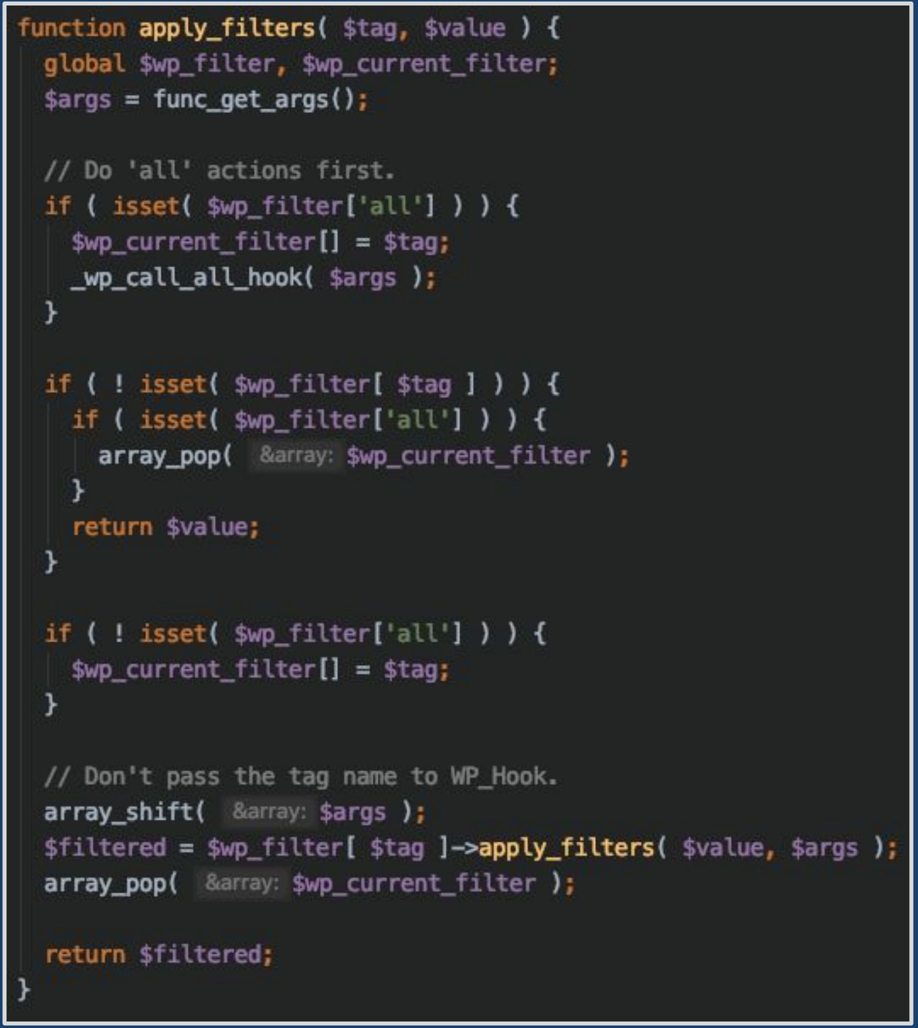
```php
<?php
// Show all subscribers in footer.
add_action( 'wp_footer', function() {
  global $wp_filter;
  var_dump($wp_filter);
} );
```

Event Registry

WordPress hooks are registered in a global array named **$wp_filter**



```php
function add_filter( $tag, $function_to_add, $priority = 10, $accepted_args = 1 ) {
  global $wp_filter;
  if ( ! isset( $wp_filter[ $tag ] ) ) {
    $wp_filter[ $tag ] = new WP_Hook();
  }
  $wp_filter[ $tag ]->add_filter( $tag, $function_to_add, $priority, $accepted_args );
  return true;
}
```

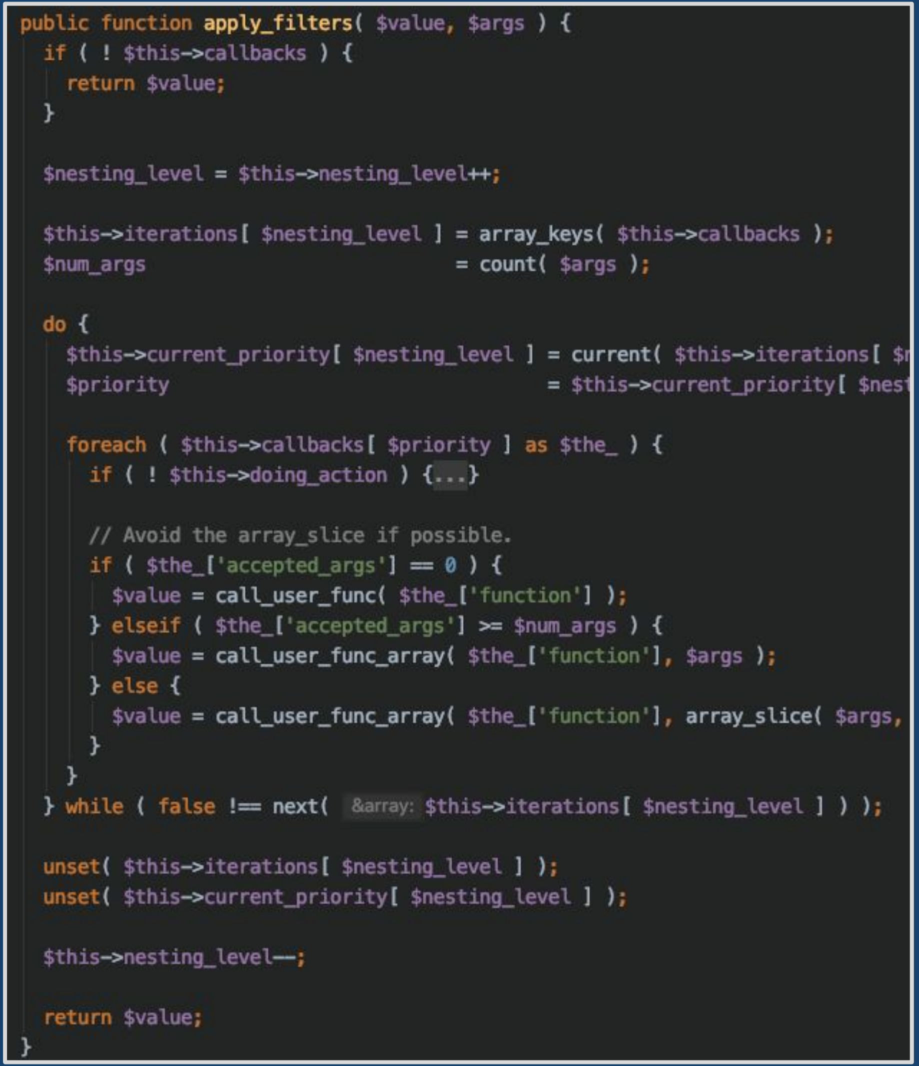


```
function apply_filters( $tag, $value ) {
  global $wp_filter, $wp_current_filter;
  $args = func_get_args();

  // Do 'all' actions first.
  if ( isset( $wp_filter['all'] ) ) {
    $wp_current_filter[] = $tag;
    _wp_call_all_hook( $args );
  }

  if ( ! isset( $wp_filter[ $tag ] ) ) {
    if ( isset( $wp_filter['all'] ) ) {
      array_pop( &array: $wp_current_filter );
    }
    return $value;
  }

  if ( ! isset( $wp_filter['all'] ) ) {
    $wp_current_filter[] = $tag;
  }

  // Don't pass the tag name to WP_Hook.
  array_shift( &array: $args );
  $filtered = $wp_filter[ $tag ]->apply_filters( $value, $args );
  array_pop( &array: $wp_current_filter );

  return $filtered;
}
```

Event Dispatcher

`do_action`( )
`apply_filters`( )

both look at the global **$wp_filter** variable for subscribers to the hook, then calls each and provides the event context.



```
public function apply_filters( $value, $args ) {
  if ( ! $this->callbacks ) {
    return $value;
  }

  $nesting_level = $this->nesting_level++;

  $this->iterations[ $nesting_level ] = array_keys( $this->callbacks );
  $num_args                           = count( $args );

  do {
    $this->current_priority[ $nesting_level ] = current( $this->iterations[ $n
    $priority                                 = $this->current_priority[ $nest

    foreach ( $this->callbacks[ $priority ] as $the_ ) {
      if ( ! $this->doing_action ) {...}

      // Avoid the array_slice if possible.
      if ( $the_['accepted_args'] == 0 ) {
        $value = call_user_func( $the_['function'] );
      } elseif ( $the_['accepted_args'] >= $num_args ) {
        $value = call_user_func_array( $the_['function'], $args );
      } else {
        $value = call_user_func_array( $the_['function'], array_slice( $args,
      }
    }
  } while ( false !== next( &array: $this->iterations[ $nesting_level ] ) );

  unset( $this->iterations[ $nesting_level ] );
  unset( $this->current_priority[ $nesting_level ] );

  $this->nesting_level--;

  return $value;
}
```

Event Dispatcher Continued…

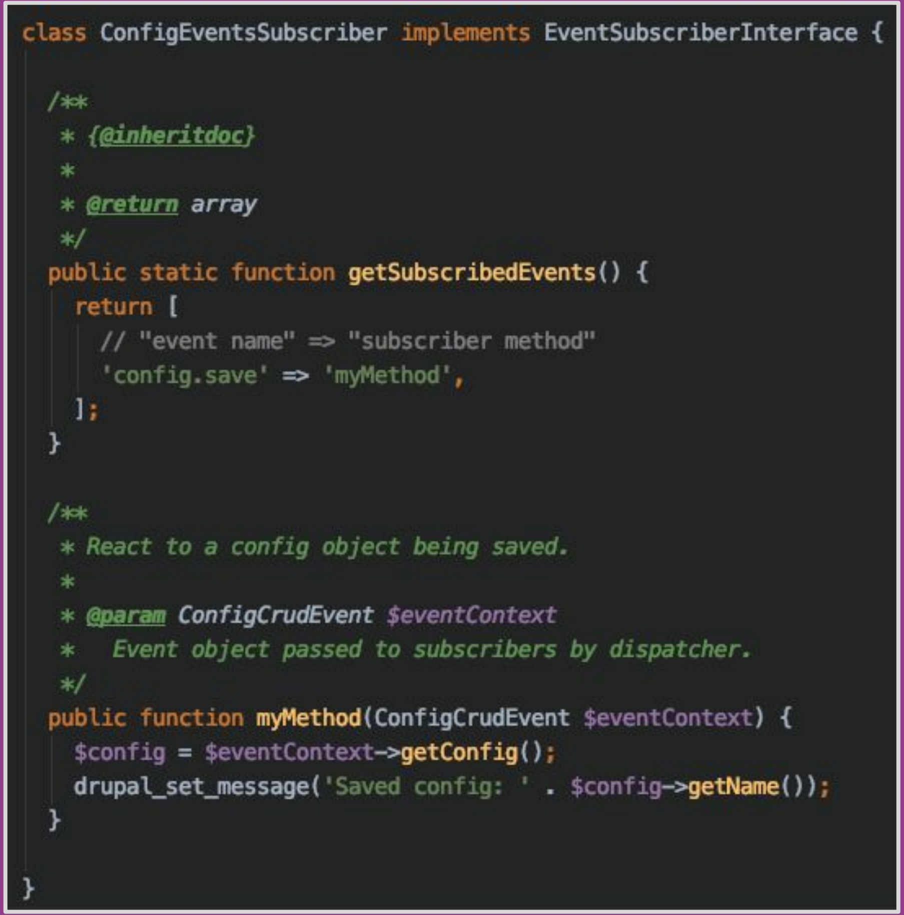
WP_Hook::apply_filters( )

loops over the list of subscribers (**$callbacks**) and calls each, providing the event context (**$args**)

Drupal 8 Events (Symfony)

Event, Subscriber, & Context

A Symfony event subscriber is a class with methods and a list of events those methods listen to. Context is often an event-specific object containing useful data and methods about the event.

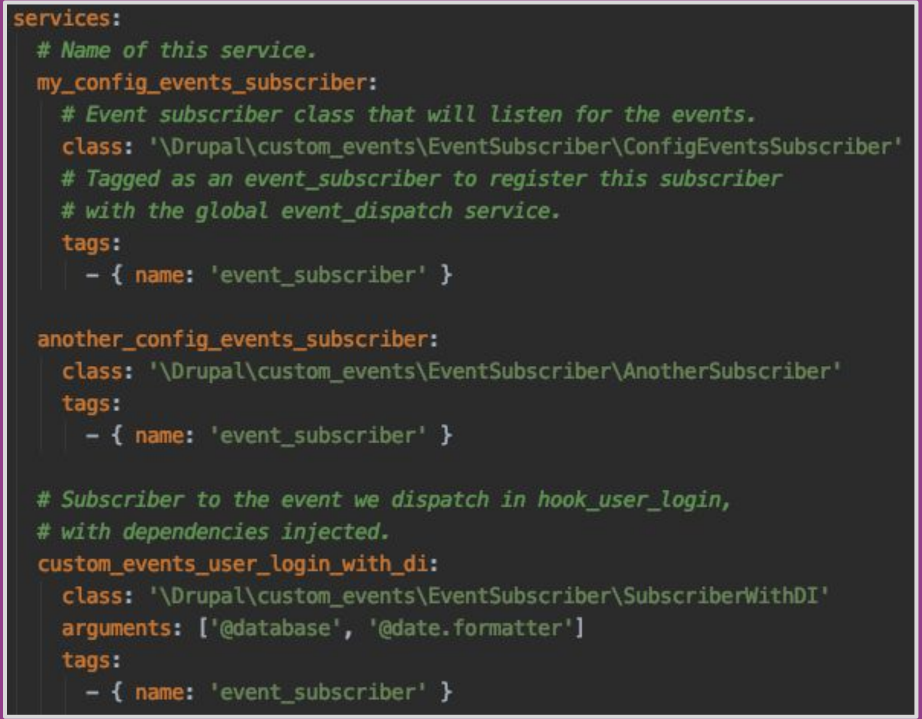


```
class ConfigEventsSubscriber implements EventSubscriberInterface {

  /**
   * {@inheritdoc}
   *
   * @return array
   */
  public static function getSubscribedEvents() {
    return [
      // "event name" => "subscriber method"
      'config.save' => 'myMethod',
    ];
  }

  /**
   * React to a config object being saved.
   *
   * @param ConfigCrudEvent $eventContext
   *   Event object passed to subscribers by dispatcher.
   */
  public function myMethod(ConfigCrudEvent $eventContext) {
    $config = $eventContext->getConfig();
    drupal_set_message('Saved config: ' . $config->getName());
  }

}
```

HOOK42

Registering an Event Subscriber

Event subscribers are registered as a **symfony service**, tagged with an object named "**event_subscriber**"

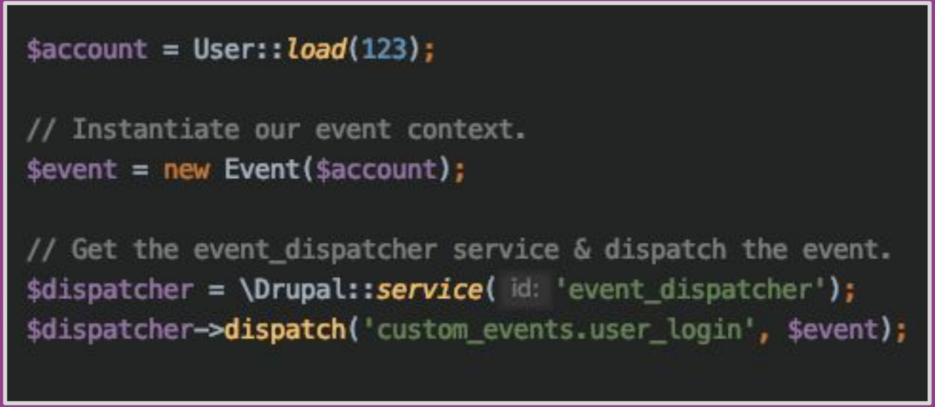


```yaml
services:
  # Name of this service.
  my_config_events_subscriber:
    # Event subscriber class that will listen for the events.
    class: '\Drupal\custom_events\EventSubscriber\ConfigEventsSubscriber'
    # Tagged as an event_subscriber to register this subscriber
    # with the global event_dispatch service.
    tags:
      - { name: 'event_subscriber' }

  another_config_events_subscriber:
    class: '\Drupal\custom_events\EventSubscriber\AnotherSubscriber'
    tags:
      - { name: 'event_subscriber' }

  # Subscriber to the event we dispatch in hook_user_login,
  # with dependencies injected.
  custom_events_user_login_with_di:
    class: '\Drupal\custom_events\EventSubscriber\SubscriberWithDI'
    arguments: ['@database', '@date.formatter']
    tags:
      - { name: 'event_subscriber' }
```

HOOK42

Event Dispatcher Service

Drupal provides a global instance of the dispatcher as a service named "**event_dispatcher**". This dispatcher is where subscribers are registered when defined in a module's ***.services.yml** file.



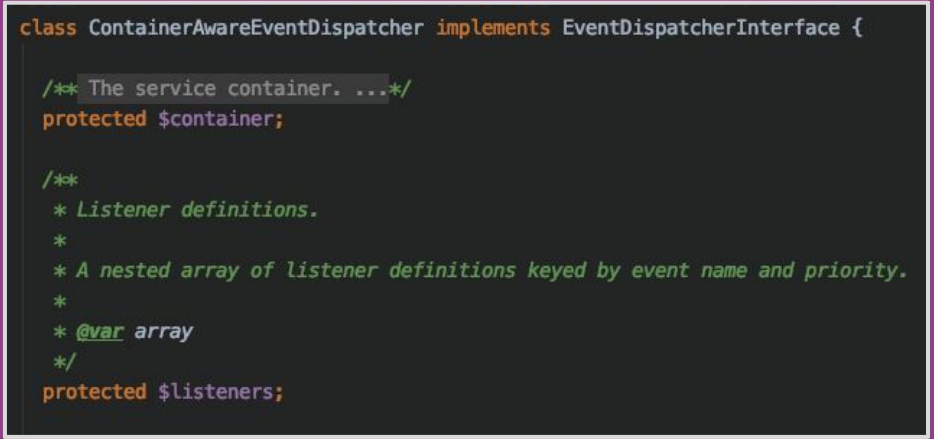
```php
$account = User::load(123);

// Instantiate our event context.
$event = new Event($account);

// Get the event_dispatcher service & dispatch the event.
$dispatcher = \Drupal::service( id: 'event_dispatcher');
$dispatcher->dispatch('custom_events.user_login', $event);
```

HOOK42

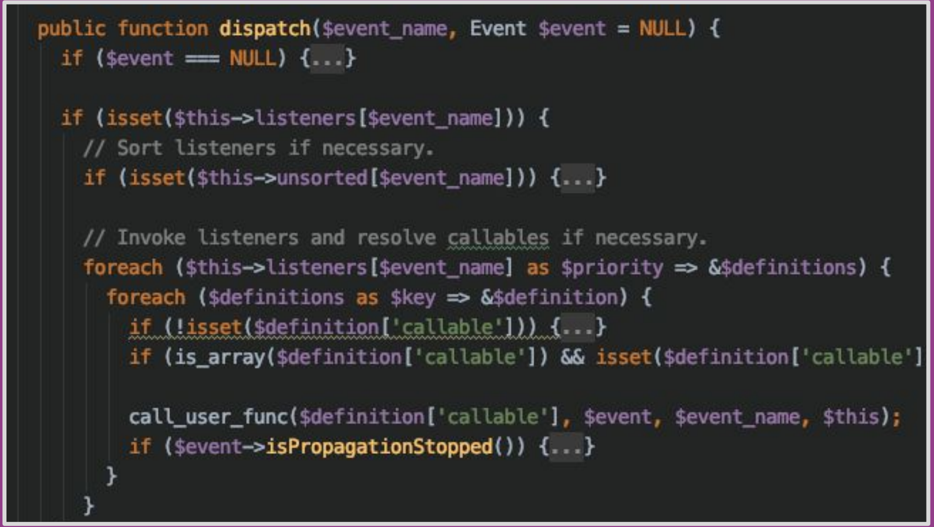
Event Registry & Dispatcher

ContainerAwareEventDispatcher contains its registry on a property named “**$listeners**” (array).
The **dispatch()** method loops through the array and calls each subscriber with the **$event** context

HOOK42

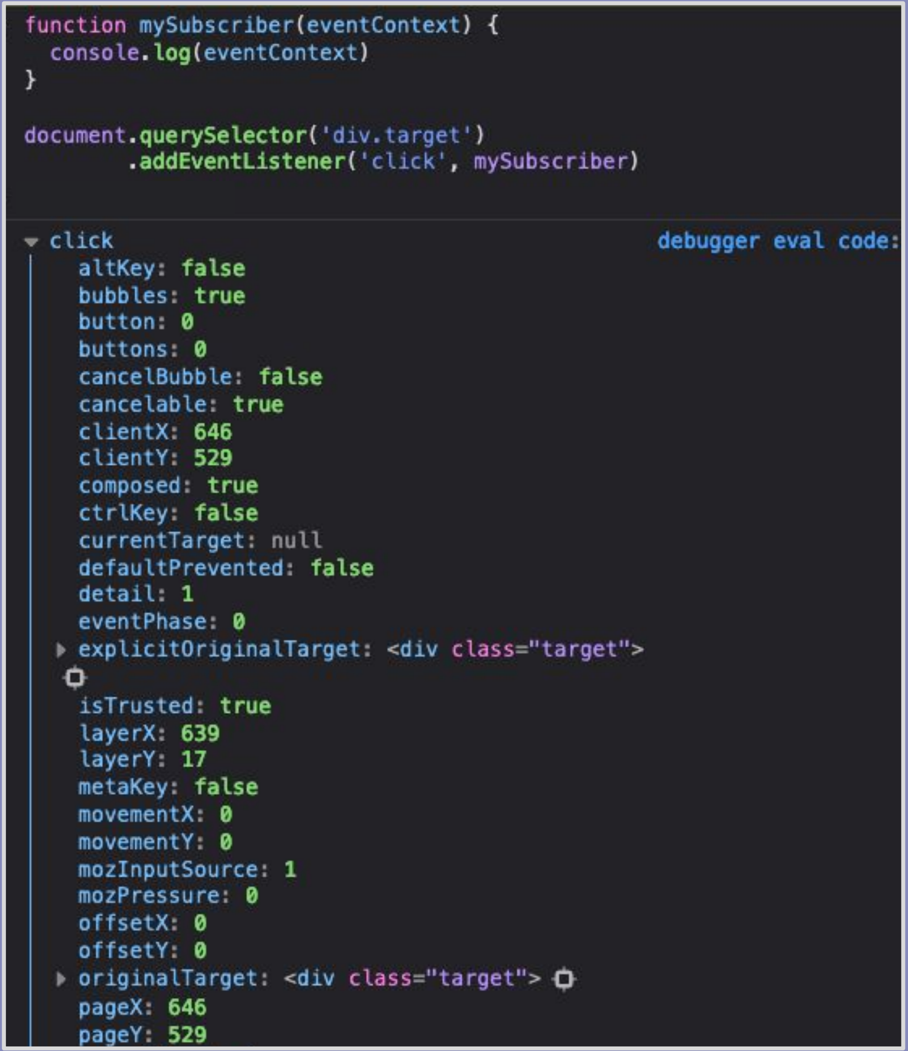


```
class ContainerAwareEventDispatcher implements EventDispatcherInterface {

  /** The service container. ...*/
  protected $container;

  /**
   * Listener definitions.
   *
   * A nested array of listener definitions keyed by event name and priority.
   *
   * @var array
   */
  protected $listeners;
```



```
  public function dispatch($event_name, Event $event = NULL) {
    if ($event === NULL) {...}

    if (isset($this->listeners[$event_name])) {
      // Sort listeners if necessary.
      if (isset($this->unsorted[$event_name])) {...}

      // Invoke listeners and resolve callables if necessary.
      foreach ($this->listeners[$event_name] as $priority => &$definitions) {
        foreach ($definitions as $key => &$definition) {
          if (!isset($definition['callable'])) {...}
          if (is_array($definition['callable']) && isset($definition['callable']

          call_user_func($definition['callable'], $event, $event_name, $this);
          if ($event->isPropagationStopped()) {...}
        }
      }
```

JavaScript Events (Web)

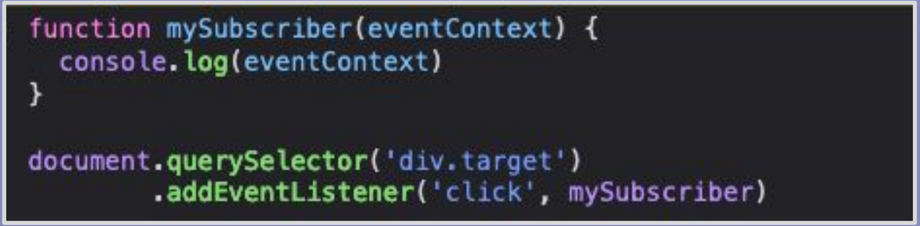


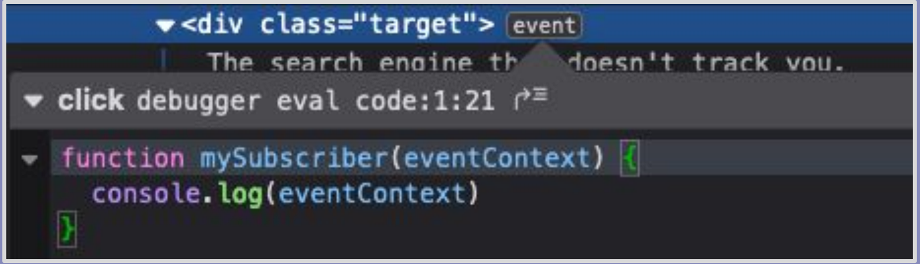
```
function mySubscriber(eventContext) {
  console.log(eventContext)
}

document.querySelector('div.target')
        .addEventListener('click', mySubscriber)
```

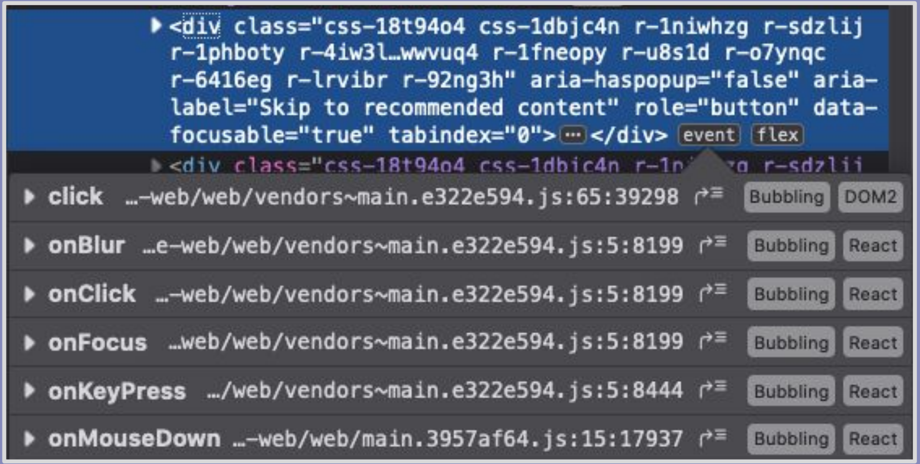
```
▾ click                                              debugger eval code:
    altKey: false
    bubbles: true
    button: 0
    buttons: 0
    cancelBubble: false
    cancelable: true
    clientX: 646
    clientY: 529
    composed: true
    ctrlKey: false
    currentTarget: null
    defaultPrevented: false
    detail: 1
    eventPhase: 0
  ▸ explicitOriginalTarget: <div class="target">
    isTrusted: true
    layerX: 639
    layerY: 17
    metaKey: false
    movementX: 0
    movementY: 0
    mozInputSource: 1
    mozPressure: 0
    offsetX: 0
    offsetY: 0
  ▸ originalTarget: <div class="target">
    pageX: 646
    pageY: 529
```

Event, Subscriber, & Context

Subscribing to events in JavaScript involves adding functions as "**listeners**" to DOM elements. The **event context** is an **object** passed into the listener function.


```
function mySubscriber(eventContext) {
  console.log(eventContext)
}

document.querySelector('div.target')
        .addEventListener('click', mySubscriber)
```


```
▾<div class="target"> event
   The search engine th  doesn't track you.
▾ click debugger eval code:1:21
▾ function mySubscriber(eventContext) {
    console.log(eventContext)
  }
```

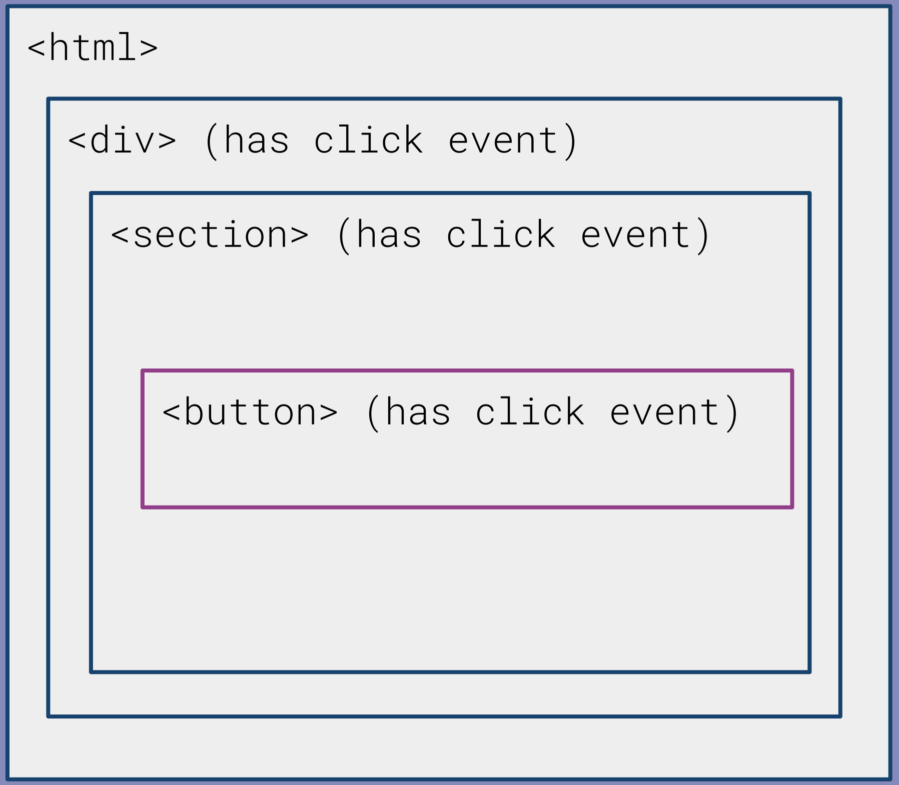

```
▸<div class="css-18t94o4 css-1dbjc4n r-1niwhzg r-sdzlij
 r-1phboty r-4iw3l…wwvuq4 r-1fneopy r-u8s1d r-o7ynqc
 r-6416eg r-lrvibr r-92ng3h" aria-haspopup="false" aria-
 label="Skip to recommended content" role="button" data-
 focusable="true" tabindex="0">…</div> event flex
▸<div class="css-18t94o4 css-1dbjc4n r-1n   g r-sdzlij
▸ click  …-web/web/vendors~main.e322e594.js:65:39298   Bubbling DOM2
▸ onBlur  …e-web/web/vendors~main.e322e594.js:5:8199   Bubbling React
▸ onClick  …-web/web/vendors~main.e322e594.js:5:8199   Bubbling React
▸ onFocus  …web/web/vendors~main.e322e594.js:5:8199    Bubbling React
▸ onKeyPress  …/web/vendors~main.e322e594.js:5:8444    Bubbling React
▸ onMouseDown …-web/web/main.3957af64.js:15:17937      Bubbling React
```

Event Registry

The DOM is the event registry for JavaScript web events.
Functions are registered to elements, the document, or window with the use of `addEventListener()`

Event Dispatcher

Web events are not a part of the core JavaScript language — they are agreed upon (mostly) APIs built into browsers. **Browsers** detect and **dispatch events** to DOM-registered **subscribers**



Gotcha! Nested Subscribers

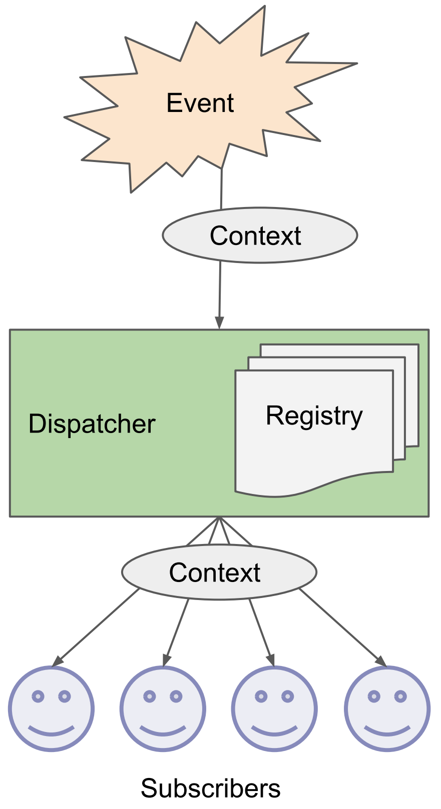
Since DOM elements are event subscribers, then subscribers can be nested within other subscribers.

`Event.stopPropagation()` to the rescue.

Recap! 😅

WHAT WE LEARNED

- Regardless of the framework, event systems **share common concepts**.
- **Subscribers** are functions.
- **Registries** are a collection subscribers, mapped to event names.
- **Context** is just data that subscribers may need to understand the event.
- **Dispatchers** loop through the registry, call subscribers, and provide them context.



Questions?

HOOK42

Thanks!

References

- Drupal: Hooks, Events, and Event Subscribers
- WordPress: Hooks
  - Plugin API/Action Reference
  - Plugin API/Filter Reference « WordPress
- Symfony: Events and Event Listeners
- JavaScript: Introduction to events - Learn web development
- Patterns:
  - 3.4. Mediator
  - 3.7. Observer