JavaScript Events and more

Andrew Lyons
Senior Analyst / Developer / Integrator

JS in Moodle - brief history

When	Version	Format	Loading	Building
2001	1.0	Native JS	Custom loading	
2006	1.7?	YUI2	YUI Loader	
2009	2.0	YUI3	YUI Loader	
2012	2.5	YUI3 in Module format	YUI Loader	Shifter for dev and production
2015	2.9	AMD Modules	RequireJS	Grunt for production
2018	3.8	ES Modules	RequireJS	Grunt for dev and production

JS in Moodle - current recommendations

- All new JS code should be in an ES module (since 3.8)
- `const > let > ~~var~~`
- Use named exports unless exporting a single class
- Listen to eslint
- Consider using a `repository.js` file for your fetches
- Make use of subdirectories to organise code (since 3.8)
- Don't use jQuery unless you really have to
- Use `then/catch` (Not `done/fail`)

moodle

Why the jQuery hate?

- Not spec-compliant
- Not necessary
- Not maintained
- No real benefit with modern JavaScript features

Why can’t I use done/fail?

- They are jQuery features, and do not comply with the specification for Promises
- Make it harder for code to be updated to not use jQuery
- Have a slightly different behaviour in error handling
- Not understood properly by eslint

Named exports? Default exports? Huh?

- Feature of ES modules
- Can usually co-exist
- But not when transpiled to AMD :(

```
// some_module.js
export const doSomething = (list, of, params) => {
    doThis(list);
    doThat(of);
    doTheOther(params);
};

// some_other_module.js
import {doSomething} from './some_module';
doSomething('foo', 'bar', 'baz');

import * as someModule from './some_module';
someModule.doSomething('foo', 'bar', 'baz');
```

moodle

```
// some_module.js
export default (list, of, params) => {
    doThis(list);
    doThat(of);
    doTheOther(params);
};


// some_other_module.js
import doSomething from './some_module';
doSomething('foo', 'bar', 'baz');
```

```
// some_module.js
import SomeOtherClass from './some_other_class';
export default class extends SomeOtherClass {
    doSomething(list, of, params) {
        this.doThis(list);
        this.doThat(of);
        this.doTheOther(params);
    }
}

// some_other_module.js
import SomeModuleClass from './some_module';
const myModule = new SomeModuleClass();
myModule.doSomething('foo', 'bar', 'baz');
```

Tell me about this subdirectory thing...

- Possible since Moodle 3.8
- Follows the same rules as class subdirectories:
  - First subdirectory must be a subsystem or `local`
  - Free reign in Second-level directories
- Really helpful
- ES modules can load them relatively:

```
import {fetchNotifications} from ‘./local/myfeature/repository’;
```

moodle

Examples

```
// user/amd/src/repository.js
import {call as fetchMany} from 'core/ajax';

/**
 * Unenrol the user with the specified user enrolmentid ID.
 *
 * @param {Number} ueid The user enrolment ID
 * @return {Promise}
 */
export const unenrolUser = ueid => {
    return fetchMany([{
        methodname: 'core_enrol_unenrol_user_enrolment',
        args: {
            ueid,
        },
    }])[0];
};
```

```
import {call as fetchMany} from 'core/ajax';

/**
 * Unenrol the user with the specified user enrolmentid ID.
 *
 * @param {Number} ueid The user enrolment ID
 * @return {Promise}
 */
export const unenrolUser = ueid => fetchMany([{
    methodname: 'core_enrol_unenrol_user_enrolment',
    args: {
        ueid,
    },
}])[0];


// user/amd/src/some_feature.js
import * as userRepository from './repository';
import {unenrolUser} from 'core_user/repository';
```

```
// user/amd/src/status_field.js

import * as Repository from './repository';

const submitUnenrolFormAjax = (clickedLink, modal, args, userData) => {
    Repository.unenrolUser(args.ueid)
    .then(data => {
        if (!data.result) {
            // Display an alert containing the error message
            Notification.alert(data.errors[0].key, data.errors[0].message);

            return data;
        }

        // Dismiss the modal.
        modal.hide();
        modal.destroy();

        return data;
    })
    .then(() => {
        DynamicTable.refreshTableContent(getDynamicTableFromLink(clickedLink))
        .catch(Notification.exception);

        return Str.get_string('unenrolleduser', 'core_enrol', userData);
    })
    .then(notificationString => {
        notifyUser(notificationString);

        return;
    })
    .catch(Notification.exception);
};
```

```
// lib/amd/src/event_dispatcher.js

export const dispatchEvent = (
    eventName,
    detail = {},
    container = document,
    {
        bubbles = true,
        cancelable = false,
        composed = false,
    } = {}
) => {
    const customEvent = new CustomEvent(
        eventName,
        {
            bubbles,
            cancelable,
            composed,
            detail,
        }
    );

    container.dispatchEvent(customEvent);

    return customEvent;
};
```

```
// lib/amd/src/event_dispatcher.js

export const dispatchEvent = (
    eventName,
    detail = {},
    container = document,
    {
        bubbles = true,
        cancelable = false,
        composed = false,
    } = {}
) => {
    const customEvent = new CustomEvent(
        eventName,
        {
            bubbles,
            cancelable,
            composed,
            detail,
        }
    );

    container.dispatchEvent(customEvent);

    return customEvent;
};
```

```
// filter/amd/src/events.js

import {dispatchEvent} from 'core/event_dispatcher';

export const notifyFilterContentUpdated = nodes => {
    // Historically this could be a jQuery Object.
    // Normalise the list of nodes to a NodeList.
    nodes = normalistNodeList(nodes);

    return dispatchEvent(eventTypes.filterContentUpdated, {nodes});
};
```

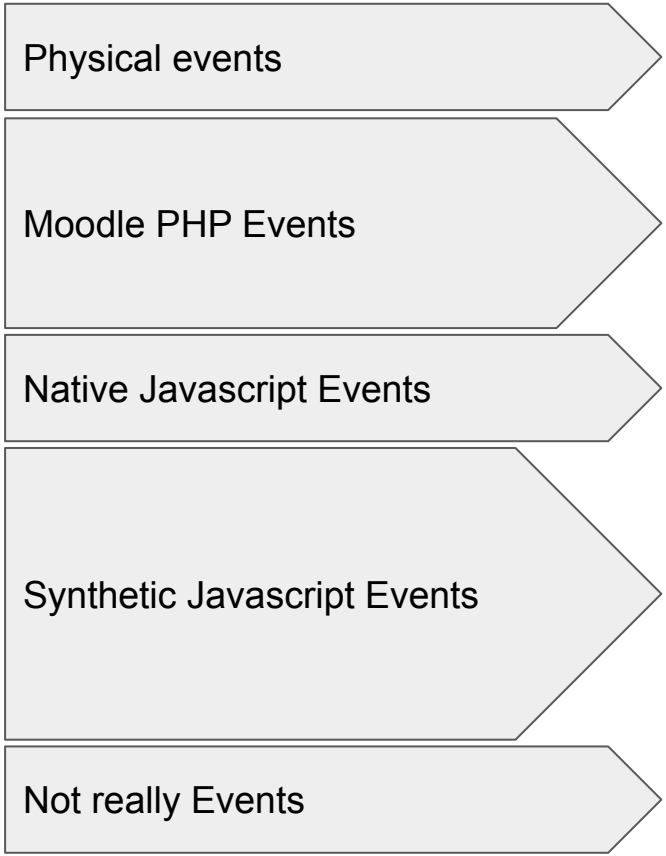
```
// lib/amd/src/event_dispatcher.js

export const dispatchEvent = (
    eventName,
    detail = {},
    container = document,
    {
        bubbles = true,
        cancelable = false,
        composed = false,
    } = {}
) => {
    const customEvent = new CustomEvent(
        eventName,
        {
            bubbles,
            cancelable,
            composed,
            detail,
        }
    );

    container.dispatchEvent(customEvent);

    return customEvent;
};
```

```
export const dispatchEvent = (
    eventName,
    detail,
    container,
    config
    {
        bubbles = true,
        cancelable = false,
        composed = false,
    } = {}
) => {
    if (typeof detail === 'undefined') {
        detail = {};
    }

    if (typeof container === 'undefined') {
        container = document;
    }

    if (typeof config === 'undefined') {
        config = {};
    }

    if (!config.hasOwnProperty('bubbles')) {
        config.bubbles = true;
    }
    if (!config.hasOwnProperty('cancelable')) {
        config.cancelable = false;
    }
    if (!config.hasOwnProperty('composed')) {
        config.composed = false;
    }

    const customEvent = new CustomEvent(
        eventName,
        {
            bubbles,
            cancelable,
            composed,
            detail,
        }
    );

    container.dispatchEvent(customEvent);

    return customEvent;
};
```

Events

Let’s talk about Events

- Moots
- Events API
- Events 2 API
- Native DOM Events
- YUI3 Events
- jQuery Events
- Native Custom Events
- PubSub

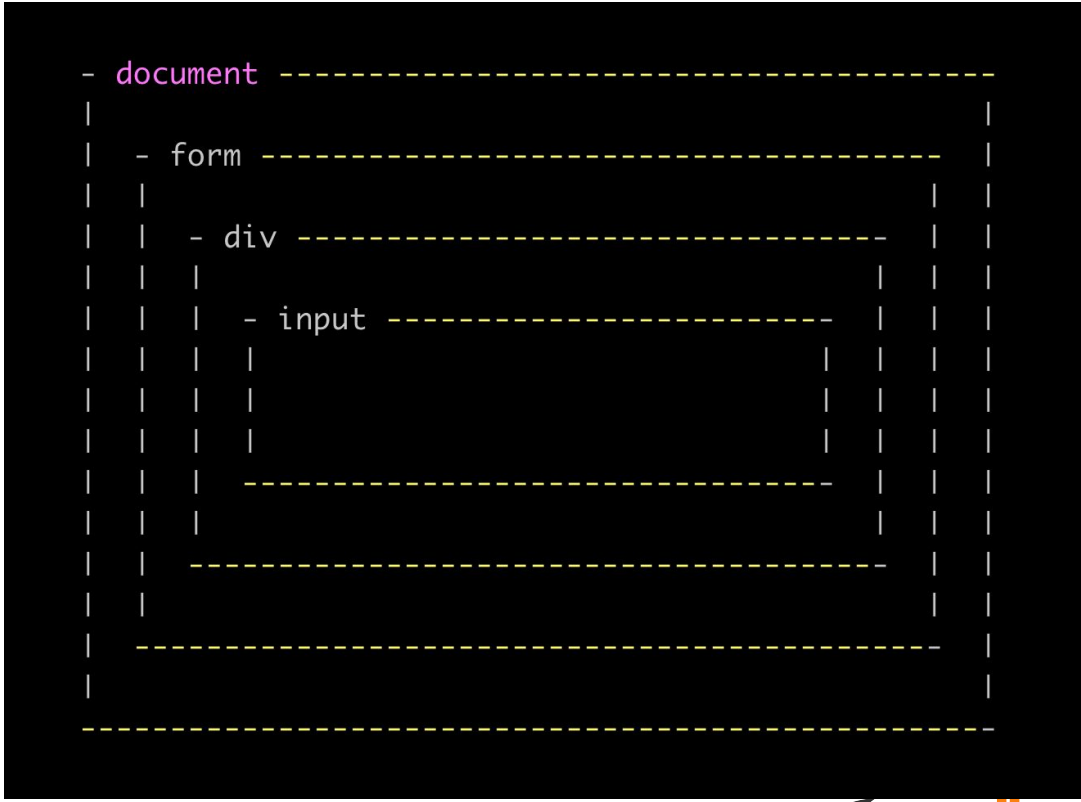


Moodle Moots

- A very popular kind of event
- Lots of networking opportunities
- Not relevant to this discussion
- Good food
- Good wifi

Terminology: Bubbles

Most Events bubble up from child DOM Elements to their parent



Terminology: Delegation

Listen to an event on a parent element.

If you have multiple elements you can listen once for all `click` elements and filter.

Important for performance

Terminology: Cancelable

Some events can be cancelled

`event.cancelable (bool)`

Cancel by calling `event.preventDefault()`

`event.defaultPrevented (bool)` current state

Once cancelled, cannot be uncancelled

Native DOM Events

Things like:

- `click`
- `mousemove`
- `mouseover`
- `mousedown/mouseup`
- `keydown/keyup/key`

- Not all events are cancellable
- Cancellable events prevent the default behaviour
- *Most* bubble and can be delegated

Native CustomEvent

- Name must be a string
- Configurable to support:
  - Bubbling
  - Cancellable
  - Extra args
- Can be fired on any NodeElement
- Available in all supported browsers

Native CustomEvent

```
> const someAnchor = document.querySelector('a');

  document.addEventListener('someEvent', window.console.log);

  someAnchor.dispatchEvent(new CustomEvent('someEvent', {
    bubbles: true,
    detail: {extra: 'arguments'},
  }));
```

```
▼ CustomEvent {isTrusted: false, detail: {…}, type: "someEvent
  document, …} i
    bubbles: true
    cancelBubble: false
    cancelable: false
    composed: false
    currentTarget: null
    defaultPrevented: false
  ▶ detail: {extra: "arguments"}
    eventPhase: 0
    isTrusted: false
  ▶ path: (7) [a.sr-only.sr-only-focusable, div, div#page-wrap
    returnValue: true
  ▶ srcElement: a.sr-only.sr-only-focusable
  ▶ target: a.sr-only.sr-only-focusable
    timeStamp: 7925.309999845922
    type: "someEvent"
  ▶ __proto__: CustomEvent
```

YUI3 Custom Events

```
Y.on('eventname', eventHandler);
Y.fire('eventname', {some: 'data'});

Y.Global.on('eventname', eventHandler);
Y.Global.fire('eventname', {some: 'data'})

Y.one('.foo').on('eventname', eventHandler);
Y.delegate('eventname', eventHandler, '.foo');
```

jQuery Events

```
// Listen on node
$('a').on('click', console.log);

// Delegated
$('document').on('click', 'a', console.log);
```

moodle

jQuery Custom Events

```
// Custom events
$('document').on('someEvent', (e, extraArgs) => {
  console.log(e.detail); // undefined
  console.log(extraArgs); // {extra: 'arguments'}
});

$('a').trigger('someEvent', {extra: 'arguments'});
```

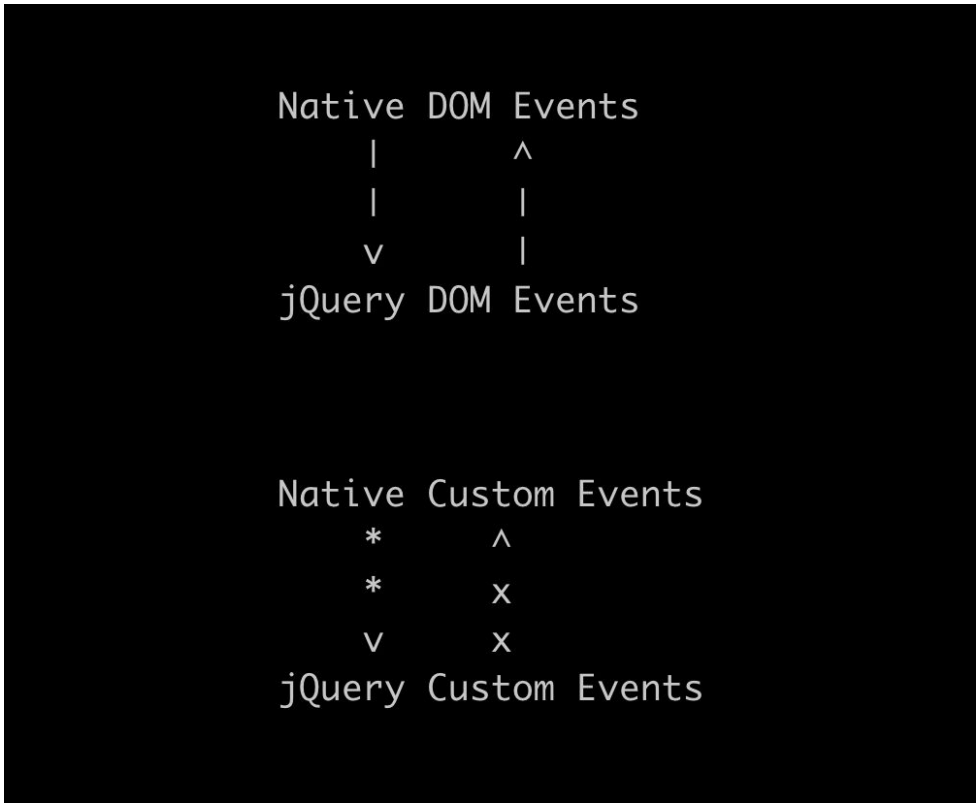
moodle

jQuery Custom Events

```
// Custom events
$('document').on('someEvent', (e, extraArgs) => {
  console.log(e.detail); // {extra: 'arguments'}
  console.log(extraArgs); // undefined
});

$('a')[0].dispatchEvent(new CustomEvent('someEvent', {
    bubbles: true,
    detail: {extra: 'arguments'},
});
```

moodle

jQuery vs Native events



More Moodle events

- We have `core/event` AMD module
  - Currently triggers a mixture of jQuery and YUI events
  - Breaks single-component principle
  - Heavily centralised
  - Hard-tied to jQuery
- We also have `core/pubsub` AMD module:
  - Overly simplistic
  - Centralised but does not break single-component principle
  - Works purely on knowing what events exist

Enter MDL-70990

- Deprecates all core uses of Custom YUI events
- Starts to deprecate core uses of Custom jQuery events
- Deprecates `core/event` module usage
- De-centralises
- Uses Native CustomEvent configured to bubble
- Includes `core/event_dispatcher` AMD module helper
- Encourages documentation of available event types

Going forward

- Create an `your_component/events` module centrally in your component
- Create an `eventTypes` object to map event names
- Create notify functions in your events module
- Use native (or jQuery) event listeners

What it looks like

```
// filter/amd/src/events.js
// core_filter/events

/**
 * Events for the `core_filter` subsystem.
 *
 * @constant
 * @property {String} filterContentUpdated See {@link event:filterContentUpdated}
 */
export const eventTypes = {
    /**
     * An event triggered when page content is updated and must be processed by the filter system.
     *
     * An example of this is loading user text that could have equations in it. MathJax can typeset the equations but
     * only if it is notified that there are new nodes in the page that need processing.
     *
     * @event filterContentUpdated
     * @type {CustomEvent}
     * @property {object} detail
     * @property {NodeElement[]} detail.nodes The list of parent nodes which were updated
     */
    filterContentUpdated: 'core_filters/contentUpdated',
};
```

moodle

```
// filter/amd/src/events.js
// core_filter/events

import {dispatchEvent} from 'core/event_dispatcher';
import {getList as normalistNodeList} from 'core/normalise';

/**
 * Trigger an event to indicate that the specified nodes were updated and should be processed by the filter system.
 *
 * @method notifyFilterContentUpdated
 * @param {jQuery|Array} nodes
 * @returns {CustomEvent}
 * @fires filterContentUpdated
 */
export const notifyFilterContentUpdated = nodes => {
    // Historically this could be a jQuery Object.
    // Normalise the list of nodes to a NodeList.
    nodes = normalistNodeList(nodes);

    return dispatchEvent(eventTypes.filterContentUpdated, {nodes});
};
```

```
// lib/amd/src/templates.js
// core/templates

// Notify all filters about the new content.
filterEvents.notifyFilterContentUpdated(newNodes);
```

```
// media/player/videojs/amd/src/loader.js
// media_videojs/loader
import {eventTypes} from 'core_filters/events';

/**
 * Initialise the videojs Loader.
 *
 * Adds the listener for the event to then notify video.js.
 *
 * @method
 * @param {string} lang Language to be used in the player
 * @listens event:filterContentUpdated
 */
export const setUp = (lang) => {
    language = lang;
    firstLoad = true;

    // Notify Video.js about the nodes already present on the page.
    notifyVideoJS({
        detail: {
            nodes: document.body,
        }
    });

    // We need to call popover automatically if nodes are added to the page later.
    document.addEventListener(eventTypes.filterContentUpdated, notifyVideoJS);
};
```

Events

filterContentUpdated

Source: filter/amd/src/events.js, line 35

Properties:

Name	Type	Description
detail	object	*Properties* Name: nodes, Type: Array.<NodeElement>, Description: The list of parent nodes which were updated

An event triggered when page content is updated and must be processed by the filter system.

An example of this is loading user text that could have equations in it. MathJax can typeset the equations but only if it is notified that there are new nodes in the page that need processing.

Type:

- CustomEvent

Listeners of This Event:

- module:media_videojs/loader.setUp

```
import jQuery from 'jquery';

let legacyEventsRegistered = false;
if (!legacyEventsRegistered) {
    // The following event triggers are legacy and will be removed in the future.
    // The following approach provides a backwards-compatability layer for the new events.
    // Code should be updated to make use of native events.

    Y.use('event', 'moodle-core-event', () => {
        // Provide a backwards-compatability layer for YUI Events.
        document.addEventListener(eventTypes.filterContentUpdated, e => {
            // Trigger the legacy jQuery event.
            jQuery(document).trigger(M.core.event.FILTER_CONTENT_UPDATED, [jQuery(e.detail.nodes)]);

            // Trigger the legacy YUI event.
            Y.fire(M.core.event.FILTER_CONTENT_UPDATED, {nodes: new Y.NodeList(e.detail.nodes)});
        });
    });

    legacyEventsRegistered = true;
}
```

What about..?

- HTML on[event] attributes?
- NodeElement.on[event] JS attributes?

```
html_writer::link('#', 'Click me', [
    'onclick' => 'alert("You did it!");',
]);



document.querySelector('a').onClick = () => {
    alert("You did it!");
};
```

moodle

Don’t!!!

What about the `core/pubsub` module..?

- Technically nothing wrong with it
- But it is yet another way to do the same thing
- And very limited in features
- Not widely used in core
- **Ideally migrate to new model proposed here**
  - `[component]/events` module
  - Notify functions
  - Documented event types
- Need to consider how to advise plugins

Issue numbers and more

- MDL-70990 - Event Dispatcher pattern *integrated to master*
- MDL-71113 - Build functioning JSDoc
- MDL-69918 - Rewrite Form Change Checker
- MDL-70830 - Rewrite Short Forms
- MDL-71867 - Update custom_event_interaction
- MDL-71868 - Update pubsub