Usage

Initialization

Create an OriDomi instance by passing your target element to the constructor:

```
var folded = new OriDomi(document.getElementsByClassName('paper')[0]);
```

...or pass a selector string and OriDomi will use the first element that matches:

```
var folded = new OriDomi('.paper');
```

If you prefer using jQuery, try this:

```
var $folded = $('.paper').oriDomi({/* options object */});
// when using jQuery, iterate OriDomi methods over multiple elements like
this:
$folded.oriDomi('accordion', 20);
// to access the OriDomi instance at the top of the jQuery selection
directly:
var folded = $folded.oriDomi(true);
```

Options

When creating a new OriDomi composition, you can pass a map of options as the second argument:

```
var folded = new OriDomi('.paper', {
  vPanels:         5,     // number of panels when folding left or right
(vertically oriented)
  hPanels:         3,     // number of panels when folding top or bottom
  speed:           1200,  // folding duration in ms
  ripple:          2,     // backwards ripple effect when animating
  shadingIntesity: .5,    // lessen the shading effect
  perspective:     800,   // smaller values exaggerate 3D distortion
  maxAngle:        40,    // keep the user's folds within a range of -40 to
40 degrees
  shading:         'soft' // change the shading type
});
```

A full list of options and their descriptions is available **here in the source**.

Effects

Most effect methods only require a folding angle as their first argument.

```
folded.accordion(30);
```

You can specify the anchor to fold from (left, right, top, or bottom) as the second argument:

```
folded.curl(-50, 'top');
```

`foldUp()` is a unique effect method that doesn't take an angle argument because it causes the OriDomi composition to roll up completely in a staggered fashion.

```
// completely hides the element:
folded.foldUp();
```

Its counterpart `unfold()` will be automatically called before another effect method can be applied.

You can browse through all the effect methods **here in the source**.

Callbacks

Maybe you'd like to do something when an animation's complete? Pass a callback function:

```
folded.curl(-50, 'top', function(event, instance) {
  // arguments are the transition event and the OriDomi instance
  alert('It seems my folding days are through.');
});
```

Keep in mind that arguments are flexible. The anchor you used last is assumed when leaving out an anchor argument. OriDomi can usually figure out what you meant:

```
folded.ramp(14, function() {
  alert('A callback as a second argument...');
});
```

Queueing

Callbacks are useful, but can become cumbersome when creating a sequence of animations:

```
// a pyramid you can choose to avoid:
folded.curl(50, function() {
  folded.collapse(function() {
    folded.setSpeed(2000);
    folded.stairs(-29, function() {
      folded.foldUp(function() {
        folded.unfold();
      });
    });
  });
});
```

OriDomi features a built-in queueing system that can be used in harmony with its fluent, chainable interface to easily create sequences:

```
// same result as the previous example:
folded.curl(50).collapse().setSpeed(2000).stairs(-29).foldUp().unfold();
```

In other words, you can call asynchronous methods synchronously and the operations will intelligently queue themselves. There's even no need to chain or make the calls within the same event loop.

You can programmatically empty the queue by calling `emptyQueue()`. Touch and mouse events also immediately clear queued actions.

You can also call `wait()` with a value of milliseconds to queue up a delay between actions:

```
folded.reveal(20).wait(3000).fracture(-30);
```

Touch

The ability to manipulate a composition with the mouse or by touch is enabled by default. To create a non-interactive OriDomi composition, pass this in the initialization options:

```
var handsOff = new OriDomi('.sandpaper', { touchEnabled: false });
```

You can also change it on the fly:

```
handsOff.enableTouch();

handsOff.disableTouch();
```

If you're interested in tracking the user's actions without dealing with event handlers yourself, you can specify callback functions in the initialization options:

```
var slider = new Oridomi('.slider', {
  touchStartCallback: function(startCoordinate, event) {},
  touchMoveCallback:  function(movementAngle, event) {},
  touchEndCallback:   function(endCoordinate, event) {}
});
```

Content manipulation

OriDomi does plenty of DOM manipulation behind the scenes to create what you see. If you want to change the content or the styling of the element after it's been modified, OriDomi provides a setter method to help you out:

```
folded.modifyContent(function(el) {
  el.querySelector('h1').innerHTML = 'ch-ch-ch-ch-changes... turn and face
the strange'
  el.style.backgroundColor = '#000';
});
```

By passing a function to `modifyContent()`, you can easily apply your manipulations on every panel in the composition, with the first argument referring to an individual panel element. `modifyContent()` also passes to the supplied function the anchor (top, left, right, or bottom) and the panel's index within that anchor (as the second and third arguments respectively) should you want to make precise and unique manipulations.

If you're feeling lazy, you can pass a map of selectors and manipulation instructions instead:

```
folded.modifyContent({
  h1: {
    content: 'Hello there',
    style: {
      color: 'green',
      textDecoration: 'underline'
    }
  },
  'div > p': 'just some text.',
  img: {
    style: {
      width: '99%'
    }
  }
});
```

Ripple

By default, every crease in an OriDomi composition will fold simultaneously. For a staggered effect, you can enable “ripple” mode by either passing `ripple: true` in the initial options or by calling `setRipple()`:

```
// staggered, rippling animations:
folded.setRipple().accordion(28).stairs(-40);

// disable ripple:
folded.setRipple(0);

// ripple forwards (default):
folded.setRipple(1);

// ripple backwards:
folded.setRipple(2);
```

The demos at the top of this page have it enabled.

Responsive

OriDomi compositions adapt to dynamic dimensions just as well as normal elements. This means that if your OriDomi composition’s size is changed by percentage-based sizing, media queries, CSS transitions, scripting, et cetera, the panels will resize themselves relative to their parent container.

If you resize this site to a small width, you’ll see the demos above change width accordingly.

Custom panel sizing

If you’d like uneven panel sizing, you can pass an array of percentages in the options instead of a number:

```
var simple = new OriDomi('.simple', { vPanels: [10, 10, 10, 70] });

var fibonacci = new OriDomi('.fibonacci',
  {
    vPanels: [1, 1, 2, 3, 5, 8, 13, 21, 34].map(function(n) {
      return n * 1.1363636363636365;
    })
  }
);
```

The only requirement is that the percentages sum to 100 or near it (for example, `[33, 33, 33]` is valid).

Custom behavior

And now you’ve decided the built-in effects just aren’t good enough for you?

Luckily, you can precisely control the folding behavior of every panel by passing a custom function to the `map()` method:

```
folded.map(function(angle, index, length){ return angle * index *
Math.random() })(20);
```

In the contrived example above, an anonymous function is passed to `map()` that simply multiplies the supplied angle by a given panel's index and a random float. The function is called for every panel in the series and may return a different value for each panel based on the same input value.

Since the function you supply is called with the panel index and the set length, you can create some complex behavior based on evens/odds, angle ranges, or special behavior based on first/last position. For example, if you wanted to create a fold-in effect that keeps the first and last panels flat against the page surface, your function could return different values based on the index and length arguments supplied to it.

Minutiæ

OriDomi requires a modern browser with support for CSS3 transforms, particularly `preserve-3d` support. IE 10 and below lack this, but I'm sure you could've guessed that.

You can test for browser support by checking `OriDomi.isSupported` at runtime. Initializing an OriDomi instance on an unsupported browser will return nothing so it's best to use a conditional around your code dealing with OriDomi.

If you have an improvement for OriDomi, by all means **fork and contribute it**.

Coda

I usually tweet about updates and other projects I'm working on.**Follow me** if that sounds like it might interest you.

For similar open-source projects, take a look at my **personal site**.

And remember, *the DOM is your oyster*.