Layers

v2.0.2

Layers is a Skel plugin that gives it the ability to create and manipulate structures known as **layers**, which are powerful, general purpose regions of content that dock to the edges of the viewport. Layers can be used to create off-canvas navigation, toolbars, and other handy components in a way that's flexible, cross-platform, and free of non-semantic markup bloat.

Note: Layers requires Skel v2.1.0+ and jQuery v1.9+

Getting Started

First, load `skel-layers.min.js` after you've loaded both jQuery and Skel (but **before** you've initialized Skel):

```
<!DOCTYPE HTML>
<html>
    <head>
      <title>Untitled</title>
      <script src="jquery.min.js"></script>
      <script src="skel.min.js"></script>
      <script src="skel-layers.min.js"></script>
      <script>
        skel.init({
          ...
        });
      </script>
    </head>
    <body>
      <div class="container">
        <h1>Hello World!</h1>
        <p>This is a test.</p>
      </div>
    </body>
```

```
</html>
```

You can then configure Layers using the `plugins.layers` option in your Skel configuration, which is simply a key/value list of all of the layers you want to create (key being the layer's unique ID, value being its configuration). A special `config` key can also be used to set certain plugin-wide options.

```
skel.init({
  ...
  plugins: {
    layers: {
      config: {
        option: value,
        option: value,
        option: value,
        ...
      },
      layerID: { /* Layer Configuration */ },
      layerID: { /* Layer Configuration */ },
      layerID: { /* Layer Configuration */ },
      ...
    }
  }
  ...
});
```

As of v2.0.0, you can also create layers using the create API method:

```
$(function() {

  skel.layers.plugin.create('layerID', {
    /* Layer Configuration */
  });

  skel.layers.plugin.create('layerID', {
    /* Layer Configuration */
  });

  skel.layers.plugin.create('layerID', {
    /* Layer Configuration */
  });

  ...

});
```

Each layer is made up of the following:

- A unique **ID**, which simply identifies the layer (eg. `navPanel`).
- A **configuration**, which determines where the layer should be docked, whether it starts out hidden or visible, how it behaves, and what goes in it. There are many options available, with the following being required for all layers:

  `position`

  The position of the layer within the viewport. Can be any of the following: `top-left`, `top`, `top-right`, `right`, `bottom-right`, `bottom`, `bottom-left`, `left`, or `center`.

  `width`

  The width of the layer. Can be in pixels (`150`, `'150px'`), in `em` units (`'10em'`), a percentage (`'25%'`), or `auto`.

  `height`

  The height of the layer. Can be in pixels (`150`, `'150px'`), in `em` units (`'10em'`), a percentage (`'25%'`), or `auto`.

  `html`

  The contents of the layer in HTML. Can be augmented with layer actions to do some really cool stuff.

- **Styling** for the layer, which can be applied directly to the layer's ID (eg. `#navPanel { /* Styling for navPanel layer */ }`).

Examples

A Basic Example: Title Bar

At its simplest, a layer can simply dock to the viewport and just ... sit there, which is actually handy for stuff like title bars and toolbars. For example, to create a simple **title bar** at the top of the viewport:

Configuration

```
skel.init({
  ...
  plugins: {
    layers: {
      titleBar: {
        position: 'top-left',
        width: '100%',
        height: 44,
        html: '<span class="title">Untitled</span>'
      }
    }
  }
  ...
});
```

CSS

```
#titleBar {
  background: #444;
  line-height: 44px;
  text-align: center;
}

#titleBar .title {
  color: #fff;
  font-weight: bold;
}
```

A Useful Example: Off-Canvas Navigation

A common responsive design pattern is **off-canvas navigation**, which is exactly what it sounds like: your navigation, placed off canvas, and called up by a user interaction when it's needed. This can be accomplished with a hidden layer for the navigation (docked to the left), and a button inside the title bar from the previous example to toggle it on/off (using the `toggleLayer` action):

Configuration

```
skel.init({
  ...
  plugins: {
    layers: {
      titleBar: {
```

```
        position: 'top-left',
        width: '100%',
        height: 44,
        html: '<div class="toggle" data-action="toggleLayer" data-
args="navPanel">&equiv;</div>' +
        '<span class="title">Untitled</span>'
      },
      navPanel: {
        position: 'top-left',
        width: 300,
        height: '100%',
        orientation: 'vertical',
        side: 'left',
        hidden: true,
        animation: 'pushX',
        clickToHide: true,
        html: '<a href="#">Home</a>' +
        '<a href="#">Features</a>' +
        '<a href="#">Sign Up</a>' +
        '<a href="#">About</a>' +
        '<a href="#">Contact</a>'
      }
    }
  }
  ...
});
```

CSS

```
#titleBar {
  background: #444;
  line-height: 44px;
  text-align: center;
}

#titleBar .title {
  color: #fff;
  font-weight: bold;
}

#titleBar .toggle {
  position: absolute;
  top: 0;
  left: 0;
  width: 70px;
  height: 44px;
  background: #49bf9d;
  color: #fff;
  font-weight: bold;
}

#navPanel {
```

```
  background: #444;
  line-height: 3em;
  padding: 1.5em;
}

#navPanel a {
  border-top: solid 2px #555;
  color: #fff;
  display: block;
  font-weight: bold;
  text-decoration: none;
}

#navPanel a:first-child {
  border-top: 0;
}
```

An Even More Useful Example: Responsive Off-Canvas Navigation

If you're using the Breakpoint Manager, layers can be set to enable only when specific breakpoints are active. This allows you to limit the use of certain layers to just the situations where they make sense (for example, only enabling off-canvas navigation on mobile devices). For instance, here's the previous example with both the title bar and off-canvas navigation layers linked to just the `small` breakpoint:

Configuration

```
skel.init({
  ...
  breakpoints: {
    large: {
      media: '(min-width: 1025px) and (max-width: 1280px)',
      containers: 960
    },
    medium: {
      media: '(min-width: 769px) and (max-width: 1024px)',
      containers: '90%'
    },
    small: {
      media: '(max-width: 768px)',
      containers: '95%',
      grid: {
        collapse: true
      }
    },
    xsmall: {
      media: '(max-width: 480px)'
    }
```

```
  },
  ...
  plugins: {
    layers: {
      titleBar: {
        breakpoints: ['small'],
        position: 'top-left',
        width: '100%',
        height: 44,
        html: '<div class="toggle" data-action="toggleLayer" data-
args="navPanel">&equiv;</div>' +
        '<span class="title">Untitled</span>'
      },
      navPanel: {
        breakpoints: ['small'],
        position: 'top-left',
        width: 300,
        height: '100%',
        orientation: 'vertical',
        side: 'left',
        hidden: true,
        animation: 'pushX',
        clickToHide: true,
        html: '<a href="#">Home</a>' +
        '<a href="#">Features</a>' +
        '<a href="#">Sign Up</a>' +
        '<a href="#">About</a>' +
        '<a href="#">Contact</a>'
      }
    }
  }
  ...
});
```

Actions

An **action** grants an element *within* a layer's HTML a Layers-specific behavior, which can be anything from simply toggling the visibility of a layer when clicked to automatically "borrowing" the contents of another element when the containing layer is enabled. Actions are applied using the `data-action` attribute and (when needed) the `data-args` attribute:

```
<element data-action="actionName" data-args="arg1,arg2,...,argN" />
```

For example:

```
<div class="toggle" data-action="toggleLayer" data-args="navPanel">Menu</div>
<div class="title" data-action="copyText" data-args="logo"></div>
```

You can also assign actions to an element *outside* a layer's HTML by giving it the `skel-layers-include` class, for example:

```
<p>Click <span class="skel-layers-include" data-action="toggleLayer" data-
args="navPanel">here</span> to toggle the menu.</p>
```

A list of all actions can be found in the Actions Reference.

API

Layers exposes the following methods via the `skel.plugins.layers` object:

`create(layerId, layerConfiguration)`

Creates a new layer using `layerId` as its unique ID and `layerConfiguration` as its configuration. Returns the layer's `Layer` object.

`destroy(layerId)`

Destroys the layer identified by `layerId`.

`get(layerId)`

Gets the `Layer` object for the layer identified by `layerId`.

`hide(layerId)`

Hides the layer identified by `layerId`.

show(*layerId*)

Shows the layer identified by `layerId`.

toggle(*layerId*)

Toggles the visibility of the layer identified by `layerId`.

Events

As of v2.0.0, handlers can now be attached to certain **layer events** (for example, when a layer is shown or hidden). To do this, first get the layer's `Layer` object, then use its `on()` method to attach a handler. For example:

```
var layer = skel.plugins.layers.get('navPanel');

layer.on('show', function() {
  alert('Showing navPanel!');
});

layer.on('hide', function() {
  alert('Hiding navPanel!');
});
```

The following events are currently supported:

show

Triggered when the layer is shown.

hide

Triggered when the layer is hidden.

showstart

Triggered when the layer's "show" animation begins.

showend

Triggered when the layer's "show" animation ends.

hidestart

Triggered when the layer's "hide" animation begins.

hideend

Triggered when the layer's "hide" animation ends.

Actions Reference

toggleLayer

Args	`layerId`
When	User Click

Toggles the visibility of a layer.

copyText

Args	`elementId`
When	Containing layer is enabled

Copies the text contents of the target element to this one.

copyHTML

Args	`elementId`
When	Containing layer is enabled

Copies the HTML contents of the target element to this one.

`moveElementContents`

Args	`elementId`
When	Containing layer is enabled

Moves the the children of the target element to this one. When the containing layer is disabled, the children are moved back to the target element.

`moveElement`

Args	`elementId`
When	Containing layer is enabled

Moves the target element to this one. When the containing layer is disabled, the target element is moved back to its original location.

`moveCell`

Args	`cellId`
When	Containing layer is enabled

Moves the target grid system cell to this element, proportionally redistributing the row space it occupied to its sibling cells. When the containing layer is disabled, the target cell is restored to its original location and width.

Configuration Reference

Note: These go under the `plugins.layers.config` option.

`baseZIndex`

Type	integer
Default	`10000`

Sets the base z-index for all layers. Should be well above anything else on the page.

mode

Type	string, function
Default	`position`

Animation mode. Can be any of the following:

transform

Animates layers using CSS transforms. Results in super fluid animations on all modern desktop (Firefox, Chrome, Safari, IE10+) and mobile (iOS and Android) browsers. Automatically switches to `position` if CSS transforms aren't supported.

*Note: CSS transforms have a habit of breaking `position: fixed` elements (apparently by design – details here). A workaround that works in **most** situations is to simply apply the `skel-layers-fixed` class to each of your fixed elements (also works on dynamically generated elements; just call `skel.plugins.layers.refresh()` after applying the class). For more complex situations, however, using a different mode might be the better choice.*

position

Animates layers using standard CSS positoning (`left`, `top`, etc.) and CSS transitions. Not as fluid as `transform`, but a good alternative in situations where CSS transforms aren't feasible. Automatically switches to `animate` if CSS transitions aren't supported.

animate

Animates layers using jQuery's `animate` function. Slower than `transform` and slightly slower than `position`, but works pretty much everywhere (including IE8).

function() { ... }

A callback function that returns the mode to use. For example, this results in `transform` on mobile devices, and `position` everywhere else:

```
mode: function() {

  if (skel.vars.isMobile)
    return "transform";
```

```
    return "position";

},
```

speed

Type	integer
Default	`250`

Animation speed (in milliseconds).

wrap

Type	bool
Default	`true`

If `true`, Layers will set up its wrapper elements as soon as Skel is initialized. If `false`, this will be delayed until a layer is actually created.

Layer Configuration

hidden

Type	bool
Default	`false`

If `true`, this layer will start out hidden. A layer action or API call will be required to show it.

position

Type	string

Sets the position of this layer. Can be any of the following:

- `top-left`
- `top`
- `top-right`

- `right`
- `bottom-right`
- `bottom`
- `bottom-left`
- `left`
- `center`

side

Type	string
Default	*(undefined)*

Sets this layer's preferred side of the viewport (impacts `animation`; see below). Can be any of the following:

- `top`
- `right`
- `bottom`
- `left`

width

Type	mixed
Default	`"auto"`

Sets the width of this layer. Can be in pixels (`960`, `"960px"`), in `em` units (`8em`), a percentage (`"75%"`), or `auto`.

height

Type	mixed
Default	`"auto"`

Sets the height of this layer. Can be in pixels (`960`, `"960px"`), in `em` units (`8em`), a percentage (`"75%"`), or `auto`.

maxWidth

Type	mixed

Default	`"100%"`

Sets the **maximum** width of this layer. Can be in pixels (`960`, `"960px"`), in `em` units (`8em`), a percentage (`"75%"`), or `auto`.

`maxHeight`

Type	mixed
Default	`"100%"`

Sets the **maximum** height of this layer. Can be in pixels (`960`, `"960px"`), in `em` units (`8em`), a percentage (`"75%"`), or `auto`.

`orientation`

Type	string
Default	`"none"`

Sets the orientation of this layer, which impacts how it should scroll when its content exceeds its boundaries. Can be any of the following:

`horizontal`

Horizontal orientation (left/right scrolling).

`vertical`

Vertical orientation (up/down scrolling).

`html`

Type	string

Defines this layer's contents in HTML.

Note: If omitted, this layer will look for an existing element that shares its ID. If found, it'll "absorb" that element's content before destroying it.

`animation`

Type	string
Default	`"none"`

Sets the style of animation used when this layer is shown or hidden. Can be any of the following:

`none`

No animation.

`fade`

Fades this layer in/out.

`overlayX`

Slides this layer over the viewport (when `side` is `left` or `right`).

`overlayY`

Slides this layer over the viewport (when `side` is `top` or `bottom`).

`pushX`

Slides this layer and viewport over (when `side` is `left` or `right`)

`pushY`

Slides this layer and viewport over (when `side` is `top` or `bottom`)

`revealX`

Slides the viewport over to reveal this layer (when `side` is `left` or `right`)

`clickToHide`

Type	bool
Default	`false`

If `true`, automatically hides this layer when any links inside it are clicked.

`swipeToHide`

Type	bool
Default	`true`

If `true`, allows a user to swipe a layer (in the direction of its `side` option) to hide it. Only applies to layers where `side` is either `left` or `right`.

`resetForms`

Type	bool
Default	`true`

If `true`, resets any forms within this layer whenever it's hidden.

`resetScroll`

Type	bool
Default	`true`

If `true`, resets the scroll position of this layer whenever it's hidden.

`breakpoints`

Type	array
Default	*(undefined)*

If defined, ties this layer to one or more breakpoints (eg. `['small','xsmall']`). This layer will only be enabled when at least one of these breakpoints is active, and will be disabled if not.

`states`

Type	array
Default	*(undefined)*

If defined, ties this layer to one or more states identified by their state IDs (eg. `['/small','/small/xsmall']`). This layer will only be enabled when the current state ID matches one of these, and will be disabled if not.

Credits

- jQuery (jquery.com | (c) 2005, 2014 jQuery Foundation, Inc. and other contributors | MIT license)
- UMD Wrapper (github.com/umdjs/umd | @umdjs)

License