Boundless Web SDK

Manual

Published
with GitBook

Table of Contents

Introduction

The Boundless Web SDK is based on OpenLayers 3 and React 0.14.

It provides out of the box components for making it easy to build modern web mapping applications.

Getting Started

JSFiddle

Check out the following JSFiddle example.

Node version manager (nvm)

Install the node version manager from https://github.com/creationix/nvm For the Web SDK, you should be using node 4.3.2 so:

```
nvm install 4.3.2
nvm use 4.3.2
```

Using the web-sdk application generator

The easiest way to get started is to use the web-sdk application generator. We will also outline the manual steps at the end of this section for advanced users, but the preferred way is to use the application generator.

Installing the generator

Run the following command:

```
npm install -g web-sdk-generator --registry https://repo.boundlessgeo.com/api/npm/
npmall
```

To install the package globally might need `sudo` rights.

You can then run the web-sdk command like this:

```
web-sdk --help
```

If you have insufficient rights to install globally, install the package like this:

```
npm install web-sdk-generator --registry https://repo.boundlessgeo.com/api/npm/npm
all
```

And run:

```
node_modules/.bin/web-sdk --help
```

This will output the usage instructions:

```
$ web-sdk --help

  Usage: web-sdk [options] [dir]

  Options:

    -h, --help     output usage information
    -V, --version  output the version number
    -f, --force    force on non-empty directory
```

Creating the application

Let's create our first application with the following command (when using a globally installed `web-sdk` you can simply use `web-sdk /tmp/myapp`):

```
$ node_modules/.bin/web-sdk /tmp/myapp

   create : /tmp/myapp
   create : /tmp/myapp/app.jsx
   create : /tmp/myapp/app.css
   create : /tmp/myapp/createbuild.js
   create : /tmp/myapp/debug-server.js
   create : /tmp/myapp/index.html
   create : /tmp/myapp/package.json

   install dependencies:
     $ cd /tmp/myapp && npm install --registry https://repo.boundlessgeo.com/api/n
pm/npmall

   run the app:
     $ npm start
```

The application is generated and instructions are outputted on the next steps to undertake. Change to this directory and run:

```
npm install --registry https://repo.boundlessgeo.com/api/npm/npmall
```

This will install all the needed dependencies for the application.

We can now run the debug server with the following command:

```
npm start
```

This will start up a debug server on port 3000:

```
$ npm start

> myapp@0.0.0 start /private/tmp/myapp
> npm-run-all --parallel createdir start:debug


> myapp@0.0.0 createdir /private/tmp/myapp
> node createbuild.js


> myapp@0.0.0 start:debug /private/tmp/myapp
> node debug-server.js

info serve-lib Parsing dependencies.
info serve-lib Debug server running http://localhost:3000/loader.js (Ctrl+C to sto
p)
```

To create a zip file package for production use the following command:

```
$ npm run package
```

You will be prompted for a destination file path and file name for the zip file, for instance `/tmp/myapp.zip`.

Using Boundless Web SDK from npm

The following packages are relevant for the Boundless Web SDK:

- `boundless-sdk`, the main package containing the components
- `sdk-tools`, package containing debug server
- `web-sdk-generator`, package containing CLI for creating apps

The packages are stored in our repo server at:
https://repo.boundlessgeo.com/api/npm/npmall

So to install them you will need to use:

```
npm install boundless-sdk --registry https://repo.boundlessgeo.com/api/npm/npmall
npm install sdk-tools --registry https://repo.boundlessgeo.com/api/npm/npmall
npm install web-sdk-generator --registry https://repo.boundlessgeo.com/api/npm/npm
all
```

Tutorial

1. Using the generator

In the getting started section we have learned how to create a skeleton application with the `web-sdk` command.

The skeleton application contains two base map layers, MapQuest streets and aerial, and a single widget, the `LayerList` widget.

The main file of the application is `app.jsx`. Open up this file in your favorite text editor. Look for the definition of `ol.Map`. The map gets defined with a layer group, that combines the MapQuest streets and aerial layers. The view is defined with an initial center and zoom level. If you are not familiar with OpenLayers 3, the recommendation is to use the workshop at http://openlayers.org/workshop/ to get up to speed with OpenLayers 3.

2. Adding a vector layer from a GeoJSON file

Create a subdirectory called `data` in the root of the application directory, download USA states data as GeoJSON from: http://data.okfn.org/data/datasets/geo-boundaries-us-110m and save it to the `data` subdirectory.

```
mkdir data
curl "https://raw.githubusercontent.com/datasets/geo-boundaries-us-110m/master/jso
n/ne_110m_admin_1_states_provinces_shp_scale_rank.geojson" -o data/ne_110m_admin_1
_states_provinces_shp_scale_rank.geojson
```

Add the following layer definition in `app.jsx` after the existing `ol.layer.Group`:

```
  new ol.layer.Vector({
    id: 'states',
    title: 'USA States',
    source: new ol.source.Vector({
      url: 'data/ne_110m_admin_1_states_provinces_shp_scale_rank.geojson',
      format: new ol.format.GeoJSON()
    })
  })
```

And change the view's center so that the GeoJSON layer will be visible on start up:

```
  view: new ol.View({
    center: [-10605790.55, 4363637.07],
    zoom: 4
  })
```

The full map definition should now look like:

```
var map = new ol.Map({
  layers: [
    new ol.layer.Group({
      type: 'base-group',
      title: 'Base maps',
      layers: [
        new ol.layer.Tile({
          type: 'base',
          title: 'Streets',
          source: new ol.source.MapQuest({layer: 'osm'})
        }),
        new ol.layer.Tile({
          type: 'base',
          visible: false,
          title: 'Aerial',
          source: new ol.source.MapQuest({layer: 'sat'})
        })
      ]
    }),
    new ol.layer.Vector({
      id: 'states',
      title: 'USA States',
      source: new ol.source.Vector({
        url: 'data/ne_110m_admin_1_states_provinces_shp_scale_rank.geojson',
        format: new ol.format.GeoJSON()
      })
    })
  ],
  view: new ol.View({
    center: [-10605790.55, 4363637.07],
    zoom: 4
  })
});
```

3. Adding a feature grid

In this step we'll learn how to add another component to the application, besides the LayerList component which is already in the application. First, we'll add an import statement at the top of `app.jsx`:

```
import FeatureTable from './node_modules/boundless-sdk/js/components/FeatureTable.
jsx';
```

In the render function of our application, we need to add the definition of our new component, FeatureTable:

The feature table needs to get configured with at least a layer and a map:

```
  <div ref='map' id='map'></div>
  <div><LayerList map={map} /></div>
  <div id='table'><FeatureTable map={map} layer={map.getLayers().item(1)} /></div>
```

Our div with id `table` does not yet have a size, so open up `app.css` and give it some space on the page:

```
#map {
  width: 100%;
  height: 60%;
}
#table {
  width: 100%;
  height: 40%;
}
```

We have reduced the map div to 60% height and allocated the other 40% for the table.

If you reload the debug server at http://localhost:3000/ you'll see that we now have a feature table at the bottom of the page. The width of the feature table however is not optimal. Let's define that we want our feature table to resize to the table div, for this we use the `resizeTo` property to point to the id of the table div:

```
<FeatureTable resizeTo='table' map={map} layer={map.getLayers().item(1)} />
```

When you click a row in the feature table, it will select the corresponding geometry in the map.

Our app.jsx will now look like:

```
import React from 'react';
import ReactDOM from 'react-dom';
import ol from 'openlayers';
import {addLocaleData, IntlProvider} from 'react-intl';
import App from './node_modules/boundless-sdk/js/components/app.js';
import LayerList from './node_modules/boundless-sdk/js/components/LayerList.jsx';
import FeatureTable from './node_modules/boundless-sdk/js/components/FeatureTable.
jsx';
import enLocaleData from './node_modules/react-intl/locale-data/en.js';
import enMessages from './node_modules/boundless-sdk/locale/en.js';

addLocaleData(
  enLocaleData
);

var map = new ol.Map({
  layers: [
    new ol.layer.Group({
      type: 'base-group',
      title: 'Base maps',
      layers: [
        new ol.layer.Tile({
          type: 'base',
          title: 'Streets',
          source: new ol.source.MapQuest({layer: 'osm'})
        }),
        new ol.layer.Tile({
          type: 'base',
          visible: false,
          title: 'Aerial',
          source: new ol.source.MapQuest({layer: 'sat'})
        })
      ]
    }),
    new ol.layer.Vector({
      id: 'states',
      title: 'USA States',
      source: new ol.source.Vector({
        url: 'data/ne_110m_admin_1_states_provinces_shp_scale_rank.geojson',
        format: new ol.format.GeoJSON()
      })
    })
  ],
  view: new ol.View({
    center: [-10605790.55, 4363637.07],
    zoom: 4
  })
});

class MyApp extends App {
  render() {
```

```
    return (
      <article>
        <div ref='map' id='map'></div>
        <div><LayerList map={map} /></div>
        <div id='table'><FeatureTable resizeTo='table' map={map} layer={map.getLay
ers().item(1)} /></div>
      </article>
    );
  }
}

ReactDOM.render(<IntlProvider locale='en' messages={enMessages}><MyApp map={map} /
></IntlProvider>, document.getElementById('main'));
```

Our app.css will look like:

```
html, body, article {
  width: 100%;
  height: 100%;
}
#main {
  width: 100%;
  height: 100%;
}
#map {
  width: 100%;
  height: 60%;
}
#table {
  width: 100%;
  height: 40%;
}
```

4. Adding an upload component

In this step we'll be adding a button to the application that will open up a dialog where the user can upload a vector file, such as a KML, GPX or GeoJSON file.

Again, we will start by adding an `import` statement to import our component:

```
import AddLayer from './node_modules/boundless-sdk/js/components/AddLayer.jsx';
```

In the `render` function of our application, we need to add a toolbar to accommodate for the button of the upload component:

```
  <nav role='navigation'>
    <div className='toolbar'>
      <ul><AddLayer map={map} /></ul>
    </div>
  </nav>
```

Add the `nav` just after the `<article>` opening tag.

Also add a div to group the map and featuretable divs, with id `content`:

```
  <div id='content'>
    <div ref='map' id='map'></div>
    <div><LayerList map={map} /></div>
    <div id='table'><FeatureTable resizeTo='table' map={map} layer={map.getLayers(
).item(1)} /></div>
  </div>
```

Open up app.css to give a bit of space to the button in the toolbar with `padding-top`, also give the toolbar a fixed height and adjust the `content` div's height accordingly:

```
.toolbar {
  height: 50px;
}
#content {
  width: 100%;
  height:  calc(100% - 50px);
}
.toolbar ul {
  padding-top: 4px;
}
```

The final app.jsx will look like:

```
import React from 'react';
import ReactDOM from 'react-dom';
import ol from 'openlayers';
import {addLocaleData, IntlProvider} from 'react-intl';
import App from './node_modules/boundless-sdk/js/components/app.js';
import LayerList from './node_modules/boundless-sdk/js/components/LayerList.jsx';
import FeatureTable from './node_modules/boundless-sdk/js/components/FeatureTable.
jsx';
import AddLayer from './node_modules/boundless-sdk/js/components/AddLayer.jsx';
import enLocaleData from './node_modules/react-intl/locale-data/en.js';
import enMessages from './node_modules/boundless-sdk/locale/en.js';

addLocaleData(
```

```
  enLocaleData
);

var map = new ol.Map({
  layers: [
    new ol.layer.Group({
      type: 'base-group',
      title: 'Base maps',
      layers: [
        new ol.layer.Tile({
          type: 'base',
          title: 'Streets',
          source: new ol.source.MapQuest({layer: 'osm'})
        }),
        new ol.layer.Tile({
          type: 'base',
          visible: false,
          title: 'Aerial',
          source: new ol.source.MapQuest({layer: 'sat'})
        })
      ]
    }),
    new ol.layer.Vector({
      id: 'states',
      title: 'USA States',
      source: new ol.source.Vector({
        url: 'data/ne_110m_admin_1_states_provinces_shp_scale_rank.geojson',
        format: new ol.format.GeoJSON()
      })
    })
  ],
  view: new ol.View({
    center: [-10605790.55, 4363637.07],
    zoom: 4
  })
});

class MyApp extends App {
  render() {
    return (
      <article>
        <nav role='navigation'>
          <div className='toolbar'>
            <ul><AddLayer map={map} /></ul>
          </div>
        </nav>
        <div id='content'>
          <div ref='map' id='map'></div>
          <div><LayerList map={map} /></div>
          <div id='table'><FeatureTable resizeTo='table' map={map} layer={map.getL
ayers().item(1)} /></div>
        </div>
```

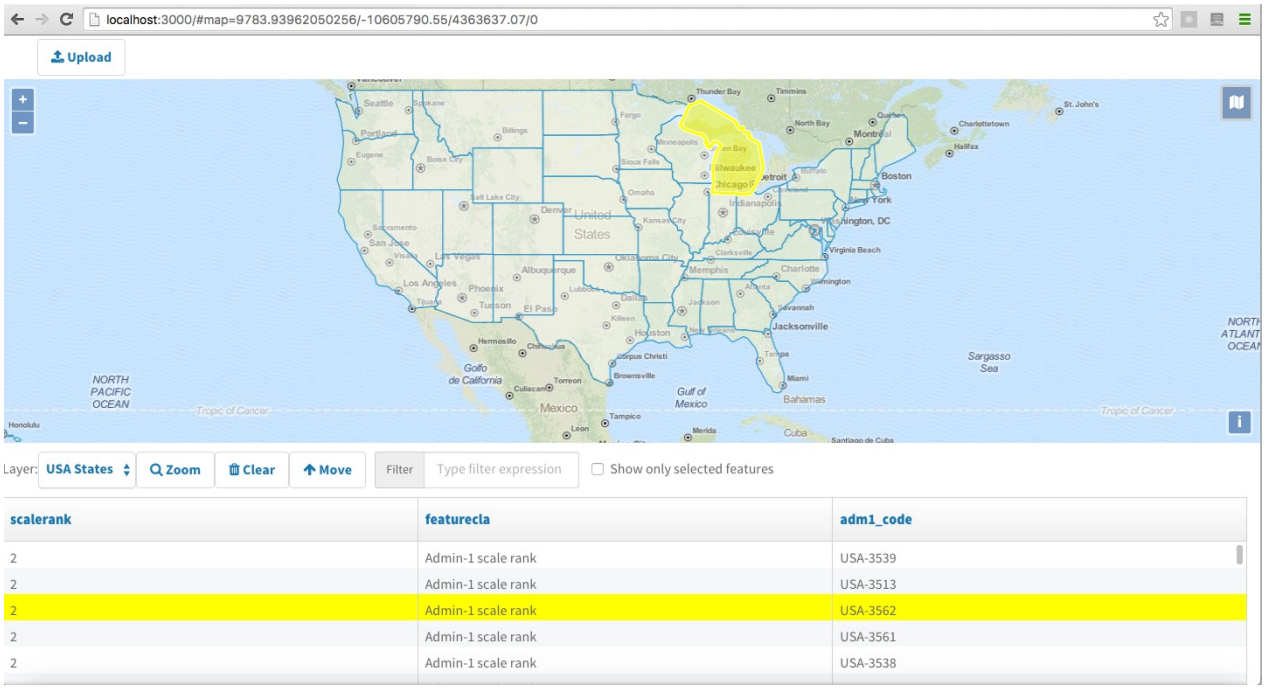
```
      </article>
    );
  }
}

ReactDOM.render(<IntlProvider locale='en' messages={enMessages}><MyApp map={map} />
></IntlProvider>, document.getElementById('main'));
```

The final app.css will look like:

```
html, body, article {
  width: 100%;
  height: 100%;
}
#main {
  width: 100%;
  height: 100%;
}
#map {
  width: 100%;
  height: 60%;
}
#table {
  width: 100%;
  height: 40%;
}
.toolbar {
  height: 50px;
}
#content {
  width: 100%;
  height:  calc(100% - 50px);
}
.toolbar ul {
  padding-top: 4px;
}
```

The final application will look like:



.

Information for component developers

Introduction

The Web SDK is based on EcmaScript 6 (ES6). It uses babelify to translate to ES5. It is based on OpenLayers 3, React 0.14 and Flux. For the user interface, currently the Pivotal UI (pui) React components are used. See http://styleguide.pivotal.io for more information. For internationalization, Yahoo's react-intl (v2) is used. Node and npm are used as the development environment.

React

Component API (e.g. `componentWillMount`, `componentWillUnmount`) is important to know about. Components have properties (`this.props`), more like configuration options. Components have state (`this.state`), and on change of state (`setState`) they are re-rendered. React uses a virtual DOM, so it can diff the changes and only apply those (performance). Render method needs to return single root (`<article>` can be used as workaround). To pass on all properties to a child component use: `{…this.props}`. Define `propTypes` static on the class, for example: `Chart.propTypes = { charts: React.PropTypes.array.isRequired };`.

JSX

JSX is an XML like syntax to create components. For example:

```
<Component property1={variable1} />
```

It can be nested, and can be mixed with normal HTML. You can use the ref property instead of the HTML id (`ref='map'` in JSX and then `this.refs.map` in code). Babelify will transform this to regular javascript.

Flux

Flux means Uni-directional flow of data. A dispatched change needs to be handled, before another one can commence. It has concepts such as stores, actions, dispatchers. This is internal to the SDK, the app developer should not need to know too much about it. The store extends the standard EventEmitter class. Stores should be singletons. We use a central application dispatcher that extends Flux's Dispatcher. Actions are functions that ask the dispatcher to handle certain actions (using constants). Components (or stores) can register listener from actions on the dispatcher. An example:

```
AppDispatcher.register((payload) => {
  let action = payload.action;
  switch(action.type) {
    case SelectConstants.SELECT_FEATURES_IN:
      // do something
      break;
    default:
      break;
  }
});
```

Internationalisation (i18n)

At the component level:

```
import {defineMessages, injectIntl, intlShape} from ‘react-intl';

const messages = defineMessages({
  buttontitle: {
    id: 'homebutton.buttontitle',
    description: 'Title for the home button',
    defaultMessage: 'Home'
  }
});

class HomeButton extends React.Component {
  render() {
    const {formatMessage} = this.props.intl;
    return (
      <button title={formatMessage(messages.buttontitle)} />
    );
  }
}

HomeButton.propTypes = {
 intl: intlShape.isRequired
};

export default injectIntl(HomeButton);
```

At the application level:

```
import {addLocaleData, IntlProvider} from ‘react-intl';
import enLocaleData from './node_modules/react-intl/dist/locale-data/en.js';
import enMessages from ‘./node_modules/boundless-sdk/locale/en.js';

addLocaleData(
  enLocaleData
);

ReactDOM.render(<IntlProvider locale='en' messages={enMessages}><App /></IntlProvi
der>, document.getElementById('main'));
```

We use the babel-plugin-react-intl to extract messages from source code. It generates JSON files per component. `npm run i18n` generates i18n template for all components in `locale/en.js` In `.babelrc` this is activated with:

```
  "plugins": ["react-intl"],
  "extra": {
    "react-intl": {
      "messagesDir": "./build/messages/",
      "enforceDescriptions": true
    }
  }
```

API docs

Uses `react-docgen`, which reads a documentation block on top of the class and a documentation block on top of each prop. `npm run generate:docs` will generate the API docs.

CSS

Two approaches are used:

- dr-frankenstyle
- cssify (browserify transform)

dr-frankenstyle

Tool by Pivotal that looks for style keys in package.json of all node dependencies. It will bundle this up with all the resources (fonts, images). `dr-frankenstyle` is used to generates dist/css/* for the SDK.

cssify

A post install task copies css files needed from external components (such as fixed-data-table). By using import statements to css files, browserify bundles up the css (but won't copy any needed resources, so this only works for plain css).

```
import './FeatureTable.css';
```

Debug server

Can be used by any Web SDK application. In `package.json`:

```
  "sdk-debug-server": "git+https://github.com/bartvde/sdk-debug-server.git"
```

To use it in a node script file:

```
require('sdk-debug-server').startServer();
```

Any application generated by the `web-sdk` will facilitate this.

Proxy to local geoserver

Add the following dependency to your `package.json`:

```
"json-http-proxy": "1.0.0-ALPHA-4",
```

To start the proxy, add a script to the `package.json` file:

```
"start:proxy": "node_modules/.bin/json-http-proxy -p 4000",
```

Using `npm run start:proxy` will start up the proxy server on port 4000 using config from `proxy-config.json`:

```
{
  "routes": [
    {
      "path": "/geoserver/*",
      "upstream": "geoserver"
    },
    {
      "path": "/*",
      "upstream": "default"
    }
  ],
  "upstreams": {
    "geoserver": {
      "protocol": "http",
      "hostname": "localhost",
      "port": 8080
    },
    "default": {
      "port": 3000
    }
  }
}
```

This means in another shell you will need to start up the normal debug server on port 3000 using `npm start`.

API

The API describes the components of the Web SDK with their corresponding properties.

`AddLayer` (component)

Adds a menu entry that can be used by the web app user to add a layer to the map. Only vector layers can be added. Supported formats for layers are GeoJSON, GPX and KML.

```
<AddLayer map={map} />
```

Properties

`intl`

i18n message strings. Provided through the application through context.

type: `custom`

`map` (required)

The ol3 map instance to add to.

type: `instanceOf ol.Map`

`pointRadius`

The point radius used for the circle style.

type: `number` defaultValue: `7`

`strokeWidth`

The stroke width in pixels used in the style for the uploaded data.

type: `number` defaultValue: `2`

`App` (component)

Base class for applications. Can handle using the browser history to navigate through map extents. An initial extent can be provided as well.

```
import React from 'react';
import ReactDOM from 'react-dom';
import ol from 'openlayers';
import {IntlProvider} from 'react-intl';
import App from './node_modules/boundless-sdk/js/components/App.js';
import enMessages from './node_modules/boundless-sdk/locale/en.js';

class MyApp extends App {
  componentDidMount() {
    super.componentDidMount();
    // your code here
  }
  render() {
    // we need to provide a reference to the map
    return (
      <div id='map' ref='map'></div>
    );
  }
}

var extent = [1327331, 4525032, 5123499, 5503426];
ReactDOM.render(<IntlProvider locale='en' messages={enMessages}><MyApp extent={ext
ent} useHistory={false} map={map} /></IntlProvider>, document.getElementById('main
'));
```

Properties

`extent`

Extent to fit on the map on loading of the application.

type: `arrayOf number`

`map` (required)

The map to use for this application.

type: `instanceOf ol.Map`

`useHistory`

Use the back and forward buttons of the browser for navigation history.

type: `bool` defaultValue: `true`

`Bookmarks` (component)

Adds the ability to retrieve spatial bookmarks. A spatial bookmark consists of a name, an extent and a description.

```
var bookmarks = [{
  name: 'Le Grenier Pain',
  description: '<b>Address: </b>38 rue des Abbesses<br><b>Telephone:</b> 33 (0)1 4
6 06 41 81<br><a href=""http://www.legrenierapain.com"">Website</a>',
  extent: [259562.7661267497, 6254560.095662868, 260675.9610346824, 6256252.988234
103]
}, {
  name: 'Poilne',
  description: '<b>Address: </b>8 rue du Cherche-Midi<br><b>Telephone:</b> 33 (0)1
 45 48 42 59<br><a href=""http://www.poilane.fr"">Website</a>',
   extent: [258703.71361629796, 6248811.5276565505, 259816.90852423065, 6250503.27
1278702]
}];

class BookmarkApp extends App {
  render() {
    return (
      <div id='content'>
        <div ref='map' id='map'>
          <div id='bookmarks-panel'>
            <Bookmarks introTitle='Paris bakeries' introDescription='Explore the b
est bakeries of the capital of France' map={map} bookmarks={bookmarks} />
          </div>
        </div>
      </div>
    );
  }
}

ReactDOM.render(<IntlProvider locale='en' messages={enMessages}><BookmarkApp map={
map} /></IntlProvider>, document.getElementById('main'));
```

Properties

`animatePanZoom`

Should we animate the pan and zoom operation?

type: `bool` defaultValue: `true`

`animationDuration`

The duration of the animation in milleseconds. Only relevant if animatePanZoom is true.

type: `number` defaultValue: `500`

`autoplay`

Should the scroller auto scroll?

type: `bool` defaultValue: `false`

`autoplaySpeed`

delay between each auto scoll in ms.

type: `number`

`bookmarks` (required)

The bookmark data. An array of objects with name (string, required), description (string, required) and extent (array of number, required) keys. The extent should be in the view projection.

type: `arrayOf shape`

`dots`

Should we show indicators? These are dots to navigate the bookmark pages.

type: `bool` defaultValue: `true`

`intl`

i18n message strings. Provided through the application through context.

type: `custom`

`introDescription`

The description of the introduction (first) page of the bookmarks.

type: `string` defaultValue: `''`

`introTitle`

The title on the introduction (first) page of the bookmarks.

type: `string` defaultValue: `''`

`map` (required)

The ol3 map instance on whose view we should navigate.

type: `instanceOf ol.Map`

`markerUrl`

Url to the marker image to use for bookmark position.

type: `string` defaultValue: `'./resources/marker.png'`

`menu`

Display as a menu drop down list.

type: `bool` defaultValue: `false`

`showMarker`

Should we display a marker for the bookmark? Default is true.

type: `bool` defaultValue: `true`

`Chart` (component)

Allow for the creation of charts based on selected features of a layer.

```
var charts = [{
  title: 'Airports count per use category',
  categoryField: 'USE',
  layer: 'lyr03',
  valueFields: [],
  displayMode: 2,
  operation: 2
}, {
  title: 'Forest area total surface',
  categoryField: 'VEGDESC',
  layer: 'lyr01',
  valueFields: ['AREA_KM2'],
  displayMode: 1,
  operation: 2
}];
```

```
<div  id='chart-panel'><Chart ref='chartPanel' combo={true} charts={charts}/></div>
```

Properties

`charts` (required)

An array of configuration objects. Configuration objects have title, categoryField, layer, valueFields, displayMode and operation keys. title (string, required) is the title to display for the chart. categoryField (string, optional) is the attribute to use as the category. layer (string, required) is the id property of the corresponding layer to use. valueFields (array of string, required) is an array of field names to use for displaying values in the chart. displayMode (enum(0, 1, 2), required) defines how the feature attributes will be used to create the chart. When using a value of 0 (by feature), an element will be added to the chart for each selected feature. When using a value of 1 (by category), selected features will be grouped according to a category defined by the categoryField. When using a value of 2 (count by category) the chart will show the number of features in each

category. The statistic function to use when displayMode is by category (1) is defined in the operation (enum(0, 1, 2, 3), optional) key. A value of 0 means MIN, a value of 1 means MAX, a value of 2 means SUM and a value of 3 means AVG (Average).

type: `arrayOf shape`

`combo`

If true, show a combo box to select charts instead of dropdown button.

type: `bool` defaultValue: `false`

`container`

The id of the container to show when a chart is selected.

type: `string`

`intl`

i18n message strings. Provided through the application through context.

type: `custom`

`Edit` (component)

A component that allows creating new features, so drawing their geometries and setting feature attributes through a form.

```
<div ref='editToolPanel' className='edit-tool-panel'><Edit toggleGroup='navigation'
 map={map} /></div>
```

Properties

`intl`

i18n message strings. Provided through the application through context.

type: `custom`

`pointRadius`

The point radius used for the circle style.

type: `number` defaultValue: `7`

`strokeWidth`

The stroke width in pixels used in the style for the created features.

type: `number` defaultValue: `2`

`EditPopup` (component)

Popup that can be used for feature editing (attribute form).

```
<div id='popup' className='ol-popup'><EditPopup map={map} /></div>
```

Properties

`intl`

i18n message strings. Provided through the application through context.

type: `custom`

`map` (required)

The ol3 map to register for singleClick.

type: `instanceOf ol.Map`

`FeatureTable` (component)

A table to show features. Allows for selection of features.

```
var selectedLayer = map.getLayers().item(2);
```

```
<div ref='tablePanel' id='table-panel' className='attributes-table'>
  <FeatureTable ref='table' resizeTo='table-panel' offset={[30, 30]} layer={select
edLayer} map={map} />
</div>
```

Properties

`columnWidth`

The width in pixels per column.

type: `number` defaultValue: `100`

`headerHeight`

The height of the table header in pixels.

type: `number` defaultValue: `50`

`height`

The height of the table component in pixels.

type: `number` defaultValue: `400`

`intl`

i18n message strings. Provided through the application through context.

type: `custom`

`layer` (required)

The layer to use initially for loading the table.

type: `instanceOf ol.layer.Vector`

`map` (required)

The ol3 map in which the source for the table resides.

type: `instanceOf ol.Map`

`offset`

Array with offsetX and offsetY, the number of pixels to make the table smaller than the resizeTo container.

type: `array` defaultValue: `[0, 0]`

`pointZoom`

The zoom level to zoom the map to in case of a point geometry.

type: `number` defaultValue: `16`

`refreshRate`

Refresh rate in ms that determines how often to resize the feature table when the window is resized.

type: `number` defaultValue: `250`

`resizeTo`

The id of the container to resize the feature table to.

type: `string`

`rowHeight`

The height of a row in pixels.

type: `number` defaultValue: `30`

`width`

The width of the table component in pixels.

type: `number` defaultValue: `400`

`Geocoding` (component)

Input field to search for placenames using a geocoding service (OSM nominatim).

```
<div id='geocoding' className='pull-right'><Geocoding /></div>
```

Properties

`intl`

i18n message strings. Provided through the application through context.

type: `custom`

`maxResults`

The maximum number of results to return on a search.

type: `number` defaultValue: `5`

`GeocodingResults` (component)

This component displays the results of geocoding search. The geocoding search is initiated by the Geocoding component.

```
<div id='geocoding-results' className='geocoding-results'>
  <GeocodingResults map={map} />
</div>
```

Properties

`intl`

i18n message strings. Provided through the application through context.

type: `custom`

`map` (required)

The ol3 map on whose view to perform the center action.

type: `instanceOf ol.Map`

`markerUrl`

Url to the marker image to use for bookmark position.

type: `string` defaultValue: `'./resources/marker.png'`

`zoom`

The zoom level used when centering the view on a geocoding result.

type: `number` defaultValue: `10`

`Geolocation` (component)

Enable geolocation which uses the current position of the user in the map.

```
<div id='geolocation-control' className='ol-unselectable ol-control'>
  <Geolocation map={map} />
</div>
```

Properties

`intl`

i18n message strings. Provided through the application through context.

type: `custom`

`map` (required)

The ol3 map for which to change its view's center.

type: `instanceOf ol.Map`

`Globe` (component)

Adds a button to toggle 3D mode. The HTML page of the application needs to include a script tag to cesium:

```
<script src="./resources/ol3-cesium/Cesium.js" type="text/javascript" charset="utf
-8"></script>
```

```
<div ref='map' id='map'>
  <div id='globe-button' className='ol-unselectable ol-control'>
    <Globe map={map} />
  </div>
</div>
```

Properties

`map` (required)

The ol3 map instance to work on.

type: `instanceOf ol.Map`

`HomeButton` (component)

A button to go back to the initial extent of the map.

```
<div id='home-button' className='ol-unselectable ol-control'>
  <HomeButton map={map} />
</div>
```

Properties

`intl`

i18n message strings. Provided through the application through context.

type: `custom`

`map` (required)

The ol3 map for whose view the initial center and zoom should be restored.

type: `instanceOf ol.Map`

`ImageExport` (component)

Export the map as a PNG file. This will only work if the canvas is not tainted.

```
<ImageExport map={map} />
```

Properties

`intl`

i18n message strings. Provided through the application through context.

type: `custom`

`map` (required)

The ol3 map to export as PNG.

type: `instanceOf ol.Map`

`InfoPopup` (component)

Popup to show feature info. This can be through WMS GetFeatureInfo or local vector data.

```
<div id='popup' className='ol-popup'>
  <InfoPopup toggleGroup='navigation' map={map} />
</div>
```

Properties

`hover`

Should we show feature info on hover instead of on click?

type: `bool` defaultValue: `false`

`intl`

i18n message strings. Provided through the application through context.

type: `custom`

`map` (required)

The ol3 map to register for singleClick.

type: `instanceOf ol.Map`

`LayerList` (component)

A list of layers in the map. Allows setting visibility and opacity.

```
<div id='layerlist'>
  <LayerList allowFiltering={true} showOpacity={true} showDownload={true} showGrou
pContent={true} showZoomTo={true} allowReordering={true} map={map} />
</div>
```

Properties

`addLayer`

Should we allow adding layers through WMS or WFS GetCapabilities? Object with keys url (string, required), allowUserInput (boolean, optional) and asVector (boolean, optional). If asVector is true, layers will be retrieved from WFS and added as vector. If allowUserInput is true, the user will be able to provide a url through an input.

type: `shape [object Object]`

`allowEditing`

Should we allow for editing of features in a vector layer? This does require having a WFST component in your application.

type: `bool` defaultValue: `false`

`allowFiltering`

Should we allow for filtering of features in a layer?

type: `bool` defaultValue: `false`

`allowLabeling`

Should we allow for labeling of features in a layer?

type: `bool` defaultValue: `false`

`allowReordering`

Should we allow for reordering of layers?

type: `bool` defaultValue: `false`

`allowStyling`

Should we allow for styling of features in a vector layer?

type: `bool` defaultValue: `false`

`downloadFormat`

The feature format to serialize in for downloads.

type: `enum ('GeoJSON'|'KML'|'GPX')` defaultValue: `'GeoJSON'`

`expandOnHover`

Should we expand when hovering over the layers button?

type: `bool` defaultValue: `true`

`intl`

i18n message strings. Provided through the application through context.

type: `custom`

`map` (required)

The map whose layers should show up in this layer list.

type: `instanceOf ol.Map`

`showDownload`

Should we show a download button for layers?

type: `bool` defaultValue: `false`

`showGroupContent`

Should we show the contents of layer groups?

type: `bool` defaultValue: `true`

`showOnStart`

Should we show this component on start of the application?

type: `bool` defaultValue: `false`

`showOpacity`

Should we show an opacity slider for layers?

type: `bool` defaultValue: `false`

`showZoomTo`

Should we show a button that allows the user to zoom to the layer's extent?

type: `bool` defaultValue: `false`

`tipLabel`

Text to show on top of layers.

type: `string`

`LoadingPanel` (component)

A loading panel shows a spinner when tiles or images are loading.

```
<LoadingPanel map={map} />
```

Properties

`map` (required)

type: `instanceOf ol.Map`

`Login` (component)

Button that shows a login dialog for integration with GeoServer security.

```
<Login />
```

Properties

`intl`

i18n message strings. Provided through the application through context.

type: `custom`

`url`

Url to geoserver.

type: `string` defaultValue: `'/geoserver/'`

`MapConfig` (component)

Export the map configuration and ability to reload it from local storage.

```
<MapConfig map={map} />
```

Properties

`intl`

i18n message strings. Provided through the application through context.

type: `custom`

`map` (required)

The ol3 map to save the layers from.

type: `instanceOf ol.Map`

`Measure` (component)

Adds area and length measure tools as a menu button.

```
<Measure toggleGroup='navigation' map={map}/>
```

Properties

`geodesic`

Should measurements be geodesic?

type: `bool` defaultValue: `true`

`intl`

i18n message strings. Provided through the application through context.

type: `custom`

`map` (required)

The map onto which to activate and deactivate the interactions.

type: `instanceOf ol.Map`

`toggleGroup`

The toggleGroup to use. When this tool is activated, all other tools in the same toggleGroup will be deactivated.

type: `string`

`Playback` (component)

Adds a slider to the map that can be used to select a given date, and modifies the visibility of layers and features depending on their timestamp and the current time.

```
class PlaybackApp extends App {
  render() {
    return (
      <div id='content'>
        <div ref='map' id='map'>
          <div id='timeline'><Playback map={map} minDate={324511200000} maxDate={1
385938800000} /></div>
        </div>
      </div>
    );
  }
}
```

Properties

`autoPlay`

Should the playback tool start playing automatically?

type: `bool` defaultValue: `false`

`interval`

The time, in milliseconds, to wait in each position of the slider. Positions are defined by dividing the slider range by the number of intervals defined in the numIntervals parameter.

type: `number` defaultValue: `500`

`intl`

i18n message strings. Provided through the application through context.

type: `custom`

`map` (required)

The map whose time-enabled layers should be filtered. Time-enabled layers are layers that have a timeInfo property.

type: `instanceOf ol.Map`

`maxDate` (required)

The maximum date to use for the slider field and the date picker.

type: `number`

`minDate` (required)

The minimum date to use for the slider field and the date picker.

type: `number`

`numIntervals`

The number of intervals into which the full range of the slider is divided.

type: `number` defaultValue: `100`

`QGISLegend` (component)

A component that shows a legend based on artefacts created by the QGIS plugin Web Application Builder.

```
var legendData = {
  'geo20130827100512660': [
    {
     'href': '0_1.png',
      'title': 'Hill_111'
    }, {
      'href': '0_2.png',
      'title': 'Hill_112'
    }
  ],
  'pt120130827095302041': [
    {
      'href': '2_0.png',
      'title': '85.0-116.84'
    }, {
      'href': '2_1.png',
      'title': '116.84-148.68'
    }
  ]
};
```

```
<div id='legend'>
  <QGISLegend map={map} legendBasePath='./resources/legend/' legendData={legendDat
a} pullRight/>
</div>
```

Properties

`expandOnHover`

Should we expand when hovering over the legend button?

type: `bool` defaultValue: `true`

`intl`

i18n message strings. Provided through the application through context.

type: `custom`

`legendBasePath`

The base path (relative url) to use for finding the artefacts.

type: `string` defaultValue: `'./legend/'`

`legendData` (required)

The label and image to use per layer. The object is keyed by layer name currently. For example: {'swamp': [['', '4_0.png']]}.

type: `object`

`map` (required)

The map from which to extract the layers.

type: `instanceOf ol.Map`

`showExpandedOnStartup`

Should we expand on startup of the application?

type: `bool` defaultValue: `false`

`QGISPrint` (component)

A print component which is dependent on artefacts generated by QGIS Web Application Builder.

```
var printLayouts = [{
  name: 'Layout 1',
  thumbnail: 'layout1_thumbnail.png',
  width: 420.0,
  elements: [{
    name: 'Title',
    height: 40.825440467359044,
    width: 51.98353115727002,
    y: 39.25222551928783,
    x: 221.77507418397624,
    font: 'Helvetica',
    type: 'label',
    id: '24160ce7-34a3-4f25-a077-8910e4889681',
    size: 18
  }, {
    height: 167.0,
    width: 171.0,
    grid: {
      intervalX: 0.0,
      intervalY: 0.0,
      annotationEnabled: false,
      crs: ''
    },
    y: 19.0,
    x: 16.0,
    type: 'map',
    id: '3d532cb9-0eca-4e50-9f0a-ce29b1c7f5a6'
  }],
  height: 297.0
}];
```

```
<QGISPrint map={map} layouts={printLayouts} />
```

Properties

`intl`

i18n message strings. Provided through the application through context.

type: `custom`

`layouts` (required)

An array of print layouts. Each layout is an object with keys such as: name (string, required), thumbnail (string, required), width (number, required), height (number, required) and an array of elements. Elements are objects with keys such as name (string, optional), type (enum('map', 'label', legend'), optional), height (number, required), width (number, required), x (number, required), y (number, required), font (string), id (string, required), size (number), grid (object with intervalX, intervalY, annotationEnabled and crs keys).

type: `arrayOf shape`

`map` (required)

The ol3 map to use for printing.

type: `instanceOf ol.Map`

`resolutions`

A list of resolutions from which the user can choose from. Please note that artefacts for all resolutions need to get pre-generated by QGIS.

type: `array` defaultValue: `[72, 150, 300]`

`thumbnailPath`

The relative path where thumbnails of the print layouts can be found. Thumbnails are also generated by QGIS.

type: `string` defaultValue: `'../../resources/print/'`

`QueryBuilder` (component)

A component that allows users to perform queries on vector layers. Queries can be new queries, added to existing queries or users can filter inside of an existing query a.k.a. drill-down.

```
<QueryBuilder map={map} />
```

Properties

`intl`

i18n message strings. Provided through the application through context.

type: `custom`

`map` (required)

The ol3 map whose layers can be used for the querybuilder.

type: `instanceOf ol.Map`

`Select` (component)

The select tool allows users to select features in multiple layers at a time by drawing a rectangle.

```
<Select toggleGroup='navigation' map={map}/>
```

Properties

`intl`

i18n message strings. Provided through the application through context.

type: `custom`

`Toolbar` (component)

Adds the ability to show a toolbar with buttons. On small screen sizes a dropdown will be shown instead.

```
var options = [{
  jsx: (<Login />)
}, {
  jsx: (<ImageExport map={map} />)
}, {
  jsx: (<Measure toggleGroup='navigation' map={map}/>)
}, {
  jsx: (<AddLayer map={map} />)
}, {
  jsx: (<QGISPrint map={map} layouts={printLayouts} />)
}, {
  jsx: (<Select toggleGroup='navigation' map={map}/>)
}, {
  text: formatMessage(messages.navigationbutton),
  title: formatMessage(messages.navigationbuttontitle),
  onClick: this._navigationFunc.bind(this),
  icon: 'hand-paper-o'
}];
```

```
<Toolbar options={options} />
```

Properties

`media`

Handled automatically by the responsive decorator.

type: `func`

`options` (required)

The options to show in the toolbar. An array of objects with jsx (element), icon (string), text (string), title (string), pullRight (boolean) and onClick (function) keys. When using jsx, use exclude to not have the item show up in the menu on small screens, but

separate in the toolbar.

type: `arrayOf shape`

`width`

Width in pixels below which mobile layout should kick in.

type: `number` defaultValue: `1024`

`WFST` (component)

Allows users to make changes to WFS-T layers. This can be drawing new features and deleting or modifying existing features. Only geometry changes are currently supported, no attribute changes. This depends on ol.layer.Vector with ol.source.Vector. The layer needs to have isWFST set to true. Also a wfsInfo object needs to be configured on the layer with the following properties:

- featureNS: the namespace of the WFS typename
- featureType: the name (without prefix) of the underlying WFS typename
- geometryType: the type of geometry (e.g. MultiPolygon)
- geometryName: the name of the geometry attribute
- url: the online resource of the WFS endpoint

```
<WFST map={map} />
```

Properties

`intl`

i18n message strings. Provided through the application through context.

type: `custom`

`layerSelector`

Show a layer selector for the user to choose from?

type: `bool` defaultValue: `true`

`map` (required)

The ol3 map whose layers can be used for the WFS-T tool.

type: `instanceOf ol.Map`

`visible`

Should this component be visible from the start?

type: `bool` defaultValue: `true`