Controller

The base class for all viewport controllers.

A controller class can be passed to either the `Deck` class's controller prop or a `View` class's controller prop to specify viewport interactivity.

Options

The base Controller class supports the following options:

- `scrollZoom` (Boolean|Object) - enable zooming with mouse wheel. Default `true`. If an object is supplied, it may contain the following fields to customize the zooming behavior:
  - `speed` (Number) - scaler that translates wheel delta to the change of viewport scale. Default `0.01`.
  - `smooth` (Boolean) - smoothly transition to the new zoom. If enabled, will provide a slightly lagged but smoother experience. Default `false`.
- `dragPan` (Boolean) - enable panning with pointer drag. Default `true`
- `dragRotate` (Boolean) - enable rotating with pointer drag. Default `true`
- `doubleClickZoom` (Boolean) - enable zooming with double click. Default `true`
- `touchZoom` (Boolean) - enable zooming with multi-touch. Default `true`
- `touchRotate` (Boolean) - enable rotating with multi-touch. Use two-finger rotating gesture for horizontal and three-finger swiping gesture for vertical rotation. Default `false`
- `keyboard` (Boolean|Object) - enable interaction with keyboard. Default `true`. If an object is supplied, it may contain the following fields to customize the keyboard behavior:
  - `zoomSpeed` (Number) - speed of zoom using +/- keys. Default `2`.
  - `moveSpeed` (Number) - speed of movement using arrow keys, in pixels.
  - `rotateSpeedX` (Number) - speed of rotation using shift + left/right arrow keys, in degrees. Default `15`.
  - `rotateSpeedY` (Number) - speed of rotation using shift + up/down arrow keys, in degrees. Default `10`.

- `dragMode` (String) - drag behavior without pressing function keys, one of `pan` and `rotate`.
- `inertia` (Boolean|Number) - Enable inertia after panning/pinching. If a number is provided, indicates the duration of time over which the velocity reduces to zero, in milliseconds. Default `false`.

Methods

> A controller is not meant to be instantiated by the application. The following methods are documented for creating custom controllers that extend the base Controller class.

constructor

```
import {Controller} from 'deck.gl';

class MyController extends Controller {
  constructor(props) {
    super(props);
  }
}
```

The constructor takes one argument:

- `props` (Object) - contains the following options:
  - `eventManager`- handles events subscriptions
  - `makeViewPort (viewState)` - creates new `Viewport` based on provided `ViewState`, and current view's `width` and `height`
  - `onStateChange` callback function
  - `onViewStateChange` callback function
  - `timeline` - an instance of `luma.gl` animation timeline class

`handleEvent(event)`

Called by the event manager to handle pointer events. This method delegate to the following methods to handle the default events:

- `_onPanStart(event)`
- `_onPan(event)`

- `_onPanEnd(event)`
- `_onPinchStart(event)`
- `_onPinch(event)`
- `_onPinchEnd(event)`
- `_onTriplePanStart(event)`
- `_onTriplePan(event)`
- `_onTriplePanEnd(event)`
- `_onDoubleTap(event)`
- `_onWheel(event)`
- `_onKeyDown(event)`

See Event object documentation.

`setProps(props)`

Called by the view when the view state updates. This method handles adding/removing event listeners based on user options.

`updateViewport(newMapState, extraProps, interactionState)`

Called by the event handlers, this method updates internal state, and invokes `onViewStateChange` callback with a new map state.

`getCenter(event)`

Utility used by the event handlers, returns pointer position `[x, y]` from any event.

`isFunctionKeyPressed(event)`

Utility used by the event handlers, returns `true` if ctrl/alt/meta key is pressed during any event.

`isPointInBounds(pos, [event])`

Utility used by the event handlers, returns `true` if a pointer position `[x, y]` is inside the current view.

If `event` is provided, returns `false` if the event is already handled, and mark the event as handled if the point is in bounds. This can be used to make sure that certain events are only handled by one controller, when there are overlapping viewports.

`isDragging()`

Returns `true` if the user is dragging the view.

Example: Implementing A Custom Controller

```
import {Controller} from 'deck.gl';

class MyController extends Controller{
  constructor(props) {
    super(props);
    this.events = ['pointermove'];
  }

  handleEvent(event) {
    if (event.type === 'pointermove') {
      // do something
    } else {
      super.handleEvent(event);
    }
  }
}
```

In its constructor, a controller class can optionally specify a list of event names that it subscribes to with the `events` field. A full list of supported events can be found here.

Note that the following events are always toggled on/off by user options:

- `scrollZoom` - `['wheel']`
- `dragPan` and `dragRotate` - `['pan']`
- `touchZoom` - `['pinch']`
- `touchRotate` - `['pinch', 'tripan']`
- `doubleClickZoom` - `['doubletap']`
- `keyboard` - `['keydown']`

Source

modules/core/src/controllers/controller.ts

Edit this page