
jQuery Seat Charts

jQuery Seat Charts (JSC) is a full-blown seat map library. It will generate an accessible map, legend, handle mouse & keyboard events and finally give you powerful selectors to control your map.

Simple demo map

Example:

Basic setup:

```
1 $(document).ready(function() {
2
3     var sc = $('#seat-map').seatCharts({
4         map: [
5             'aaaaaaaaaaaa',
6             'aaaaaaaaaaaa',
7             'bbbbbbbbbb__',
8             'bbbbbbbbbb__',
9             'bbbbbbbbbb',
10            'cccccccccc'
11        ],
12        seats: {
13            a: {
14                price    : 99.99,
15                classes  : 'front-seat' //your custom CSS class
16            }
17        },
18        click: function () {
19            if (this.status() == 'available') {
20                //do some stuff, i.e. add to the cart
21                return 'selected';
22            } else if (this.status() == 'selected') {
23                //seat has been vacated
24                return 'available';
25            } else if (this.status() == 'unavailable') {
26                //seat has been already booked
27                return 'unavailable';
28            } else {
29                return this.style();
30            }
31        }
32    });
33
34    //Make all available 'c' seats unavailable
```

```

36     sc.find('c.available').status('unavailable');
37
38     /*
39     Get seats with ids 2_6, 1_7 (more on ids later on),
40     put them in a jQuery set and change some css
41     */
42     sc.get(['2_6', '1_7']).node().css({
43         color: '#ffcfcf'
44     });
45
46     console.log('Seat 1_2 costs ' + sc.get('1_2').data().price + ' and
47         is currently ' + sc.status('1_2'));
48 });

```

Basics:

Building maps is fairly easy with jQuery Seat Charts, you can literally pass an array of strings which represents succeeding rows. Let's take a look at a theatre example:

```

1 //Seat map definition
2 [
3     'aaaaaa__DDDDD',
4     'aaaaaa__aaaaa',
5     'aaaaaa__aaaaa',
6     'bbbbbb__bbbbbb',
7     'bbbbbb__bbbbbb',
8     'bbbbbb__bbbbbb',
9     'cccccccccccccc'
10 ]

```

Each single character represents a different type of seat and you have a freedom of choosing anyone but underscore ****_**. **Underscore is used to indicate that there shouldn't be any seat at a certain place. In our example I chose a**** seats to be the closest to the screen, **D** meant for disabled and **b** and **c** as just plain seats. I also built a corridor in the middle of our theatre, so people can conveniently reach their seats.

Your chosen characters can carry a hash of data which is a great way to pass crucial seat details such as price or a description that you want to show on hover.

```

1 seats: {
2     a: {
3         price      : 24.55,
4         description : 'Fair priced seat!'
5     }
6 }

```

Once you build your map and define seats, you can start implementing the booking magic.

Booking Magic

JSC combines keyboard and mouse events to offer a unified API. There're three types of events which JSC can produce:

- **click**: click or spacebar
- **focus**: mouse or arrows
- **blur**: mouse or arrows

All three events have their default handlers but you're more than likely to overwrite at least one of them. JSC flexible API let you choose where you want to specify your handlers. You can define global click handlers like in the *Basic setup* example at the very beginning or you can implement separate handlers for each *character*:

```
1 a: {
2   click    : function () {
3     //This will only be applied to a seats
4   },
5   price    : 34.99,
6   category : 'VIP Seats'
7 }
```

Each event handler is fired in *seat* context which gives you an easy access (using *this* variable) to its properties, DOM node and data which you may have specified during the setup:

```
1 click: function () {
2   if (this.status() == 'available') {
3     //seat's available and can be taken!
4
5     //let's retrieve the data, so we can add the seat to our cart
6     var price    = this.data().price,
7         category = this.data().category;
8
9     //jQuery element access example
10    this.node().css({
11      'font-size' : '25px'
12    });
13
14    //return new seat status
15    return 'selected';
16  }
17  //...
18 }
```

Please note: event handler should return new status of a seat depending on what happened. If user clicks on a seat and the seat's *available*, *selected* status should be returned. If user clicks on a *selected* seat, it most likely should become *available* again. Full status reference:

- **available:** seat which can be taken
- **unavailable:** seat which cannot be taken
- **selected:** seat which has been taken by current user

Since JSC also works with *focus/blur* events, it features a special status called *focused* which actually doesn't apply to seat status but rather to the way it's displayed. If you use *.status* method on a focused seat, you will get its real status. To get an idea of this, please take a look at how events are handled by default:

```
1 click : function() {
2
3     if (this.status() == 'available') {
4         return 'selected';
5     } else if (this.status() == 'selected') {
6         return 'available';
7     } else {
8         /*
9         If we don't want to change the status (i.e. seat's unavailable)
            we ought to return this.style(). this.style() is a
            reference to seat's special status which means that it can
            be focused as well. You shouldn't return this.status() here
10        */
11        return this.style();
12    }
13 },
14 },
15 focus : function() {
16
17     if (this.status() == 'available') {
18         //if seat's available, it can be focused
19         return 'focused';
20     } else {
21         //otherwise nothing changes
22         return this.style();
23     }
24 },
25 blur : function() {
26     //The only place where you should return actual seat status
27     return this.status();
28 },
```

Your site's popular and people fight for your tickets? Don't forget to update your map with new bookings live!

```
1 //sc will contain a reference to the map
2 var sc = $('#sc-container').seatCharts({
3     //...
4 });
5
6 setInterval(function() {
7     $.ajax({
8         type      : 'get',
9         url       : '/bookings/get/100',
10        dataType : 'json',
11        success  : function(response) {
12            //iterate through all bookings for our event
13            $.each(response.bookings, function(index, booking) {
14                //find seat by id and set its status to unavailable
15                sc.status(booking.seat_id, 'unavailable');
16            });
17        }
18    });
19 }, 10000); //every 10 seconds
```

Options

Required params are marked with *

animate

Bool, enables animated status switches.

Please note: *animate* uses *switchClass* method of jQuery UI, so if you want to use *animate*, you need to include jQuery UI in the page.

blur

Blur handler. Fired when seat loses focus due to mouse move or arrow hit. You most likely don't want to overwrite this one.

```
1 //default handler
2 blur    : function() {
3     return this.status();
4 },
```

click

Click handler. Fired when user clicks on a seat or hits spacebar on a focused seat. You're most likely to overwrite this one based off this example:

```
1 click : function() {
2
3     if (this.status() == 'available') {
4         //do some custom stuff
5         console.log(this.data());
6
7         return 'selected';
8     } else if (this.status() == 'selected') {
9         //do some custom stuff
10
11         return 'available';
12     } else {
13         //i.e. alert that the seat's not available
14
15         return this.style();
16     }
17
18 },
```

focus

Focus handler. Fired when seat receives focus. You most likely don't want to overwrite this one.

```
1 //default handler
2 focus : function() {
3
4     if (this.status() == 'available') {
5         return 'focused';
6     } else {
7         return this.style();
8     }
9 },
```

legend

JSC is able to create an UL element with a map legend based on your seat types and custom CSS. If you want JSC to generate a legend for you, you will just need to pass some basic information:

node jQuery reference to a DIV element where legend should be rendered. If it's missing, JSC will create a DIV container itself.

```
1 node : $('#my-legend-container')
```

items An array of legend item details. Each array element should be a three-element array: [*character, status, description*]

```
1 legend : {  
2   node : $('#my-legend-container'),  
3   items : [  
4     [ v, 'available', 'VIP seats!' ],  
5     [ e, 'available', 'Economy seats'],  
6     [ e, 'unavailable', 'Unavailable economy seats' ]  
7   ]  
8 }
```

map*

An array of strings that represents your map:

```
1 [  
2   'aaa__aaa',  
3   'aaaa_aaaa',  
4   'aaaa_aaaa'  
5 ]
```

Underscore is used as a spacer between seats.

Please note: number of columns must be equal in each row.

New: You can now override label and ID per character. This is optional and can be applied to any number of seats:

```
1 [  
2   'a[ID, LABEL]a[ID2, LABEL2]a__a[JUST_ID1]aa',  
3   'aaaa_aaaa[, JUST_LABEL1]',  
4   'aaaa_aaaa'  
5 ]
```

ID and/or label should be specified after the letter and enclosed in square brackets. ID should go first, optionally followed by custom label. If you just want to specify label without overriding ID, leave ID empty: a[,Just Label]

ID may contain letters, numbers and underscores. Label can contain the same groups of characters as well as spaces.

naming

You can specify your own column and row labels as well as functions for generating seat ids and labels.

columns

An array of column names, *columns.length* must equal the actual number of columns:

```
1 columns: ['A', 'B', 'C', 'D', 'E']
```

If you don't define your own columns, succeeding numbers starting from 1 will be used.

getId

Callback which may accept the following parameters: *character*, *row*, *column*, where *row* and *column* are names either specified by you using *columns* and *rows* arrays or by default JSC settings. This function should return an id based off passed arguments. Default getId function:

```
1 getId : function(character, row, column) {  
2     return row + '_' + column;  
3 }
```

Returned id is not only used as an internal identifier but also as a DOM id.

getLabel

Callback which may accept the following parameters: *character*, *row*, *column*, where *row* and *column* are names either specified by you using *columns* and *rows* arrays or by default JSC settings. This function should return a seat label based off passed arguments. Default getLabel function:

```
1 getLabel : function (character, row, column) {  
2     return column;  
3 }
```

Labels will be displayed over seats, so if you don't want any labels, just return an empty string.

Sometimes it can be really hard to generate labels you want with getLabel, so now it's possible to specify custom labels per each seat. Please take a look at the map section.

left

Bool, defaults to true. If true, JSC will display an additional column on the left of the map with row names as specified by you using *rows* array or by default JSC settings

rows

An array of row names, *rows* length must equal the actual number of rows:

```
1 rows: ['I', 'II', 'III', 'IV', 'V']
```

If you don't define your own rows, succeeding numbers starting from 1 will be used.

top

Bool, defaults to true. If true, JSC will display an additional row on the top of the map with column names as specified by you using *columns* array or by default JSC settings

seats

A hash of seat options, seat *characters* should be used as keys. You can pass the following params:

blur

Blur event which should be applied only to seats of a particular *character*.

classes

Custom CSS classes which should be applied to seats. Either an array or a string, JSC doesn't care:

```
1 classes : 'seat-red seat-big'
2 //equals
3 classes : ['seat-red', 'seat-big']
```

click

Custom click handler.

focus

Custom focus handler.

Selectors

JSC offers you two flexible selector methods that are chainable and return *set* of seats:

.get(ids)

You can pass either one id or an array of ids:

```
1 sc.get('2_3'); //get 2_3 seat
2 sc.get(['2_3', '2_4']); //get 2_3 and 2_4 seats
```

.find(mixed)

Find method lets you search using *character*, seat status, combination of both (separated with a dot) or a regexp:

```
1 sc.find('a'); //find all a seats
2 sc.find('unavailable'); //find all unavailable seats
3 sc.find('a.available'); //find all available a seats
4 sc.find(/^1_[0-9]+/); //find all seats in the first row
```

.get and .find chained together:

```
1 sc.get(['1_2', '1_3', '1_4']).find('available'); //find available seats
   within specified seat ids
```

Both methods return either one seat or a set of seats which share similar methods:

Set Methods

.status(ids, status)

Update status for a seat set with given ids. *ids* variable may contain a single id or a an array of ids.

```
1 sc.status('2_15', 'unavailable'); //set status for one seat
2 sc.status(['2_15', '2_10'], 'unavailable'); //set status for two seats
```

.status(status)

Update status for all seats in the current set.

```
1 sc.find('unavailable').status('available'); //make all unavailable seats
   available
```

.node()

Returns a jQuery set of seat node references.

```
1 sc.find('unavailable').node().fadeOut('fast'); //make all unavailable
   seats disappear
```

.each(callback)

Iterates through a seat set, callback will be fired in the context of each element. Callback may accept seat id as an argument.

```
1 sc.find('a.unavailable').each(function(seatId) {  
2     console.log(this.data()); //display seat data  
3 });
```

You can break the loop returning *false*.

Seat Methods

.status([status])

If *status* argument is set, it will be used as a new seat status, otherwise current status will be returned.

.node()

Returns a reference to jQuery element.

.data()

Returns a reference to seat data.

.char()

Returns seat *character*.

Styling

JSC uses a few CSS classes that are pretty self explanatory:

.seatCharts-container

DIV container where seat chart's rendered.

.seatCharts-row

DIV element which serves as a row. You're most likely to edit its height.

.seatCharts-cell

This class is applied to both seats and spacers (_).

.seatCharts-seat

Applied to all seats regardless of character.

.seatCharts-space

Applied to spacers.

.seatCharts-seat.selected

Selected seats.

.seatCharts-seat.focused

Focused seats.

.seatCharts-seat.available

Available seats.

.seatCharts-seat.unavailable

Unavailable seats.

Please note: if you need each of your seat type (indicated by character) look differently, this is the easiest way:

CSS: `.seatCharts-seat.selected.vip { background-color: #ff4fff; }`

```
1 .seatCharts-seat.focused.vip {
2     background-color: #ccffcc;
3 }
4
5 //...
6
7 .seatCharts-seat.selected.economy {
8     background-color: #000fff;
9 }
10
11 //...
```

JavaScript:

```
1 var sc = $.seatCharts({
2     seats: {
3         v: {
4             classes: 'vip',
5             price : 300
6         },
7         e: {
8             classes: 'economy',
9             price : 50
10        }
11    }
12    //...
13 });
```

.seatCharts-legendList

UL element which holds the legend.

.seatCharts-legendItem

LI element of the legend.

FAQ

What licence is jQuery Seat Charts released under? jQuery Seat Charts is released under MIT license.

How is JSC accessible? JSC implements WAI-ARIA standard meaning that people using solely keyboards will share the same experience as mouse-users. You can easily check it yourself navigating with arrows and hitting spacebar instead of mouse click.