Bootstrap Tags Input

jQuery plugin providing a Twitter Bootstrap user interface for managing tags

Code on Github | Bootstrap 2.3.2 | Download

Star 2,884 | Fork 983 | Tweet

Examples

Markup

Just add `data-role="tagsinput"` to your input field to automatically change it to a tags input field.

Amsterdam x Washington x Sydney x Beijing x Cairo x

Show code

```
<input type="text" value="Amsterdam,Washington,Sydney,Beijing,Cairo" data-role="tagsinput" />
```

statement	returns
`$("input").val()`	`"Amsterdam,Washington,Sydney,Beijing,Cairo"`
`$("input").tagsinput('items')`	`["Amsterdam","Washington","Sydney","Beijing","Cairo"]`

True multi value

Use a `<select multiple />` as your input element for a tags input, to gain true multivalue support. Instead of a comma separated string, the values will be set in an array. Existing `<option />` elements will automatically be set as tags. This makes it also possible to create tags containing a comma.

Amsterdam x Washington x Sydney x Beijing x Cairo x

Show code

```
<select multiple data-role="tagsinput">
  <option value="Amsterdam">Amsterdam</option>
  <option value="Washington">Washington</option>
  <option value="Sydney">Sydney</option>
  <option value="Beijing">Beijing</option>
  <option value="Cairo">Cairo</option>
</select>
```

statement	returns
`$("select").val()`	`["Amsterdam","Washington","Sydney","Beijing","Cairo"]`
`$("select").tagsinput('items')`	`["Amsterdam","Washington","Sydney","Beijing","Cairo"]`

Typeahead

Typeahead is not included in Bootstrap 3, so you'll have to include your own typeahead library. I'd recommed typeahead.js. An example of using this is shown below.

Amsterdam x Washington x

Show code

```
<input type="text" value="Amsterdam,Washington" data-role="tagsinput" />
<script>
var citynames = new Bloodhound({
  datumTokenizer: Bloodhound.tokenizers.obj.whitespace('name'),
  queryTokenizer: Bloodhound.tokenizers.whitespace,
  prefetch: {
    url: 'assets/citynames.json',
    filter: function(list) {
      return $.map(list, function(cityname) {
        return { name: cityname }; });
    }
  }
});
citynames.initialize();

$('input').tagsinput({
  typeaheadjs: {
    name: 'citynames',
    displayKey: 'name',
    valueKey: 'name',
    source: citynames.ttAdapter()
  }
});
</script>
```

statement	returns
`$("input").val()`	`"Amsterdam,Washington"`
`$("input").tagsinput('items')`	`["Amsterdam","Washington"]`

Objects as tags

Instead of just adding strings as tags, bind objects to your tags. This makes it possible to set id values in your input field's value, instead of just the tag's text.

Amsterdam x Washington x Sydney x Beijing x Cairo x

Show code

```
<input type="text" />
<script>
var cities = new Bloodhound({
  datumTokenizer: Bloodhound.tokenizers.obj.whitespace('text'),
  queryTokenizer: Bloodhound.tokenizers.whitespace,
  prefetch: 'assets/cities.json'
});
cities.initialize();

var elt = $('input');
elt.tagsinput({
  itemValue: 'value',
  itemText: 'text',
  typeaheadjs: {
    name: 'cities',
    displayKey: 'text',
    source: cities.ttAdapter()
  }
});
elt.tagsinput('add', { "value": 1 , "text": "Amsterdam"   , "continent": "Europe"    });
elt.tagsinput('add', { "value": 4 , "text": "Washington"  , "continent": "America"   });
elt.tagsinput('add', { "value": 7 , "text": "Sydney"      , "continent": "Australia" });
elt.tagsinput('add', { "value": 10, "text": "Beijing"     , "continent": "Asia"      });
elt.tagsinput('add', { "value": 13, "text": "Cairo"       , "continent": "Africa"    });
</script>
```

statement	returns
`$("input").val()`	`"1,4,7,10,13"`
`$("input").tagsinput('items')`	`[{"value":1,"text":"Amsterdam","continent":"Europe"},{"value":4,"text":"Washington","continent":"America"},{"value":7,"text":"Sydney","continent":"Australia"},{"value":10,"text":"Beijing","continent":"Asia"},{"value":13,"text":"Cairo","continent":"Africa"}]`

Categorizing tags

You can set a fixed css class for your tags, or determine dynamically by providing a custom function.

Amsterdam x Washington x Sydney x Beijing x Cairo x

Show code

```
<input type="text" />
<script>
var cities = new Bloodhound({
  datumTokenizer: Bloodhound.tokenizers.obj.whitespace('text'),
  queryTokenizer: Bloodhound.tokenizers.whitespace,
  prefetch: 'assets/cities.json'
});
cities.initialize();

var elt = $('input');
elt.tagsinput({
  tagClass: function(item) {
    switch (item.continent) {
      case 'Europe'   : return 'label label-primary';
      case 'America'  : return 'label label-danger label-important';
      case 'Australia': return 'label label-success';
      case 'Africa'   : return 'label label-default';
      case 'Asia'     : return 'label label-warning';
    }
  },
  itemValue: 'value',
  itemText: 'text',
  typeaheadjs: {
    name: 'cities',
    displayKey: 'text',
    source: cities.ttAdapter()
  }
});
elt.tagsinput('add', { "value": 1 , "text": "Amsterdam"   , "continent": "Europe"    });
elt.tagsinput('add', { "value": 4 , "text": "Washington"  , "continent": "America"   });
elt.tagsinput('add', { "value": 7 , "text": "Sydney"      , "continent": "Australia" });
elt.tagsinput('add', { "value": 10, "text": "Beijing"     , "continent": "Asia"      });
elt.tagsinput('add', { "value": 13, "text": "Cairo"       , "continent": "Africa"    });
</script>
```

statement	returns
`$("input").val()`	`"1,4,7,10,13"`
`$("input").tagsinput('items')`	`[{"value":1,"text":"Amsterdam","continent":"Europe"},{"value":4,"text":"Washington","continent":"America"},{"value":7,"text":"Sydney","continent":"Australia"},{"value":10,"text":"Beijing","continent":"Asia"},{"value":13,"text":"Cairo","continent":"Africa"}]`

Options

option	description

`tagClass` — Classname for the tags, or a function returning a classname

```
$('input').tagsinput({
  tagClass: 'big'
});
```

```
$('input').tagsinput({
  tagClass: function(item) {
    return (item.length > 10 ? 'big' : 'small');
  }
});
```

`itemValue` — When adding objects as tags, itemValue *must* be set to the name of the property containing the item's value, or a function returning an item's value.

```
$('input').tagsinput({
    itemValue: 'id'
});
```

```
$('input').tagsinput({
  itemValue: function(item) {
    return item.id;
  }
});
```

`itemText` — When adding objects as tags, you can set itemText to the name of the property of item to use for a its tag's text. You may also provide a function which returns an item's value. When this options is not set, the value of `itemValue` will be used.

```
$('input').tagsinput({
  itemText: 'label'
});
```

```
$('input').tagsinput({
  itemText: function(item) {
    return item.label;
  }
});
```

`confirmKeys` — Array of keycodes which will add a tag when typing in the input. (default: [13, 188], which are ENTER and comma)

```
$('input').tagsinput({
  confirmKeys: [13, 44]
});
```

`maxTags` — When set, no more than the given number of tags are allowed to add (default: undefined). When maxTags is reached, a class 'bootstrap-tagsinput-max' is placed on the tagsinput element.

```
$('input').tagsinput({
  maxTags: 3
});
```

`maxChars` — Defines the maximum length of a single tag. (default: undefined)

```
$('input').tagsinput({
  maxChars: 8
});
```

`trimValue` — When true, automatically removes all whitespace around tags. (default: false)

```
$('input').tagsinput({
  trimValue: true
});
```

`allowDuplicates` — When true, the same tag can be added multiple times. (default: false)

```
$('input').tagsinput({
  allowDuplicates: true
});
```

`focusClass` — When the input container has focus, the class specified by this config option will be applied to the container

```
$('input').tagsinput({
  focusClass: 'my-focus-class'
});
```

`freeInput` — Allow creating tags which are not returned by typeahead's source (default: true)

This is only possible when using string as tags. When itemValue option is set, this option will be ignored.

```
$('input').tagsinput({
  typeahead: {
    source: ['Amsterdam', 'Washington', 'Sydney', 'Beijing', 'Cairo']
  },
  freeInput: true
});
```

`typeahead` — Object containing typeahead specific options

`source` — An array (or function returning a promise or array), which will be used as source for a typeahead.

```
$('input').tagsinput({
  typeahead: {
    source: ['Amsterdam', 'Washington', 'Sydney', 'Beijing', 'Cairo']
  }
});
```

```
$('input').tagsinput({
  typeahead: {
    source: function(query) {
      return $.get('http://someservice.com');
    }
  }
});
```

`cancelConfirmKeysOnEmpty` — Boolean value controlling whether form submissions get processed when pressing enter in a field converted to a tagsinput (default: false).

```
$('input').tagsinput({
  cancelConfirmKeysOnEmpty: true
});
```

`onTagExists` — Function invoked when trying to add an item which allready exists. By default, the existing tag hides and fades in.

```
$('input').tagsinput({
  onTagExists: function(item, $tag) {
    $tag.hide().fadeIn();
  }
});
```

Methods

method	description

`add` — Adds a tag

```
$('input').tagsinput('add', 'some tag');
```

```
$('input').tagsinput('add', { id: 1, text: 'some tag' });
```

Optionally, you can pass a 3rd parameter (object or value) to the `add` method to gain more control over the process. The parameter is exposed in the `options` attribute of the event.

```
$('input').tagsinput('add', 'some tag', {preventPost: true});
```

Usage:

```
$('#tags-input').on('beforeItemAdd', function(event) {
   var tag = event.item;
   // Do some processing here

   if (!event.options || !event.options.preventPost) {
      $.ajax('/ajax-url', ajaxData, function(response) {
         if (response.failure) {
            // Remove the tag since there was a failure
            // "preventPost" here will stop this ajax call from running when the tag is removed
            $('#tags-input').tagsinput('remove', tag, {preventPost: true});
         }
      });
   }
});
```

`remove` — Removes a tag

```
$('input').tagsinput('remove', 'some tag');
```

```
$('input').tagsinput('remove', { id: 1, text: 'some tag' });
```

Optionally, you can pass a 3rd parameter (object or value) to the `remove` method to gain more control over the process. The parameter is exposed in the `options` attribute of the event.

```
$('input').tagsinput('remove', 'some tag', {preventPost: true});
```

Usage:

```
$('#tags-input').on('beforeItemRemove', function(event) {
   var tag = event.item;
   // Do some processing here

   if (!event.options || !event.options.preventPost) {
      $.ajax('/ajax-url', ajaxData, function(response) {
         if (response.failure) {
            // Re-add the tag since there was a failure
            // "preventPost" here will stop this ajax call from running when the tag is added
            $('#tags-input').tagsinput('add', tag, {preventPost: true});
         }
      });
   }
});
```

`removeAll` — Removes all tags

```
$('input').tagsinput('removeAll');
```

`focus` — Sets focus in the tagsinput

```
$('input').tagsinput('focus');
```

`input` — Returns the tagsinput's internal <input />, which is used for adding tags. You could use this to add your own typeahead behaviour for example.

```
var $elt = $('input').tagsinput('input');
```

`refresh` — Refreshes the tags input UI. This might be usefull when you're adding objects as tags. When an object's text changes, you'll have to refresh to update the matching tag's text.

```
$('input').tagsinput('refresh');
```

`destroy` — Removes tagsinput behaviour

```
$('input').tagsinput('destroy');
```

Events

event	description

`itemAddedOnInit` — During initialization, pre-defined tags being added will cause this event to be triggered. Example:

```
$('input').on('itemAddedOnInit', function(event) {
  // event.item: contains the item
});
```

`beforeItemAdd` — Triggered just before an item gets added. Example:

```
$('input').on('beforeItemAdd', function(event) {
  // event.item: contains the item
  // event.cancel: set to true to prevent the item getting added
});
```

`itemAdded` — Triggered just after an item got added. Example:

```
$('input').on('itemAdded', function(event) {
  // event.item: contains the item
});
```

`beforeItemRemove` — Triggered just before an item gets removed. Example:

```
$('input').on('beforeItemRemove', function(event) {
  // event.item: contains the item
  // event.cancel: set to true to prevent the item getting removed
});
```

`itemRemoved` — Triggered just after an item got removed. Example:

```
$('input').on('itemRemoved', function(event) {
  // event.item: contains the item
});
```