

Events

Events is a module that can be mixed in to any object, giving the object the ability to bind and trigger custom named events.

| | | |
|----------------|---|---|
| on | <code>object.on(event, callback, [context])[context]</code> | Bind a callback function to an object. |
| off | <code>object.off([event], [callback], [context])</code> | Remove a previously-bound callback function from an object. |
| trigger | <code>object.trigger(event, [*args])</code> | Trigger callbacks for the given event, or space-delimited list of events. |

Model

Models are the heart of any JavaScript application, containing the interactive data as well as a large part of the logic surrounding it: conversions, validations, computed properties, and access control.

| | | |
|---------------------------------|---|--|
| extend | <code>Backbone.Model.extend(properties, [classProperties])</code> | To create a Model class of your own, you extend <code>Backbone.Model</code> and provide instance properties, as well as optional <code>classProperties</code> to be attached directly to the constructor function. |
| constructor / initialize | <code>new Model([attributes])</code> | When creating an instance of a model, you can pass in the initial values of the attributes, which will be set on the model. |
| get | <code>model.get(attribute)</code> | Get the current value of an attribute from the model. |
| set | <code>model.set(attributes, [options])</code> | Set a hash of attributes (one or many) on the model. |
| escape | <code>model.escape(attribute)</code> | Similar to <code>get</code> , but returns the HTML-escaped version of a model's attribute. |
| has | <code>model.has(attribute)</code> | Returns true if the attribute is set to a non-null or non-undefined value. |
| unset | <code>model.unset(attribute, [options])</code> | Remove an attribute by deleting it from the internal attributes hash. |
| clear | <code>model.clear([options])</code> | Removes all attributes from the model. |
| id | <code>model.id</code> | A special property of models, the <code>id</code> is an arbitrary string (integer id or UUID). |
| idAttribute | <code>model.idAttribute</code> | A model's unique identifier is stored under the <code>id</code> attribute. |
| cid | <code>model.cid</code> | A special property of models, the <code>cid</code> or client id is a unique identifier automatically assigned to all models when they're first created. |
| attributes | <code>model.attributes</code> | The <code>attributes</code> property is the internal hash containing the model's state. |
| defaults | <code>model.defaults</code> or <code>model.defaults()</code> | The <code>defaults</code> hash (or function) can be used to specify the default attributes for your model. |
| toJSON | <code>model.toJSON()</code> | Return a copy of the model's attributes for JSON stringification. |
| fetch | <code>model.fetch([options])</code> | Resets the model's state from the server. |
| save | <code>model.save([attributes], [options])</code> | Save a model to your database (or alternative persistence layer), by delegating to <code>Backbone.sync</code> . |
| destroy | <code>model.destroy([options])</code> | Destroys the model on the server by delegating an HTTP DELETE request to <code>Backbone.sync</code> . |
| validate | <code>model.validate(attributes)</code> | Validate is called before <code>set</code> and <code>save</code> , and is passed the attributes that are about to be updated. |
| isValid | <code>model.isValid()</code> | Call <code>model.isValid()</code> to check if the model is currently in a valid state, according to your <code>validate</code> function. |
| url | <code>model.url()</code> | Returns the relative URL where the model's resource would be located on the server. |
| urlRoot | <code>model.urlRoot</code> | Specify a <code>urlRoot</code> if you're using a model outside of a collection, to enable the default <code>url</code> function to generate URLs based on the model id. |
| parse | <code>model.parse(response).url()</code> | <code>Parse</code> is called whenever a model's data is returned by the server, in <code>fetch</code> , and <code>save</code> . |
| clone | <code>model.clone()</code> | Returns a new instance of the model with identical attributes. |
| isNew | <code>model.isNew()</code> | If the model does not yet have an <code>id</code> , it is considered to be new. |
| change | <code>model.change()</code> | Manually trigger the "change" event and a "change:attribute" event for each attribute that has changed. |
| hasChanged | <code>model.hasChanged([attribute])</code> | Has the model changed since the last "change" event? |
| changedAttributes | <code>model.changedAttributes([attributes])</code> | Retrieve a hash of only the model's attributes that have changed. |
| previous | <code>model.previous(attribute)</code> | During a "change" event, this method can be used to get the previous value of a changed attribute. |
| previousAttributes | <code>model.previousAttributes()</code> | Return a copy of the model's previous attributes. |

Collection

Collections are ordered sets of models.

| | | |
|---------------------------------|--|---|
| extend | <code>Backbone.Collection.extend(properties, [classProperties])</code> | To create a Collection class of your own, extend <code>Backbone.Collection</code> , providing instance properties, as well as optional <code>classProperties</code> to be attached directly to the collection's constructor function. |
| model | <code>collection.model</code> | Override this property to specify the model class that the collection contains. |
| constructor / initialize | <code>new Collection([models], [options])</code> | When creating a Collection, you may choose to pass in the initial array of models. The collection's comparator function may be included as an option. |
| models | <code>collection.models</code> | Raw access to the JavaScript array of models inside of the collection. |
| toJSON | <code>collection.toJSON()</code> | Return an array containing the attributes hash of each model in the collection. |
| add | <code>collection.add(models, [options])</code> | Add a model (or an array of models) to the collection. |
| remove | <code>collection.remove(models, [options])</code> | Remove a model (or an array of models) from the collection. |
| get | <code>collection.get(id)</code> | Get a model from a collection, specified by id. |
| getByCid | <code>collection.getByCid(cid)</code> | Get a model from a collection, specified by client id. |

| | | |
|-------------------|--|--|
| at | collection.at(index) | Get a model from a collection, specified by index. |
| length | collection.length | Like an array, a Collection maintains a length property, counting the number of models it contains. |
| comparator | collection.comparator | If you define a comparator, it will be used to maintain the collection in sorted order. |
| sort | collection.sort([options]) | Force a collection to re-sort itself. |
| pluck | collection.pluck(attribute) | Pluck an attribute from each model in the collection. |
| url | collection.url or collection.url() | Set the url property (or function) on a collection to reference its location on the server. |
| parse | collection.parse(response) | Parse is called by Backbone whenever a collection's models are returned by the server, in fetch. |
| fetch | collection.fetch([options]) | Fetch the default set of models for this collection from the server, resetting the collection when |
| reset | collection.reset(models, [options]) | Adding and removing models one at a time is all well and good, but sometimes you have so many models to change that you'd rather just update the collection in bulk. |
| create | collection.create(attributes, [options]) | Convenience to create a new instance of a model within a collection. |

| Router | | |
|---|---|--|
| Router provides methods for routing client-side pages, and connecting them to actions and events. | | |
| extend | Backbone.Router.extend(properties, [classProperties]) | Define actions that are triggered when certain URL fragments are matched. |
| routes | router.routes | The routes hash maps URLs with parameters to functions on your router |
| constructor / initialize | new Router([options]) | When creating a new router, you may pass its routes hash directly as an option, if you choose. |
| route | router.route(route, name, [callback]) | Manually create a route for the router. |
| navigate | router.navigate(fragment, [options]) | Call navigate in order to update the URL. |

| History | | |
|---|-----------------------------------|--|
| History serves as a global router (per frame) to handle hashchange events or pushState, match the appropriate route, and trigger callbacks. | | |
| start | Backbone.history.start([options]) | Call Backbone.history.start() to begin monitoring hashchange events, and dispatching routes. |

| Sync | | |
|--|-----------------------------|--|
| Sync is the function that Backbone calls every time it attempts to read or save a model to the server. | | |
| emulateHTTP | Backbone.emulateHTTP = true | |
| emulateJSON | Backbone.emulateJSON = true | |

| View | | |
|--|---|--|
| The general idea is to organize your interface into logical views, backed by models, each of which can be updated independently when the model changes, without having to redraw the page. | | |
| extend | Backbone.View.extend(properties, [classProperties]) | |
| constructor / initialize | new View([options]) | |
| el | view.el | All views have a DOM element at all times (the el property) |
| \$el | view.\$el | A cached jQuery (or Zepto) object for the view's element. |
| setElement | view.setElement(element) | Call setElement to apply a Backbone view to a different DOM element. |
| attributes | view.attributes | A hash of attributes that will be set as HTML DOM element attributes on the view's el. |
| \$ (jQuery or Zepto) | view.\$(selector) | Function that runs queries scoped within the view's element. |
| render | view.render() | Override this function with your code that renders the view template from model data, and updates this.el with the new HTML. |
| remove | view.remove() | Convenience function for removing the view from the DOM. |
| make | view.make(tagName, [attributes], [content]) | Convenience function for creating a DOM element of the given type (tagName). |
| delegateEvents | delegateEvents([events]) | Uses jQuery's delegate function to provide declarative callbacks for DOM events within a view. |
| undelegateEvents | undelegateEvents() | Removes all of the view's delegated events. |

| Utility | | |
|----------------------|---------------------------------------|--|
| noConflict | var backbone = Backbone.noConflict(); | Returns the Backbone object back to its original value. |
| setDomLibrary | Backbone.setDomLibrary(jQueryNew); | Tell Backbone to use a particular object as it's DOM / Ajax library. |