can.Construct

`can.Construct.extend( [name,] [staticProps,] instanceProps )`
Extends can.Construct, or constructor functions derived from can.Construct, to create a new constructor function.

`new can.Construct( [args..] )`
Create a new instance of a constructor function.

`can.Construct( [args...] )`
Create a new instance of a constructor function if constructorExtends is false.

`construct.init( ...args )`
Called when a new instance of a can.Construct is created.

`construct.setup( ...args )`
A setup function for the instantiation of a constructor function.

`can.Construct.newInstance( [...args] )`
Returns an instance of can.Construct.

`can.Construct.setup( base, fullName, staticProps, protoProps )`
A static setup method provides inheritable setup functionality for a Constructor function.

can.Control

`can.Control.extend( [staticProps,] instanceProps )`
Create a new, extended, control constructor function.

`new can.Control( element, options )`
Create an instance of a control.

`control.destroy()`
Prepares a control for garbage collection and is a place to reset any changes the control has made.

`control.on( [el,] selector, eventName, func )`
Bind an event handler to a Control, or rebind all event handlers on a Control.

`control.on()`
Rebind all of a control's event handlers.

`control.setup( element, options )`
Perform pre-initialization logic.

`"[selector] eventName": handler( element, event[, args… ] )`
Listen for an event on a control.

can.route

`can.route( template [, defaults] )`
Create a route matching rule.

`can.route.current( data )`
Check if data represents the current route.

`can.route.deparam( url )`
Extract data from a route path.

`can.route.link( innerText, data, props [,merge] )`
Make an anchor tag (<A>) that when clicked on will update can.route's properties to match those in data.

`can.route.param( data )`
Get a route path from given data.

`can.route.ready( readyYet )`
Pause and resume the initialization of can.route.

`can.route.url( data [, merge] )`
Make a URL fragment that when set to window.location.hash will update can.route's properties to match those in data.

can.Observe

`can.Observe.extend( [staticProps,] instanceProps )`
Create a new, extended, observe constructor function.

`can.Observe.keys( observe )`
Iterate over the keys of an Observe.

`can.Observe.startBatch( [batchStopHandler] )`
Begin an event batch.

`can.Observe.stopBatch( [force[, callStart]] )`
End an event batch.

`can.Observe.triggerBatch( item, event [, args] )`
Dispatch an event on an item immediately if there is no batch or after stopBatch is called.

`new can.Observe( [props] )`
Create an instance of an observe.

```
observe.attr()
observe.attr( key )
observe.attr( key, value )
observe.attr( obj[, removeOthers] )
```
Gets or sets a single property or multiple properties.

`observe.bind( eventType, handler )`
Bind event handlers to an Observe.

`observe.compute( attrName )`
Make a can.compute from an observable property.

`observe.each( callback( item, propName ) )`
each iterates through the Observe, calling a function for each property value and key.

`observe.removeAttr( attrName )`
Remove a property from an Observe.

`observe.serialize()`
Get the serialized Object form of the observe.

`observe.unbind( eventType[, handler] )`
Unbind event handlers from an Observe.

can.Observe.List

`can.Observe.List.extend( [staticProps,] instanceProps )`
Create a new, extended, observeable list constructor function.

`new can.Observe.List( [array] )`
Create an observable array-like object.

```
list.attr()
list.attr( index )
list.attr( index, value )
list.attr( elements[, replaceCompletely] )
```
Gets or sets a value or values at an index or indexes.

`list.replace( collection )`
Replace all the elements of a List.

`list array functions`
can.Observe.List implements the following functions that behave same as the native array equivalents: concat, join, forEach, indexOf, pop, push, reverse, shift, slice, splice, unshift.

can.Model

`can.Model.extend( [staticProps,] instanceProps )`
Create a new, extended, model constructor function.

`can.Model.bind( eventType, handler )`
Listen for events on a Model class.

```
can.Model.create: function( serialized ) -> deferred
can.Model.create: "[METHOD] /path/to/resource"
can.Model.create: { ajaxSettings }
```
Specify a function, HTTP method and url or options object used to create persistent instances.

```
can.Model.destroy: function( id ) -> deferred
can.Model.destroy: "[METHOD] /path/to/resource"
```
Provide a function or URL. Function should implement AJAX request and if URL is provided, Model will send a request to that URL.

```
can.Model.findAll( params[, success[, error]] )
can.Model.findAll: findAllData( params ) -> deferred
can.Model.findAll: "[METHOD] /path/to/resource"
can.Model.findAll: { ajaxSettings }
```
Retrieve multiple resources from a server. It can be implemented with a HTTP method and url, function or with a AJAX settings object.

```
can.Model.findOne( params[, success[, error]] )
can.Model.findOne: findOneData( params ) -> deferred
can.Model.findOne: "[METHOD] /path/to/resource"
can.Model.findOne: { ajaxSettings }
```
Retrieve a single instance from the server. It can be implemented with a HTTP method and url, function or with a AJAX settings object.

`can.Model.model( data )`
Convert raw data into a can.Model instance.

`can.Model.models( data[, oldList] )`
Convert raw data into can.Model instances.

`can.Model.unbind( eventType, handler )`
Stop listening for events on a Model class.

```
can.Model.update: "[METHOD] /path/to/resource"
can.Model.update: function( id, serialized ) -> can.Deffered
```
Provide a function or URL. Function should implement AJAX request and if URL is provided, Model will send a request to that URL.

`new can.Model( [options] )`
Create an instance of a model.

`model.bind( eventName, handler )`
Listen to events on this Model.

`model.destroy( [success[, error]] )`
Destroy a Model on the server.

`model.isNew()`
Check if a Model has yet to be saved on the server.

`model.save( [success[, error]] )`
Save a model back to the server.

`model.unbind( eventName[, handler] )`
Stop listening to events on this Model.

can.compute

`can.compute( getterSetter[, context] )`
Create a compute that derives its value from can.Observes and other can.computes.

`can.compute( initialValue [, settings] )`
Creates a compute from a value and optionally specifies how to read, update, and listen to changes in dependent values.

`can.compute( initialValue, setter( newVal,oldVal ) )`
Create a compute that has a setter that can adjust incoming new values.

`can.compute( object, propertyName [, eventName] )`
Create a compute from an object's property value.

`compute.bind( eventType, handler )`
Bind an event handler to a compute.

`compute.unbind( eventType[, handler] )`
Unbind an event handler from a compute.

can.view

`can.view( idOrUrl, data[, helpers] )`
Loads a template, renders it with data and helper functions and returns the HTML of the template within a documentFragment .

`can.view( idOrUrl )`
Registers or loads a template and returns a renderer function that can be used to render the template with data and helpers.

`can.view.ejs( [id,] template )`
Register an EJS template string and create a renderer function.

`can.view.mustache( [id,] template )`
Register a Mustache template string and create a renderer function.

Mustache

`Mustache.registerHelper( name, helper )`
Register a helper.

`{{key}}`
Insert the value of the key into the output of the template.

`{{{key}}}`
Behaves just like {{key}} and {{helper}} but does not escape the result.

`{{#key}}BLOCK{{/key}}`
Render blocks of text one or more times, depending on the value of the key in the current context.

`{{/key}}`
Ends a {{#key}} or {{#helper}} block.

`{{^key}}BLOCK{{/key}}`
Render blocks of text if the value of the key is falsey.

`{{helper [args...] [hashKey=hashValue...]}}`
Calls a mustache helper function or a function.

`{{#helper [args...] [hashKey=hashVal...]}}BLOCK{{/helper}}`
Calls a mustache helper function or a function with a block to render.

`{{#helper [args...] [hashKey=hashVal...]}}BLOCK{{else}}INVERSE{{/helper}}`
Calls a mustache helper function or a function with a fn and inverse block to render.

`{{#if key}}BLOCK{{/if}}`
Renders the BLOCK template within the current template.

`{{#helper}}BLOCK{{else}}INVERSE{{/helper}}`
Creates an inverse block for a helper function 's options argument 's inverse property.

`{{#each key}}BLOCK{{/each}}`
Render the block of text for each item in key's value.

`{{#unless key}}BLOCK{{/unless}}`
Render the block of text if the key's value is falsey.

`{{#with key}}BLOCK{{/with}}`
Changes the context within a block.

`{{data name}}`
Adds the current context to the element's can.data .

`{{>key}}`
Render another template within the current template.

EJS

`<% CODE %>`
Runs JavaScript Code.

`<%= CODE %>`
Runs JS Code and writes the escaped result into the result of the template.

`<%== CODE %>`
Runs JS Code and writes the unescaped result into the result of the template.

`<%# CODE %>`
Used for comments.