BACKBONE.JS

javascript library

tutorialspoint

SIMPLYEASYLEARNING

www.tutorialspoint.com

https://www.facebook.com/tutorialspointindia

https://twitter.com/tutorialspoint

About the Tutorial

BackboneJS is a light weight JavaScript library that allows to develop and structure client side applications that run in a web browser. It offers MVC framework which abstracts data into models, DOM (Document Object Model) into views and bind these two using events.

This tutorial covers most of the topics required for a basic understanding of BackboneJS and to get a feel of how it works.

Audience

This tutorial is designed for software programmers who want to learn the basics of BackboneJS and its programming concepts in simple and easy ways. This tutorial will give you enough understanding on various components of BackboneJS with suitable examples.

Prerequisites

Before proceeding with this tutorial, you should have a basic understanding of HTML, CSS, JavaScript, and Document Object Model (DOM). As we are going to develop a web-based applications using BackboneJS, it will be good if you have an understanding of how web-based applications work in general.

Copyright & Disclaimer

Table of Contents

1. BackboneJS – Overview

BackboneJS is a **lightweight JavaScript library** that allows to develop and structure the client side applications that run in a web browser. It offers MVC framework which abstracts data into models, DOM into views and bind these two using events.

History: BackboneJS was developed by Jeremy Ashkenas and was initially released on October 13th, 2010.

When to use Backbone

- Consider you are creating an application with numerous lines of code using JavaScript or jQuery. In this application, if you:

    - add or replace DOM elements to the application or
    - make some requests or
    - show animation in the application or
    - add more number of lines to your code,

  then your application might become complicated.

- If you want a better design with less code, then it is better to use the BackboneJS library that provides good functionality, is well organized and in a structured manner for developing your application.

- BackboneJS communicates via events; this ensures that you do not mess up the application. Your code will be cleaner, nicer and easy to maintain.

Features

The following are a list of features of BackboneJS:

- BackboneJS allows developing of applications and the frontend in a much easier way by using JavaScript functions.
- BackboneJS provides various building blocks such as models, views, events, routers and collections for assembling the client side web applications.
- When a model changes, it automatically updates the HTML of your application.
- BackboneJS is a simple library that helps in separating business and user interface logic.
- It is free and open source library and contains over 100 available extensions.
- It acts like a backbone for your project and helps to organize your code.
- It manages the data model which includes the user data and displays that data at the server side with the same format written at the client side.
- BackboneJS has a soft dependency with **jQuery** and a hard dependency with **Underscore.js**.

- It allows to create client side web applications or mobile applications in a well-structured and an organized format.

2. BackboneJS – Environment Setup

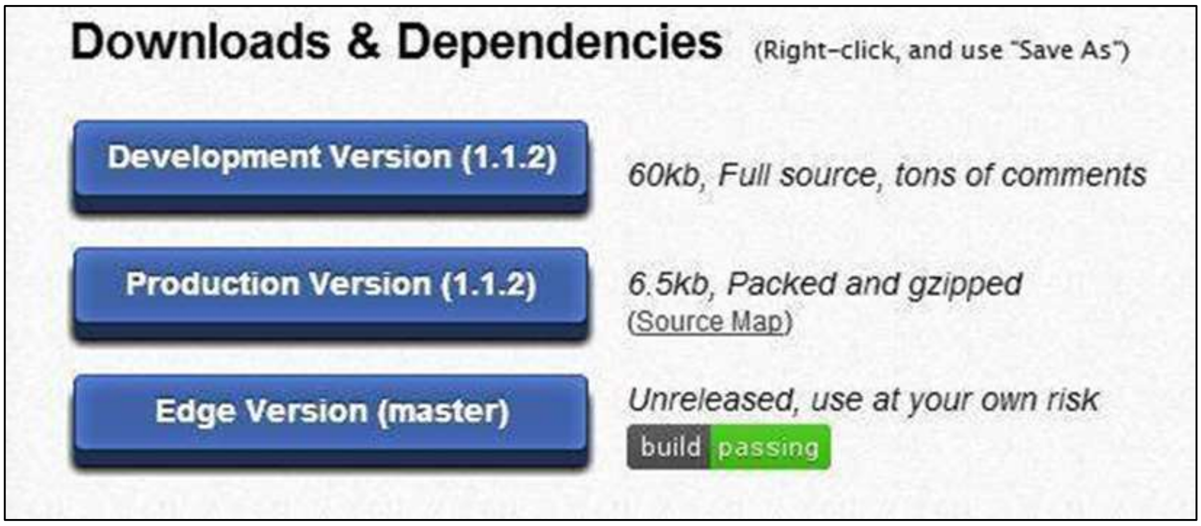
BackboneJS is very easy to setup and work. This chapter will discuss the download and setup of the **BackboneJS Library**.

BackboneJS can be used in the following two ways:

- Downloading UI library from its official website.
- Downloading UI library from CDNs

Downloading the UI library from its official website

When you open the link http://backbonejs.org/, you will get to see a screenshot as shown below:


Downloads & Dependencies (Right-click, and use "Save As")
Development Version (1.1.2)
60kb, Full source, tons of comments
Production Version (1.1.2)
6.5kb, Packed and gzipped
(Source Map)
Edge Version (master)
Unreleased, use at your own risk
build passing

As you can see, there are three options for download of this library:

- **Development Version** - Right click on this button and save as and you get the full source **JavaScript library**.
- **Production Version** - Right click on this button and save as and you get the **Backbone-min.js library** file which is packed and gzipped.
- **Edge Version** - Right click on this button and save as and you get an **unreleased version**, i.e., development is going on; hence you need to use it at your own risk.

Dependencies

BackboneJS depends on the following JavaScript files:

- **Underscore.js:** This is the only hard dependency which needs to be included. You can get it from here.
- **jQuery.js:** Include this file for RESTful persistence, history support via Backbone.Router and DOM manipulation with Backbone.View. You can get it from here.

- **json2.js:** Include this file for older Internet Explorer support. You can get it from here.

Download UI Library from CDNs

A CDN or **Content Delivery Network** is a network of servers designed to serve files to users. If you use a CDN link in your web page, it moves the responsibility of hosting files from your own servers to a series of external ones. This also offers an advantage that if the visitor to your webpage has already downloaded a copy of BackboneJS from the same CDN, it won't have to be re-downloaded.

As said above, BackboneJS has a dependency of the following JavaScript:

- jQuery
- Underscore

Hence CDN for all the above is as follows:

```
<script type="text/javascript"
src="https://ajax.googleapis.com/ajax/libs/jquery/1.5.2/jquery.min.js"></scrip
t>

<script type="text/javascript"
src="https://ajax.cdnjs.com/ajax/libs/underscore.js/1.1.4/underscore-
min.js"></script>

<script type="text/javascript"
src="https://ajax.cdnjs.com/ajax/libs/backbone.js/0.3.3/backbone-
min.js"></script>
```

Note: We are using the CDN versions of the library throughout this tutorial.

Example

Let's create a simple example using BackboneJS.

```
<!DOCTYPE html>

<html>

<head>

  <meta charset="UTF-8">

  <meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1">

  <title>Hello World using Backbone.js</title>

</head>

<body>

  <!-- ========= -->

  <!-- Your HTML -->

  <!-- ========= -->

  <div id="container">Loading...</div>
  <!-- ========= -->


  <!-- Libraries -->

  <!-- ========= -->

  <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>

  <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.3.3/underscore-
min.js" type="text/javascript"></script>

  <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/0.9.2/backbone-min.js"
type="text/javascript"></script>




  <!-- =============== -->

  <!-- Javascript code -->

  <!-- =============== -->

  <script type="text/javascript">

    var AppView = Backbone.View.extend({

      // el - stands for element. Every view has an element associated with
HTML content, will be rendered.

      el: '#container',

      // It's the first function called when this view is instantiated.

      initialize: function(){

        this.render();

      },

      // $el - it's a cached jQuery object (el), in which you can use jQuery
functions to push content. Like the Hello TutorialsPoint in this case.

      render: function(){

        this.$el.html("Hello TutorialsPoint!!!");

      }

    });


    var appView = new AppView();

  </script>

</body>

</html>
```

The code comments are self-explanatory. A few more details are given below:

There's a html code at the start of *body* tag

```
<div id="container">Loading...</div>
```

This prints ***Loading...***

Next, we have added the following CDNs

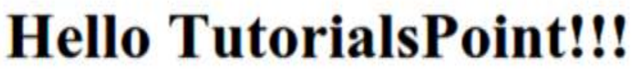
```
<script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>

  <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.3.3/underscore-
min.js" type="text/javascript"></script>

  <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/0.9.2/backbone-min.js"
type="text/javascript"></script>
```

Next, we have the following script:

```
var AppView = Backbone.View.extend({

      // el - stands for element. Every view has an element associated with
HTML content, will be rendered.

      el: '#container',

      // It's the first function called when this view is instantiated.

      initialize: function(){

        this.render();

      },

      // $el - it's a cached jQuery object (el), in which you can use jQuery
functions to push content. Like the Hello World in this case.

      render: function(){

        this.$el.html("<h1>Hello TutorialsPoint!!!</h1>");

      }

    });


    var appView = new AppView();
```

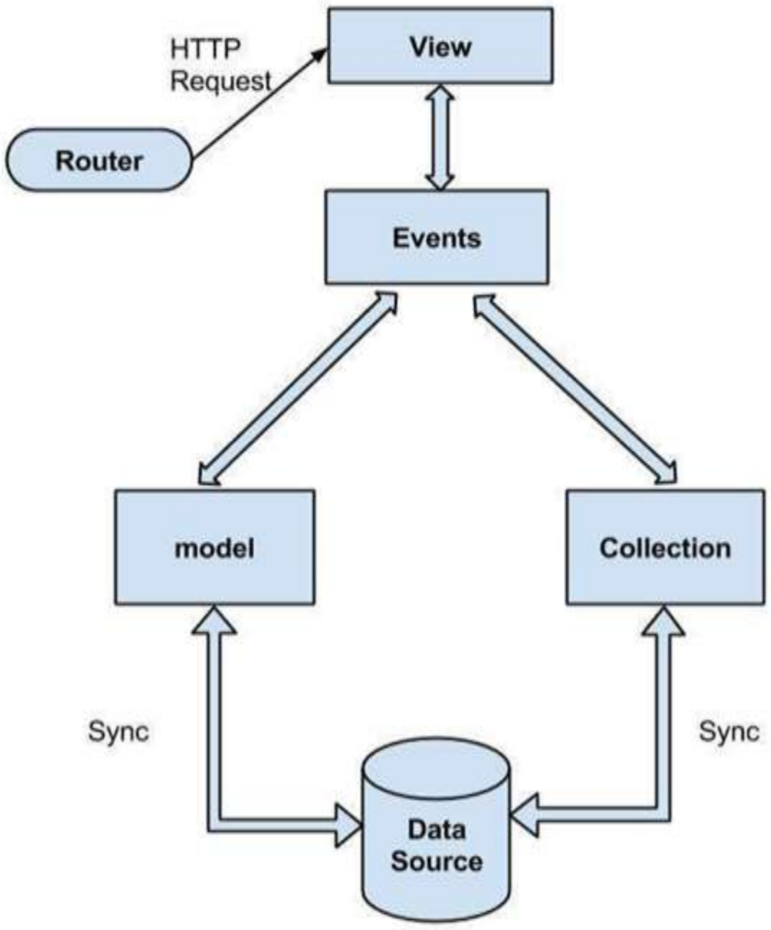
The comments are self-explanatory. In the last line, we are initializing ***new AppView()***. This will print the "Hello TutorialsPoint" in the **div** with **id="container"**

Save this page as **myFirstExample.html**. Open this in your browser and the screen will show the following text.


Hello TutorialsPoint!!!

3. BackboneJS – Applications

The BackboneJS gives a structure to the web applications that allows to separate business logic and user interface logic. In this chapter, we are going to discuss the architectural style of the BackboneJS application for implementing user interfaces. The following diagram shows the architecture of BackboneJS :



The architecture of BackboneJS contains the following modules:

- HTTP Request
- Router
- View
- Events
- Model
- Collection
- Data Source

Let us now discuss all the modules in detail.

HTTP Request

The HTTP client sends a HTTP request to a server in the form of a request message where web browsers, search engines, etc., acts like HTTP clients. The user requests for a file such as documents, images, etc., using the HTTP request protocol. In the above diagram, you could see that the HTTP client uses the router to send the client request.

Router

It is used for routing the client side applications and connects them to actions and events using URL's. It is a URL representation of the application's objects. This URL is changed manually by the user. The URL is used by the backbone so that it can understand what application state to be sent or present to the user.

The router is a mechanism which can copy the URL's to reach the view. The Router is required when web applications provide linkable, bookmarkable, and shareable URL's for important locations in the app.

In the above architecture, the router sending an HTTP request to the View. It is a useful feature when an application needs routing capability.

View

BackboneJS views are responsible for how and what to display from our application and they don't contain HTML markup for the application. It specifies an idea behind the presentation of the model's data to the user. Views are used to reflect "how your data model looks like".

The view classes do not know anything about the HTML and CSS and each view can be updated independently when the model changes without reloading the whole page. It represents the logical chunk of the UI in the DOM.

As shown in the above architecture, the View represents the user interface which is responsible for displaying the response for the user request done by using the Router.

Events

Events are the main parts of any application. It binds the user's custom events to an application. They can be mixed into any object and are capable of binding and triggering custom events. You can bind the custom events by using the desired name of your choice.

Typically, events are handled synchronously with their program flow. In the above architecture, you could see when an event occurs, it represents the model's data by using the View.

Model

It is the heart of the JavaScript application that retrieves and populates the data. Models contain data of an application, logic of the data and represents the basic data object in the framework.

Models represents business entities with some business logic and business validations. They are mainly used for data storage and business logic. Models can be retrieved from and saved to data storage. A Model takes the HTTP request from the Events passed by the View using the Router and synchronizes the data from the database and sends the response back to the client.

Collection

A Collection is a set of models which binds events, when the model has been modified in the collection. The collection contains a list of models that can be processed in the loop and supports sorting and filtering. When creating a collection, we can define what type of model that collection is going to have along with the instance of properties. Any event triggered on a model will also trigger on the collection in the model.

It also takes the request from the view, bind events and synchronizes the data with the requested data and sends the response back to the HTTP client.

Data Source

It is the connection set up to a database from a server and contains the information which is requested from the client. The flow of the BackboneJS architecture can be described as shown in the following steps:

- A User requests for the data using the router, which routes the applications to the events using the URL's.
- The view represents the model's data to the user.
- The model and collection retrieves and populates the data from the database by binding custom events.

In the next chapter, we will understand the significance of Events in BackboneJS.

4. BackboneJS – Events

Events are capable of binding objects and trigger custom events i.e. you can bind the custom events by using the desired name of our choice.

The following table lists down all the methods which you can use to manipulate the BackboneJS-Events:

S.No.	Methods & Description
1	**on** It binds an event to an object and executes the callback whenever an event is fired.
2	**off** It removes callback functions or all events from an object.
3	**trigger** It invokes the callback functions for the given events.
4	**once** It extends the backbone.Model class while creating your own backbone Model.
5	**listenTo** It informs one object to listen to an event on another object.
6	**stopListening** It can be used to stop listening to events on the other objects.
7	**listenToOnce** It causes the listenTo occur only once before the callback function is being removed.

BackboneJS –Event On

Description

It binds an event to an object and the callback function. Whenever an event is fired, it executes the callback.

Syntax

```
object.on(event, callback function, [context])
```

Parameters

- **event:** It binds an object.
- **callback:** It is the reference to the code.

- **context:** It is an object that can be passed to a callback function.

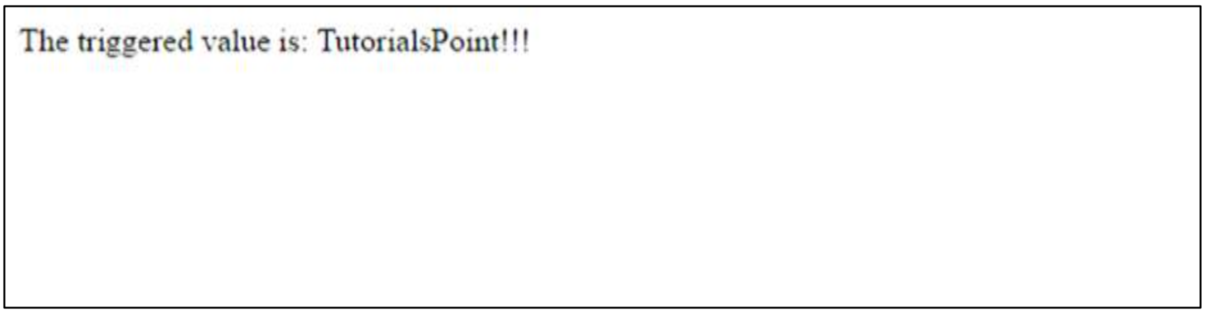
Example

```
<!DOCTYPE html>
   <head>
      <title>Event On Example</title>
         <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js"
type="text/javascript"></script>
   </head>
   <body>
      <script type="text/javascript">
            //Here creating an object 'myVal' and extending with
Backbone.Events method
         var myVal = _.extend({name:'TutorialsPoint!!!'}, Backbone.Events);


         // The on() method will bind callback function to an object and
invoked whenever an event triggers
         myVal.on('myFunc', function () {
            document.write("The triggered value is: ");
            document.write(this.name);//The name will get display by referring
the current object
         });
         //It triggers the 'myFunc' event on object 'myVal'
         myVal.trigger('myFunc');
      </script>
   </body>
</html>
```

Output

Let us carry out the following steps to see how the above code works:

- Save the above code in **on.htm** file
- Open this HTML file in a browser.



```
The triggered value is: TutorialsPoint!!!
```

BackboneJS – Event Off

Description

This event removes the callback functions or all events from an object.

Syntax

```
object.off(event, callback function, [context])
```

Parameters

- **event:** It binds an object.
- **callback:** It is the reference to the code.
- **context:** It is an object that can be passed to a callback function.

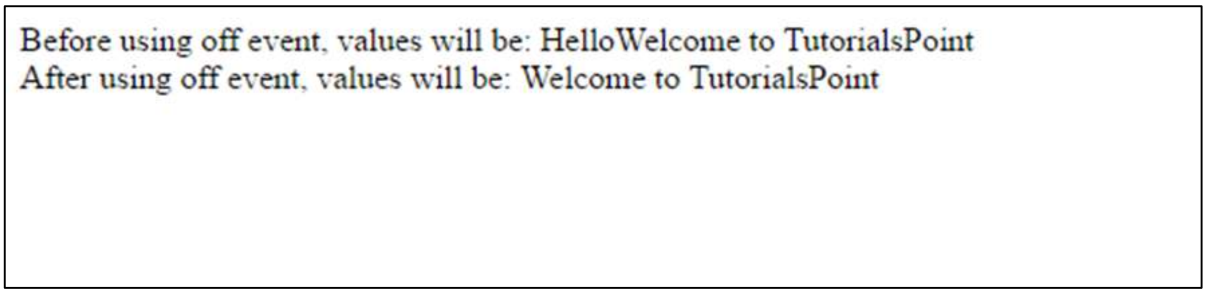
Example

```
<!DOCTYPE html>
  <head>
    <title>Event Off Example</title>
      <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>
      <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>
      <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js"
type="text/javascript"></script>
  </head>
  <body>
     <script type="text/javascript">

        //Here creating an object 'myVal' and extending with Backbone.Events method
        var myVal = _.extend({name:'hello'}, Backbone.Events);
        var myFunc = function () {
           document.write('Hello');
        };
        var myFunc1 = function () {
           document.write('Welcome to TutorialsPoint');
        };

        //The on() method will bind the callback function to objects 'myFunc'
and 'myFunc1'

        myVal.on('log',myFunc);
        myVal.on('log',myFunc1);
        document.write('Before using off event, values will be: ');

        //trigger() method callbacks for the given event and display the text
defined in the 'myFunc' and 'myFunc1' functions
        myVal.trigger('log');

        //The off() method removes the callback for 'myFunc' and logs only
text of 'myFunc1'
        myVal.off('log',myFunc);

        document.write("<br>");
        document.write('After using off event, values will be: ');
        myVal.trigger('log');
     </script>
   </body>
</html>
```

Output

Let us carry out the following steps to see how above code works:

- Save the above code in **off.htm** file
- Open this HTML file in a browser.



```
Before using off event, values will be: HelloWelcome to TutorialsPoint
After using off event, values will be: Welcome to TutorialsPoint
```

BackboneJS – Event Trigger

Description

It invokes the callback functions for the given events.

Syntax

```
object.trigger(event,[args])
```

Parameters

- **event:** It binds an object.
- **args:** It passes the values/arguments to the callback function.

Example

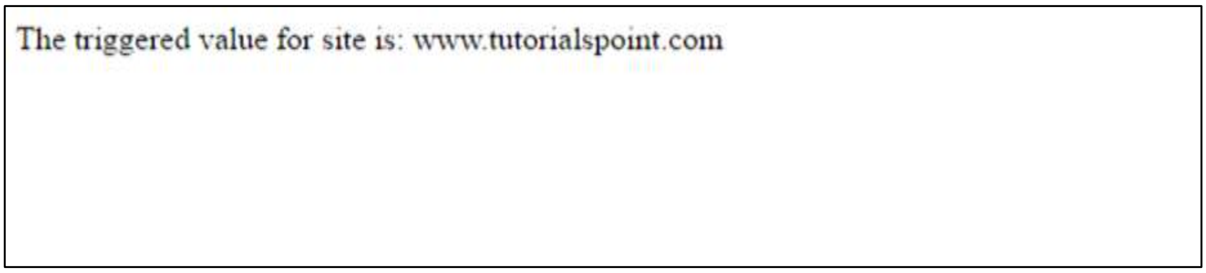
```
<!DOCTYPE html>
  <head>
    <title>Event Trigger Example</title>
      <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>
      <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>
      <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js"
type="text/javascript"></script>
  </head>
  <body>
     <script type="text/javascript">
     //Created an object 'myVal' and extended it using Backbone.Events method
        var myVal = _.extend({title:'TutorialsPoint!!!',
site:'www.tutorialspoint.com'}, Backbone.Events);
```

```
        //The on() method will bind the callback function to an object
        myVal.on('myFunc', function () {
           document.write("The triggered value for site is: ");
           document.write(this.site); //value of site will get displayed by
referring the current object
        });

        // The trigger() method triggers the 'myFunc' event on 'myVal'
        myVal.trigger('myFunc');
     </script>
   </body>
</html>
```

Output

Let us carry out the following steps to see how the above code works:

- Save the above code in the **trigger.htm** file
- Open this HTML file in a browser.



```
The triggered value for site is: www.tutorialspoint.com
```

BackboneJS – Event Once

Description

It is just like an **on** event, but causes the bound callback to only fire once before being removed.

Syntax

```
object.once(event, callback function, [context])
```

Parameters

- **event:** It binds an object.
- **callback:** It is reference to the code.
- **context:** It is an object that can be passed to a callback function.

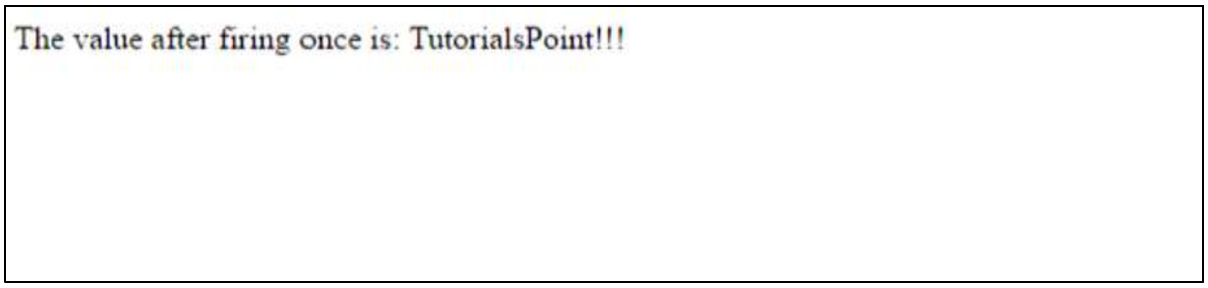
Example

```
<!DOCTYPE html>
  <head>
    <title>Event Once Example</title>
      <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>
      <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>
      <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js"
type="text/javascript"></script>
  </head>
  <body>
     <script type="text/javascript">

     //The created object 'myVal' is extended using Backbone.Events method
        var myVal = _.extend({name:'TutorialsPoint!!!'}, Backbone.Events);

        //The once() method causes the bound callback to only fire once before
being removed
        myVal.once('hello', function () {
           document.write("The value after firing once is: ");
           document.write(this.name);//name will get displayed by referring
the current object
        });

        //It triggers the 'hello' event on object 'myVal'
        myVal.trigger('hello');
     </script>
  </body>
</html>
Output
```

Output

Let us carry out the following steps to see how the above code works:

- Save the above code in **once.htm** file
- Open this HTML file in a browser.



```
The value after firing once is: TutorialsPoint!!!
```

BackboneJS – Event listenTo

Description

It tells an object to listen to an event on another object. It keeps track of events and provides callback function when an event occurs.

Syntax

```
object.listenTo(other, event, callback)
```

Parameters

- **other:** It defines name of the other object.
- **event:** It binds an object.
- **callback:** It is reference to the code.

Example

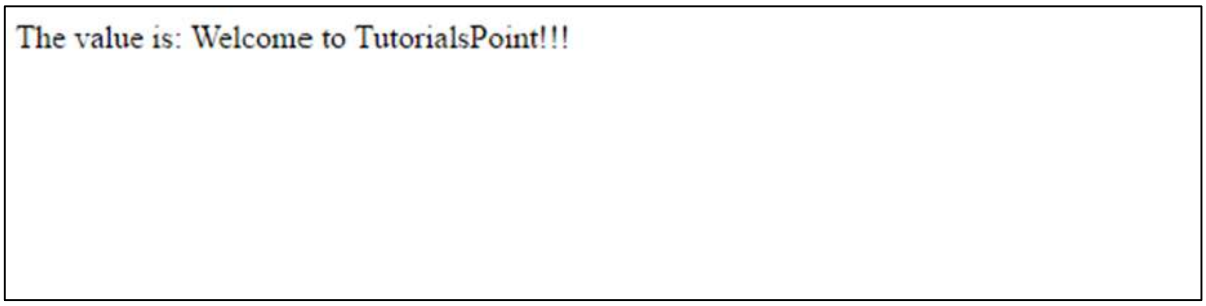
```
<!DOCTYPE html>
  <head>
    <title>Event Once Example</title>
      <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>
      <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>
      <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js"
type="text/javascript"></script>
  </head>
  <body>
     <script type="text/javascript">
     //Create an object 'myVal' and 'myVal1' and extend them using
Backbone.Events method
```

```
         var myVal = _.extend({name:'Hello..'}, Backbone.Events);

         var myVal1 = _.extend({name:'Welcome to TutorialsPoint!!!'},
Backbone.Events);


        //create the 'listenMe' callback function and invoke when one object
listens to particular event on another object
         var listenMe = function(){
            document.write("The value is: ");
            document.write(this.name);
         };
       //The object 'myVal1' listens once for the 'listenMe' event triggered on
object 'myVal'
        myVal1.listenTo(myVal, 'listenMe', listenMe);


        //The 'myVal' has no listenMe event and displays the value of 'myVal1'
        myVal.trigger('listenMe');
      </script>
   </body>
</html>
```

Output

Let us carry out the following steps to see how the above code works:

- Save the above code in **listento.htm** file
- Open this HTML file in a browser.



```
The value is: Welcome to TutorialsPoint!!!
```

BackboneJS – Event stopListening

Description

As its name specifies, it can be used to stop listening to events on the other objects.

Syntax

```
object.stopListening(other, event, callback)
```

Parameters

- **other:** It defines name of the other object.
- **event:** It binds an object.
- **callback:** It is reference to the code and called with object as context.

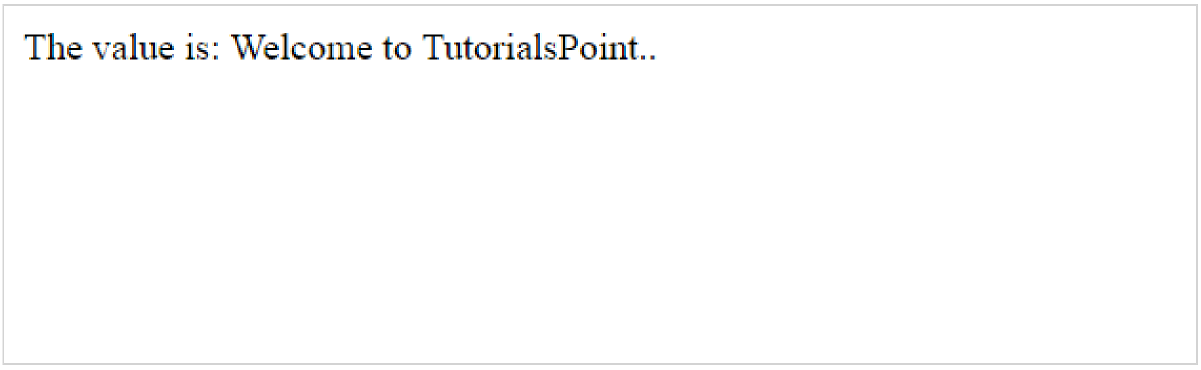
Example

```
<!DOCTYPE html>
  <head>
    <title>Event Once Example</title>
      <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>
      <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>
      <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js"
type="text/javascript"></script>
  </head>
  <body>
     <script type="text/javascript">

     //Create an object 'myVal' and 'myVal1' and extend them using
Backbone.Events method
        var myVal = _.extend({name:'Hello..'}, Backbone.Events);
        var myVal1 = _.extend({name:'Welcome to TutorialsPoint..'},
Backbone.Events);

        //created the 'listenMe' callback function and invoked when one object
listens to particular event on another object
        var listenMe = function(){
           document.write("The value is: ");
           document.write(this.name);
        };

        //The object 'myVal1' listens once for the 'listenMe' event triggered
on object 'myVal'
        myVal1.listenTo(myVal, 'listenMe', listenMe);
        //The 'myVal' has no 'listenMe' event and display the value of
'myVal1'
        myVal.trigger('listenMe');


        //The 'myVal1' stops listening to specific event on 'myVal' and
displays nothing
        myVal1.stopListening(myVal,'listenMe');
        myVal.trigger('listenMe');
     </script>
  </body>
</html>
```

Output

Let us carry out the following steps to see how the above code works:

- Save the above code in **stoplistening.htm** file
- Open this HTML file in a browser.



```
The value is: Welcome to TutorialsPoint..
```

BackboneJS – Event listenToOnce

Description

It is the same like the **listenTo** event, but causes the listento to occur only once before the callback function is being removed.

Syntax

```
object.listenToOnce(other, event, callback)
```

Parameters

- **other:** It defines name of the other object.
- **event:** It binds an object.
- **callback:** It is reference to the code and called with object as context.

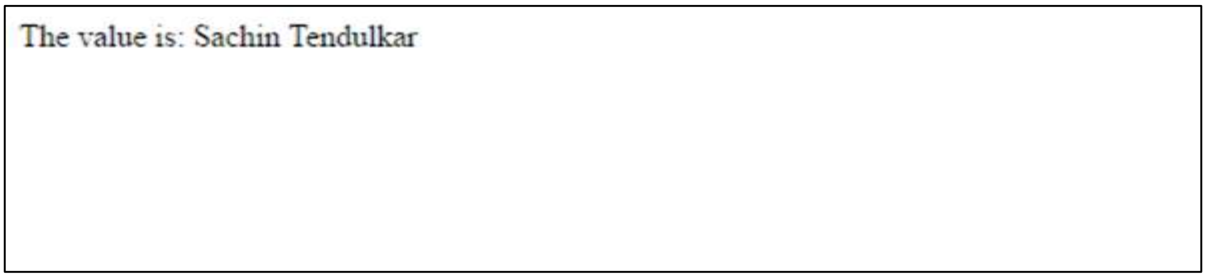
Example

```
<!DOCTYPE html>
  <head>
    <title>Event Once Example</title>
      <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>
      <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>
      <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js"
type="text/javascript"></script>
  </head>
  <body>
     <script type="text/javascript">
     //Create an object 'myVal' and 'myVal1' and extend them using
Backbone.Events method
        var myVal = _.extend({name:'Saurav Ganguly'}, Backbone.Events);
        var myVal1 = _.extend({name:'Sachin Tendulkar'}, Backbone.Events);

        //created the 'listenMe' callback function and invoked when one object
listen to particular event on another object
        var listenMe = function(){
           document.write("The value is: ");
           document.write(this.name);
        };

        //The object 'myVal1' listen once for the 'listenMe' event triggered
on object 'myVal'
        myVal1.listenToOnce(myVal, 'listenMe', listenMe);

        //The 'myVal' has no listenMe event and display the value of 'myVal1'
        myVal.trigger('listenMe');
     </script>
  </body>
</html>
```

Output

Let us carry out the following steps to see how the above code works:

- Save the above code in **listentoonce.htm** file.
- Open this HTML file in a browser.



```
The value is: Sachin Tendulkar
```

Catalog of Built-in Events

BackboneJS allows the use of global events wherever necessary in your application. It contains some of the built-in events with arguments as shown in the following table:

S.N.	Events & Description
1	**"add"(model, collection, options)** It used when a model is added to the collection.
2	**"remove"(model, collection, options)** It removes a model from the collection.
3	**"reset"(collection, options)** It is used to reset the collection content.
4	**"sort"(collection, options)** It is used when a collection needs to resorted.
5	**"change"(model, options)** It is used when changes are to be made to a model's attributes.
6	**"change:[attribute]"(model, value, options)** It is used when there is an update in an attribute.
7	**"destroy"(model, collection, options)** It fires when the model is destroyed.

8	**"request"(model_or_collection, xhr, options)** It is used when a model or a collection starts requesting to the server.
9	**"sync"(model_or_collection, resp, options)** It is used when a model or a collection is synced successfully with the server.
10	**"error"(model_or_collection, resp, options)** It activates when there is an error in requesting to the server.
11	**"invalid"(model, error, options)** When there is a fail in model validation, it returns invalid.
12	**"route:[name]"(params)** When there is a specific route match, this event can be used.
13	**"route"(route,params)** It is used when there is a match with any route.
14	**"route"(router, route, params)** It is used by history when there is a match with any route.
15	**"all"** It fires for all the triggered events by the passing event name as the first argument.

5. BackboneJS – Model

Models contain dynamic data and its logic. Logic such as conversions, validations, computed properties and access control fall under the Model category. As it contains all the application data, a model is also called as the **heart of JavaScript application**.

The following table lists down all the methods which you can use to manipulate the BackboneJS-Model:

S.No.	Methods & Description
1	**extend** It extends the **backbone.Model** class while creating your own backbone Model.
2	**initialize** When a model instance is created, the class's constructor gets called and it is invoked by defining the initialize function when the model is created.
3	**get** It gets the value of an attribute on the model.
4	**set** It sets the value of an attribute in the model.
5	**escape** It is like the **get** function, but returns the HTML-escaped version of a model's attribute.
6	**has** Returns true, if attribute value defined with non-null value or non-undefined value.
7	**unset** It removes an attribute from a backbone model.
8	**clear** Removes all attributes, including id attribute from a backbone model.
9	**id** It uniquely identifies the model entity, that might be manually set when a model is created or populated or when a model is saved on the server.
10	**idAttribute**

	Defines a model's unique identifier which contains the name of the member of the class which will be used as id.
11	**cid** It is an auto generated client id by Backbone which uniquely identifies the model on the client.
12	**Attributes** Attributes defines property of a model.
13	**changed** Changes all the attributes that have changed after setting the attributes using the **set()** method.
14	**defaults** Sets a default value to a model, that means if the user doesn't specify any data, the model won't fall with an empty property.
15	**toJSON** Returns a copy of the attributes as an object for JSON stringification.
16	**sync** It is used to communicate with the server and to represent the state of a model.
17	**fetch** Accept the data from the server by delegating **sync()** method in the model.
18	**save** Saves the data of the model by delegating to **sync()** method which reads and saves the model every time when a Backbone calls it.
19	**destroy** Destroys or removes the model from the server by using the **Backbone.sync** method which delegates the HTTP "delete" request.
20	**validate** If the input is invalid, it returns a specified error message or if the input is valid, it doesn't specify anything and simply displays the result.
21	**validationError** It displays the validation error, if the validation fails or after the **invalid** event is triggered.

22	**isValid** It checks the model state by using the **validate()** method and also checks validations for each attribute.
23	**url** It is used for the instance of the model and returns the url to where the model's resource is located.
24	**urlRoot** Enables the url function by using the model id to generate the URL.
25	**parse** Returns the model's data by passing through the response object and represents the data in the JSON format.
26	**clone** It is used to create a deep copy of a model or to copy one model object to another object.
27	**hasChanged** Returns true, if the attribute gets changed since the last **set**.
28	**isNew** Determines whether the model is a new or an existing one.
29	**changedAttributes** It returns the model's attributes that have changed since the last **set** or else becomes false, if there are no attributes.
30	**previous** It determines the previous value of the changed attribute.
31	**previousAttributes** Returns the state of the all the attributes prior to the last change event.

BackboneJS – Model Extend

Description

It is used to extend the **backbone.Model** class while creating your own backbone Model.

Syntax

```
Backbone.Model.extend(properties, [classProperties])
```

Parameters

- **properties:** It provides instance properties for the Model class.
- **classProperties:** The class properties are attached to the constructor function.

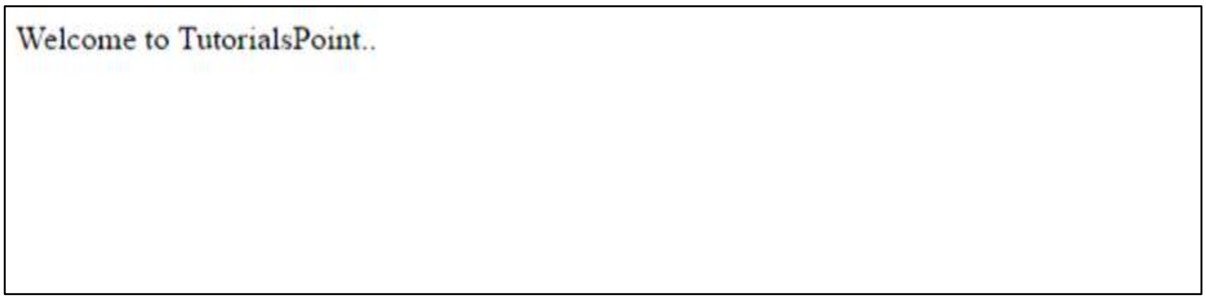
Example

```
<!DOCTYPE html>
   <head>
      <title> Model Example</title>
         <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js"
type="text/javascript"></script>
         <script type="text/javascript">
            MyModel = Backbone.Model.extend({
               initialize: function(){
                  document.write("Welcome to TutorialsPoint..");
               }
            });
            var mymodel = new MyModel;
         </script>
   </head>
   <body></body>
</html>
```

Output

Let us carry out the following steps to see how the above code works:

- Save the above code in the **extend.htm** file
- Open this HTML file in a browser.



Welcome to TutorialsPoint..

BackboneJS – Model Initialize

Description

When a model instance is created, the class's constructor gets called and it is invoked by defining the **initialize** function.

Syntax

```
new Model(attributes, options)
```

Parameters

- **attributes:** Attributes define properties of a model, when creating instance of that model.
- **options:** These are the options such as id, name, etc., used with attributes when a model is created.

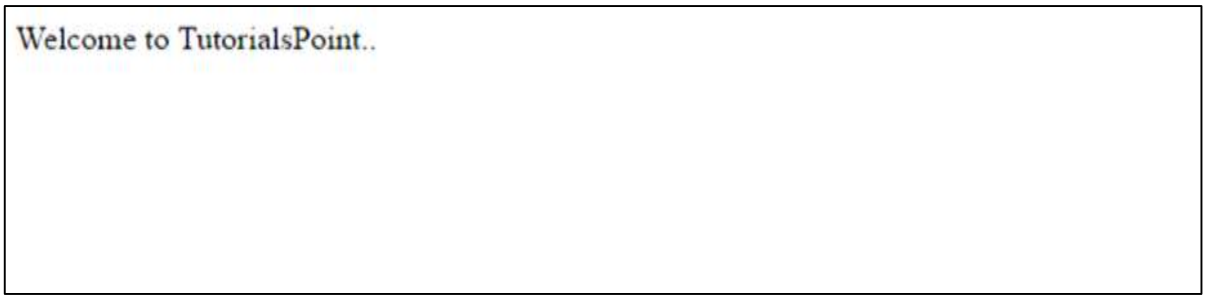
Example

```
<!DOCTYPE html>
   <head>
      <title> Model Example</title>
         <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js"
type="text/javascript"></script>
         <script type="text/javascript">
            MyModel = Backbone.Model.extend({
               initialize: function(){
                  document.write("Welcome to TutorialsPoint..");
               }
            });
            var mymodel = new MyModel;
         </script>
   </head>
   <body></body>
</html>
```

Output

Let us carry out the following steps to see how the above code works:

- Save the above code in the **initialize.htm** file.
- Open this HTML file in a browser.



```
Welcome to TutorialsPoint..
```

BackboneJS – Model Get

Description

It is used to get value of an attribute on the model.

Syntax

```
model.get(attribute)
```

Parameters

- **attribute:** Attribute defines the property of a created model.

Example

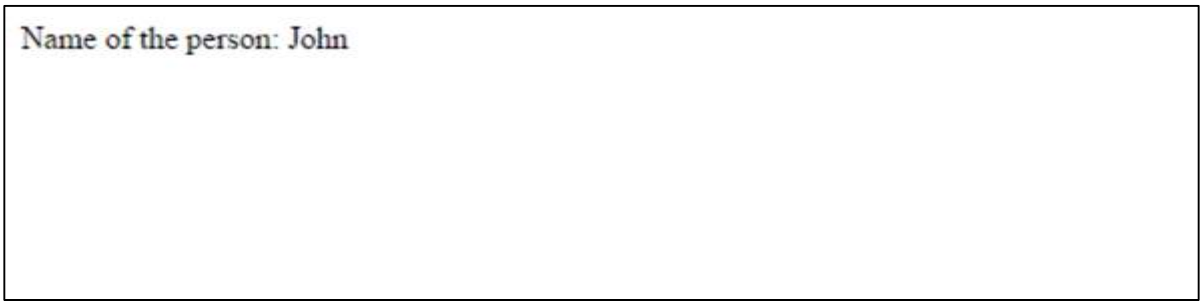
```
<!DOCTYPE html>
   <head>
      <title> Model Example</title>
         <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js"
type="text/javascript"></script>
         <script type="text/javascript">
```

```
            var  Person = Backbone.Model.extend();
            var person = new Person();
            person.set({ fname: "John", lname:"Smith"});
            document.write("Name of the person: ", person.get('fname'));
         </script>
   </head>
   <body></body>
</html>
```

Output

Let us carry out the following steps to see how the above code works:

- Save the above code in **get.htm** file.
- Open this HTML file in a browser.



```
Name of the person: John
```

BackboneJS – Model Set

Description

It is used to set the value of an attribute in the model.

Syntax

```
model.set(attribute)
```

Parameters

- **attribute:** An attribute defines the property of a created model.

Example

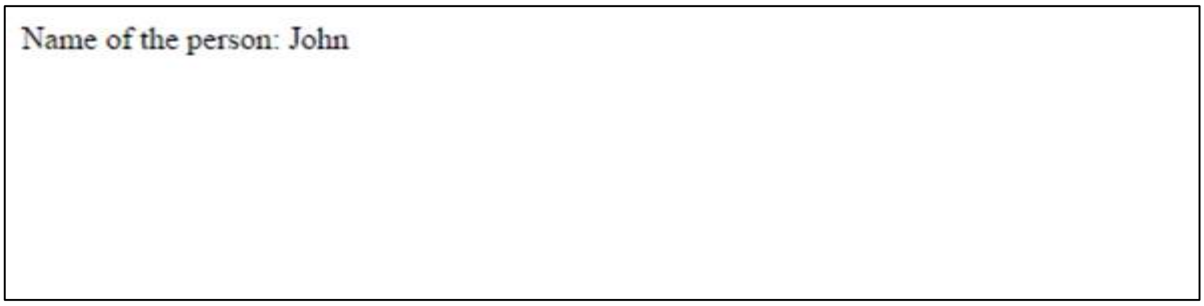
```
<!DOCTYPE html>
   <head>
      <title> Model Example</title>
         <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>
```

```
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>

         <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js"
type="text/javascript"></script>

         <script type="text/javascript">

            var  Person = Backbone.Model.extend();

            var person = new Person();


            person.set({ fname: "John", lname:"Smith"});

            document.write("Name of the person: ", person.get('fname'));

         </script>

   </head>

   <body></body>

</html>
```

Output

Let us carry out the following steps to see how the above code works:

- Save the above code in the **set.htm** file.
- Open this HTML file in a browser.



```
Name of the person: John
```

BackboneJS – Model Escape

Description

It is almost like the **get function**, but returns the HTML–escaped version of a model's attribute.

Syntax

```
model.escape(attribute)
```

Parameters

- **attribute:** An attribute defines the property of a created model.

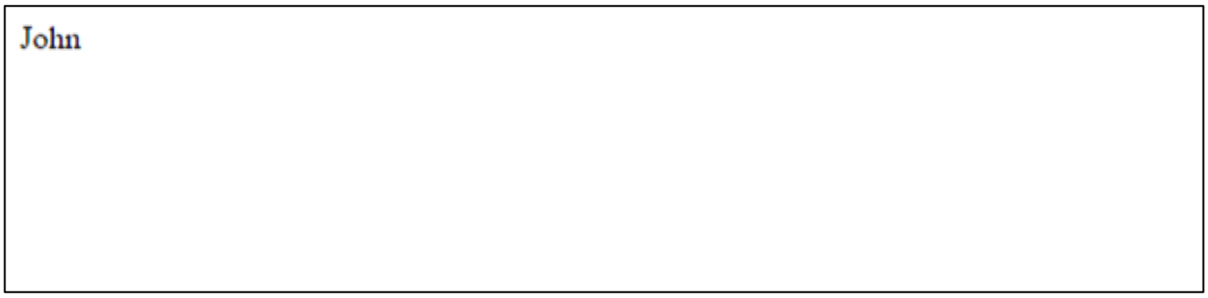
Example

```
<!DOCTYPE html>
   <head>
      <title> Model Example</title>
         <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>


         <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js"
type="text/javascript"></script>
         <script type="text/javascript">
            var  Person = Backbone.Model.extend();
            var person = new Person();
            person.set({name: "John"});
            document.write(person.escape("name"));
         </script>
   </head>
   <body></body>
</html>
```

Output

Let us carry out the following steps to see how the above code works:

- Save the above code in the **escape.htm** file.
- Open this HTML file in a browser.



```
John
```

BackboneJS – Model Has

Description

This method returns *true* if the attribute is set to a non-null or non-undefined value.

Syntax

```
model.has(attribute)
```

Parameters

- **attribute:** An attribute defines the property of a created model.

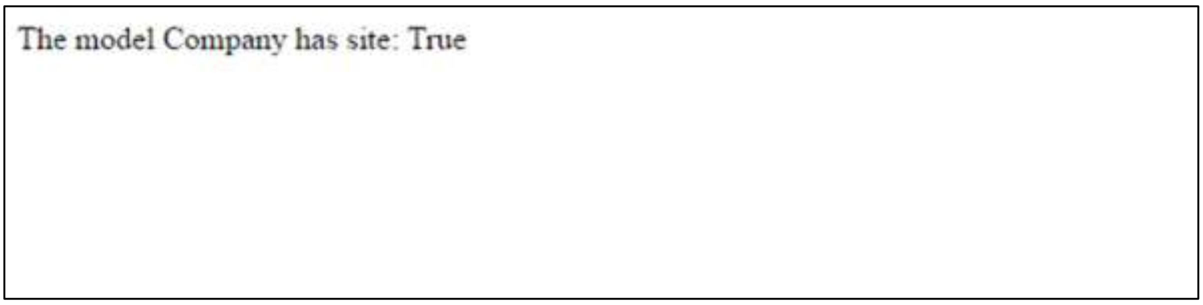
Example

```
<!DOCTYPE html>
   <head>
      <title> Model Example</title>
         <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js"
type="text/javascript"></script>
         <script type="text/javascript">
            var  Company = Backbone.Model.extend();
            var company=new Company();

            company.set({ Name: "TutorialsPoint",
comp_site:"www.tutorialspoint.com"});
            if(company.has('comp_site'))
            {
               document.write("The model Company has site: True");
            }
            else
            {
               document.write("The model Company has site: False");
            }
         </script>
   </head>
   <body></body>
</html>
```

Output

Let us carry out the following steps to see how the above code works:

- Save the above code in the **has.htm** file.
- Open this HTML file in a browser.



```
The model Company has site: True
```

BackboneJS – Model Unset

Description

It is used to remove an attribute from a backbone model.

Syntax

```
model.unset(attribute)
```

Parameters

- **attribute:** An attribute defines the property of a created model.

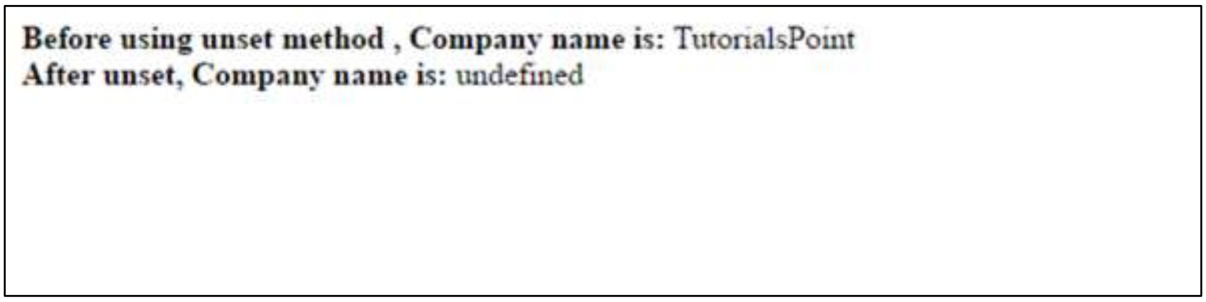
Example

```
<!DOCTYPE html>
   <head>
      <title> Model Example</title>
         <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js"
type="text/javascript"></script>
         <script type="text/javascript">
             var  Company = Backbone.Model.extend();
             var company=new Company();
             company.set({ comp_name: "TutorialsPoint"});
             document.write("<b>Before using unset method , Company name
is:</b> ", company.get('comp_name'));
             company.unset('comp_name');
             document.write("<br>");

             document.write("<b>After unset, Company name is:</b>",
company.get('comp_name'));
         </script>
   </head>
   <body></body>
</html>
```

Output

Let us carry out the following steps to see how the above code works:

- Save the above code in the **unset.htm** file.
- Open this HTML file in a browser.



```
Before using unset method , Company name is: TutorialsPoint
After unset, Company name is: undefined
```

BackboneJS – Model Clear

Description

It removes all the attributes, including the **id** attribute from a backbone model.

Syntax

```
model.clear(options)
```

Parameters

- **options:** It defines the parameters used like the id, name, etc., when they are removing from the model.

Example

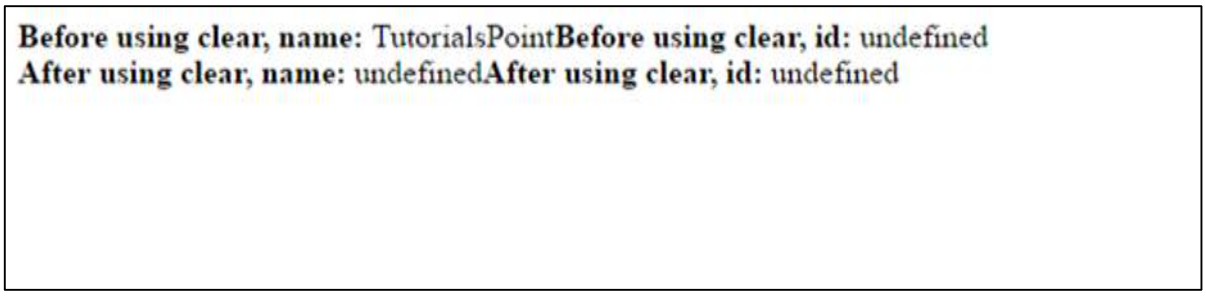
```
<!DOCTYPE html>
   <head>
      <title> Model Example</title>
```

```
         <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>


         <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js"
type="text/javascript"></script>
         <script type="text/javascript">
            var Model = Backbone.Model.extend();
            var model = new Model({name:"TutorialsPoint", id:1});
            document.write("<b>Before using clear, name: </b>",
model.get('name'));
            document.write("<b>Before using clear, id: </b>", model.get('id'));
            document.write("<br>");
            model.clear();
            document.write("<b>After using clear, name:</b> ",
model.get('name'));
            document.write("<b>After using clear, id: </b>", model.get('id'));
         </script>
   </head>
   <body></body>
</html>
```

Output

Let us carry out the following steps to see how the above code works:

- Save the above code in the **clear.htm** file.
- Open this HTML file in a browser.



Before using clear, name: TutorialsPoint**Before using clear, id:** undefined
After using clear, name: undefined**After using clear, id:** undefined

BackboneJS – Model Id

Description

It uniquely identifies the model entity, that might be manually set when the model is created or populated or when the model is saved on the server.

Syntax

```
model.id
```

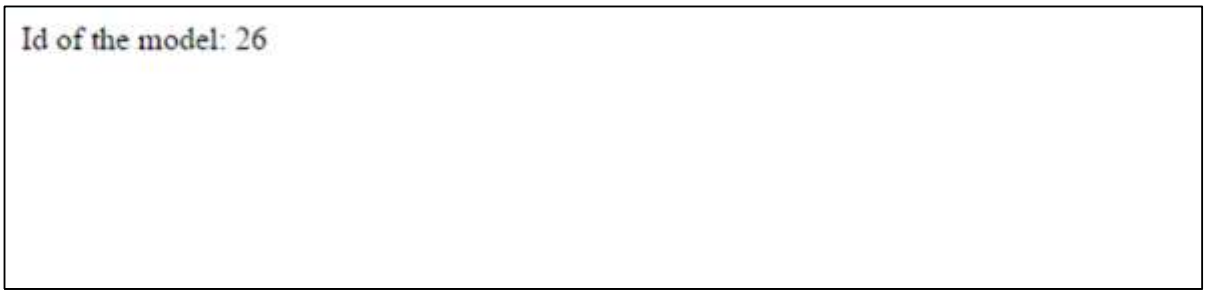
Example

```
<!DOCTYPE html>
   <head>
      <title> Model Example</title>
         <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js"
type="text/javascript"></script>
         <script type="text/javascript">
            var  Person = Backbone.Model.extend({
               defaults: {
                  id: 26
                }
             });
             var person = new Person();
             document.write("Id of the model: ", person.get('id'));
         </script>
   </head>
   <body></body>
</html>
```

Output

Let us carry out the following steps to see how the above code works:

- Save the above code in the **id.htm** file.
- Open this HTML file in a browser.



```
Id of the model: 26
```

BackboneJS – Model IdAttribute

Description

It gives us the model's unique identifier which contains the name of the member of the class which will be used as the **id**.

Syntax

```
model.idAttribute
```

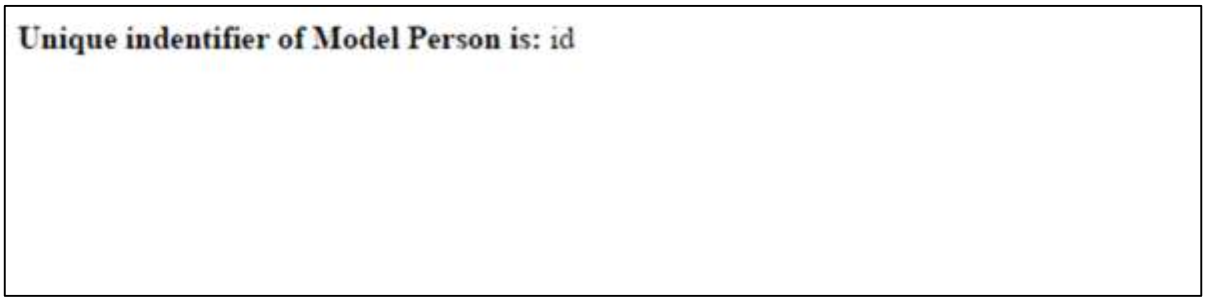
Example

```
<!DOCTYPE html>
   <head>
      <title> Model Example</title>
         <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js"
type="text/javascript"></script>
         <script type="text/javascript">
            var  Person = Backbone.Model.extend({
               defaults: {
                  idAttribute: 'id'
               }
            });
            var person = new Person({id:5, name:"TutorialsPoint"});
            document.write("<b>Unique indentifier of Model Person is:</b> ",
person.idAttribute);
         </script>

   </head>
   <body></body>
</html>
```

Output

Let us carry out the following steps to see how above code works:

- Save the above code in the **idattribute.htm** file
- Open this HTML file in a browser.



```
Unique indentifier of Model Person is: id
```

BackboneJS – Model Cid

Description

It is a client id which is auto generated by Backbone, so that the model can be uniquely identified on the client.

Syntax

```
model.cid
```

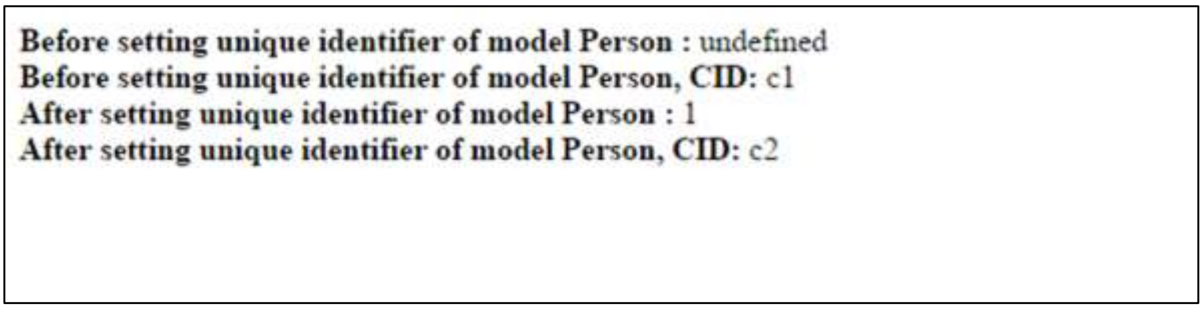
Example

```
<!DOCTYPE html>
   <head>
      <title> Model Example</title>
         <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js"
type="text/javascript"></script>
         <script type="text/javascript">

            var  Person = Backbone.Model.extend();
            var person = new Person();
            document.write("<b> Before setting unique identifier of model
Person : </b>",person.id);

            document.write("<br>");

            document.write("<b> Before setting unique identifier of model
Person, CID: </b>",person.cid);


            document.write("<br>");

            var person = new Person({id: 1});

            document.write("<b> After setting unique identifier of model Person
: </b>",person.id);

            document.write("<br>");

            document.write("<b> After setting unique identifier of model
Person, CID: </b>",person.cid);

         </script>

   </head>

   <body></body>

</html>
```

Output

Let us carry out the following steps to see how the above code works:

- Save the above code in the **cid.htm** file.
- Open this HTML file in a browser.



```
Before setting unique identifier of model Person : undefined
Before setting unique identifier of model Person, CID: c1
After setting unique identifier of model Person : 1
After setting unique identifier of model Person, CID: c2
```

BackboneJS – Model Attributes

Description

Attributes define property of a model and uses the **set()** method to update the attributes.

Syntax

```
model.attributes
```

Example

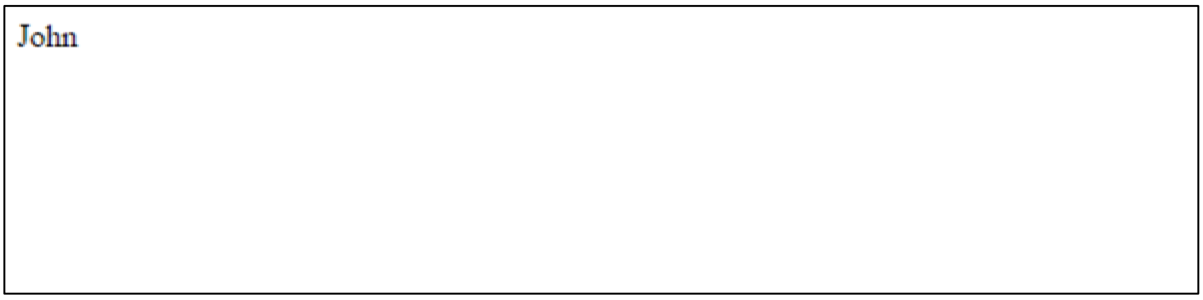
```
<!DOCTYPE html>
```

```
   <head>
      <title> Model Example</title>
         <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js"
type="text/javascript"></script>
         <script type="text/javascript">
            var  Person = Backbone.Model.extend();
            var person = new Person();
            person.set({ name: "John"});
            document.write(person.get('name'));
         </script>
   </head>
   <body></body>
</html>
```

Output

Let us carry out the following steps to see how the above code works:

- Save the above code in the **attributes.htm** file.
- Open this HTML file in a browser.



```
John
```

BackboneJS – Model Changed

Description

It changes all the attributes that have changed after setting the attributes using the set() method.

Syntax

```
model.changed
```

Example

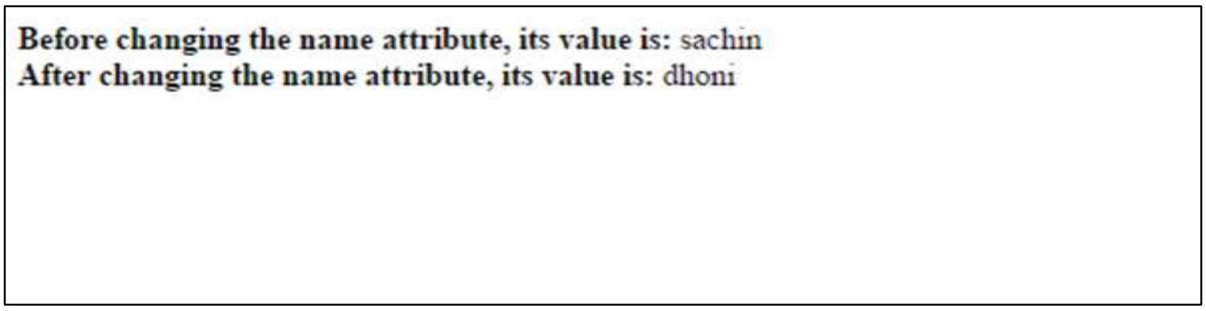
```
<!DOCTYPE html>
   <head>
      <title> Model Example</title>
         <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js"
type="text/javascript"></script>
         <script type="text/javascript">
            Player = Backbone.Model.extend({
            defaults: {
               p_name: 'sachin',
               country: 'india'
            },
            initialize: function () {
               this.bind("change:p_name", function (model) {
                  var name = model.get("p_name");
                  var ctry = model.get("country");
               });
           }
           });
           var person = new Player();
           document.write("<b>Before changing the name attribute, its value
is:</b> ", person.get("p_name"));
           person.set({ p_name: 'dhoni' });
           document.write("<br><b>After changing the name attribute, its value
is:</b> ", person.get("p_name"));
         </script>
   </head>
   <body></body>
</html>
```

Output

Let's carry out the following steps to see how above code works:

- Save above code in **changed.htm** file

- Open this HTML file in a browser.


```
Before changing the name attribute, its value is: sachin
After changing the name attribute, its value is: dhoni
```

BackboneJS – Model Defaults

Description

It sets a default value to a model, that means if user doesn't specify any data, the model won't fall with empty property.

Syntax

```
model.defaults
```

Example

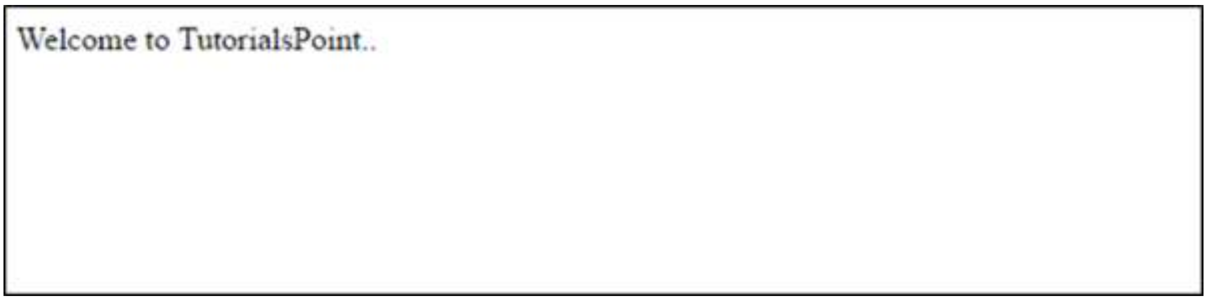
```
<!DOCTYPE html>
   <head>
```

```
      <title> Model Example</title>
         <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js"
type="text/javascript"></script>
         <script type="text/javascript">
            var  Person = Backbone.Model.extend({
            defaults : {
               "title": "Hello",
               "description": "Welcome to TutorialsPoint"

            }
            });
            var person = new Person();
            document.write(person.get('description'));
         </script>
   </head>
   <body></body>
</html>
```

Output

Let us carry out the following steps to see how the above code works:

- Save the above code in **defaults.htm** file.
- Open this HTML file in a browser.



```
Welcome to TutorialsPoint..
```

BackboneJS – Model toJSON

Description

It returns a copy of the attributes as an object for JSON stringification.

Syntax

```
model.toJSON(options)
```

Parameters

- **options:** It defines parameters such as the variable name, id used for a model when returning shallow copy of model's attributes.

Example

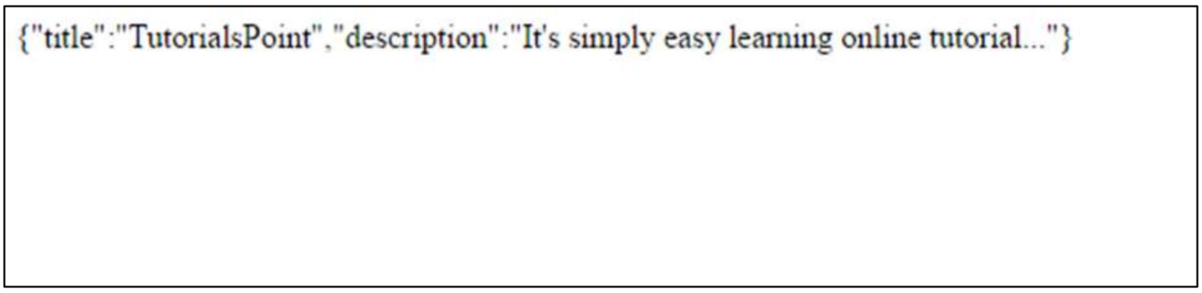
```
<!DOCTYPE html>
   <head>
      <title> Model Example</title>


         <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>
```

```
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js"
type="text/javascript"></script>
         <script type="text/javascript">
            var  Person = Backbone.Model.extend();
            var person = new Person({
               title: "TutorialsPoint",
               description: "It's simply easy learning online tutorial..."
            });
            document.write(JSON.stringify(person));
         </script>
   </head>
   <body></body>
</html>
```

Output

Let us carry out the following steps to see how the above code works:

- Save the above code in the **tojson.htm** file.
- Open this HTML file in a browser.



```
{"title":"TutorialsPoint","description":"It's simply easy learning online tutorial..."}
```

BackboneJS – Model Sync

Description

It can be used to communicate with the server and to represent the state of a model.

Syntax

```
model.sync(method,model,options)
```

Parameters

- **method**: It represents the CRUD operations such as create, read, update and delete.

- **model**: It is used to save the data on the model.
- **options**: It fires success or error message depending on the method succeeded.

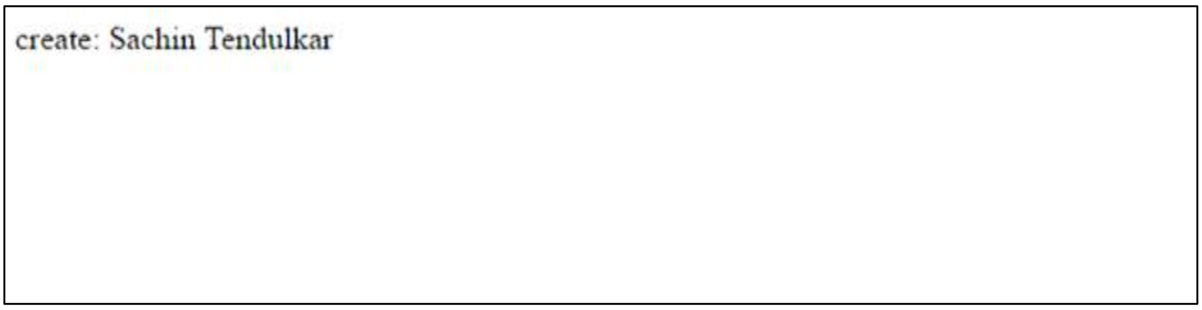
Example

```
<!DOCTYPE html>
   <head>
      <title> Model Example</title>
         <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js"
type="text/javascript"></script>
         <script type="text/javascript">
            var details = new Backbone.Model({
               fname: "Sachin",
               lname: "Tendulkar"
            });
            Backbone.sync = function(method, model) {
               document.write(method + ": " + model.get('fname')+ " "
+model.get('lname'));
            };
           details.save();
         </script>
   </head>
   <body></body>
</html>
```

Output

Let us carry out the following steps to see how the above code works:

- Save the above code in the **sync.htm** file.
- Open this HTML file in a browser.



```
create: Sachin Tendulkar
```

BackboneJS – Model Fetch

Description

It accepts data from the server by delegating the sync() method in the model.

Syntax

```
model.fetch(options)
```

Parameters

- **options:** It accepts parameters such as id, name, etc., which are used for a model.

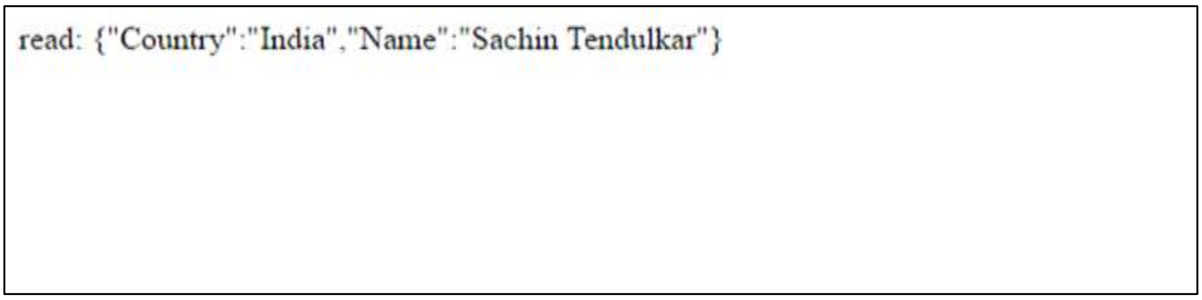
Example

```
<!DOCTYPE html>
   <head>
      <title> Model Example</title>
         <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js"
type="text/javascript"></script>
         <script type="text/javascript">
            Backbone.sync = function(method, model) {
               document.write(method + ": " + JSON.stringify(model));
            };

            var person = new Backbone.Model({
               Country:"India",

               Name:"Sachin Tendulkar"
            });
            person.fetch();

         </script>

   </head>

   <body></body>

</html>
```

Output

Let us carry out the following steps to see how the above code works:

- Save the above code in the **fetch.htm** file.
- Open this HTML file in a browser.



```
read: {"Country":"India","Name":"Sachin Tendulkar"}
```

BackboneJS – Model Save

Description

- It saves data of the model by delegating to sync() method which reads and saves the model every time when Backbone calls it.

Syntax

```
model.save(attributes,options)
```

Parameters

- **attributes:** It defines the property of a model.
- **options:** It accepts parameters such as the id, name, etc., which are used for a model.

Example

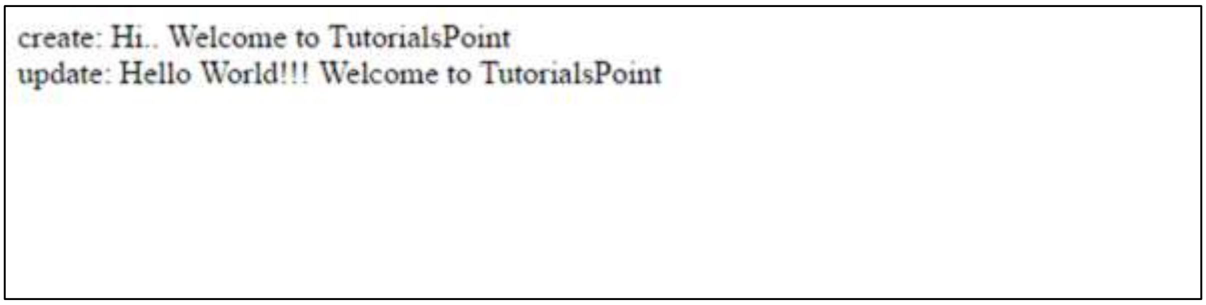
```
<!DOCTYPE html>

   <head>

      <title> Model Example</title>

         <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>

         <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>
```

```
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js"
type="text/javascript"></script>
         <script type="text/javascript">
         var details = new Backbone.Model({
            name: "Hi..",
            site: "Welcome to TutorialsPoint"
         });

         Backbone.sync = function(method, model) {
            document.write(method + ": " + model.get('name')+ " "
+model.get('site'));
            model.set('id', 1);
         };
         details.save();
         document.write("<br>");
         details.save({name : "Hello World!!!"});
         </script>
   </head>
   <body></body>
</html>
```

Output

Let us carry out the following steps to see how the above code works:

- Save the above code in the **save.htm** file.
- Open this HTML file in a browser.



```
create: Hi.. Welcome to TutorialsPoint
update: Hello World!!! Welcome to TutorialsPoint
```

BackboneJS – Model Destroy

Description

It destroys or removes the model from the server by using the **Backbone.sync** method which delegates the HTTP "delete" request.

Syntax

```
model.destroy(options)
```

Parameters

- **options:** It includes parameters such as the id, name, etc., which will get removed from the server.

Example

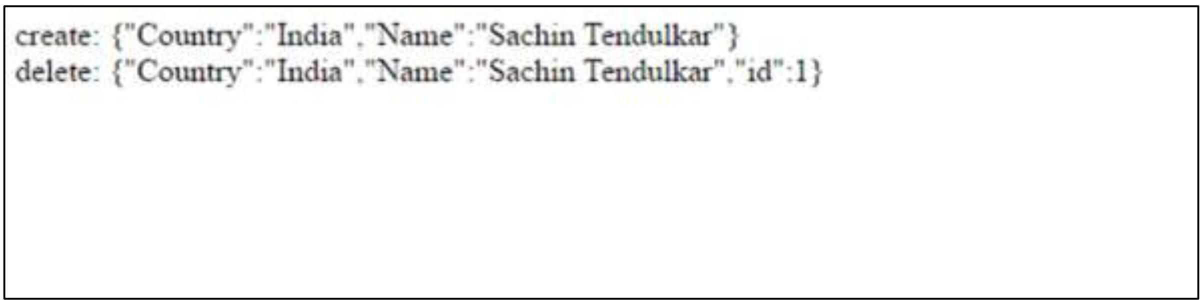
```
<!DOCTYPE html>
   <head>
      <title> Model Example</title>
         <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js"
type="text/javascript"></script>
         <script type="text/javascript">
         Backbone.sync = function(method, model) {
            document.write(method + ": " + JSON.stringify(model)+"<br>");
           model.set('id',1);
         };
         var details = new Backbone.Model({

            Country:"India",
            Name:"Sachin Tendulkar"
         });
         details.save();
         details.destroy();
         </script>
   </head>
   <body></body>
</html>
```

Output

Let us carry out the following steps to see how the above code works:

- Save the above code in the **destroy.htm** file.

- Open this HTML file in a browser.



```
create: {"Country":"India","Name":"Sachin Tendulkar"}
delete: {"Country":"India","Name":"Sachin Tendulkar","id":1}
```

BackboneJS – Model Validate

Description

It validates the model and input provided by the user. If the input is invalid, it returns a specified error message or if the input is valid, it doesn't specify anything and simply displays the result.

Syntax

```
model.validate(attributes,options)
```

Parameters

- **attributes:** These attributes define the property of a model.
- **options:** It includes true as an option to validate the attributes.

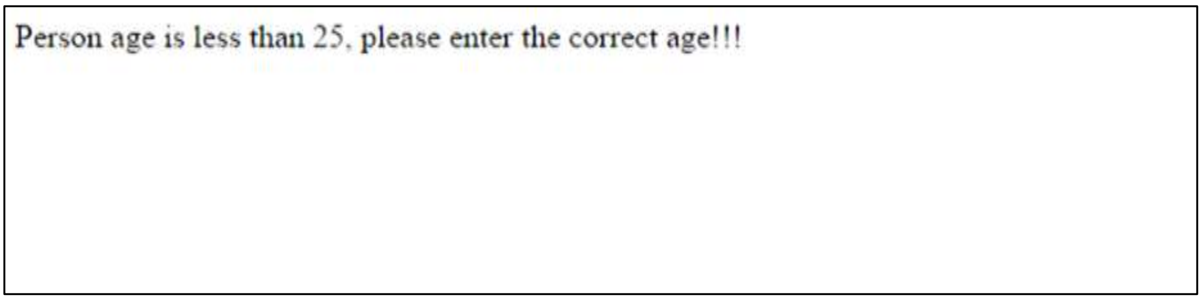
Example

```
<!DOCTYPE html>
   <head>
      <title>Model Example</title>
         <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js"
type="text/javascript"></script>
   </head>
   <body>

   <script type="text/javascript">
      var Person = Backbone.Model.extend({
         defaults: {
            name: 'john',
            age: 25,
            occupation: 'working'
         },
         initialize : function(){
            this.on("invalid",function(model,error){
            document.write(error);
         });
         },
         validate: function(attributes) {
            if ( attributes.age < 25 ) {
               return 'Person age is less than 25, please enter the correct
age!!! ';
            }
           if ( ! attributes.name ) {
              return 'please enter the name!!!';
           }
         },
   });
   var person = new Person();
   person.on('invalid', function() {
   this.arguments;
   });
   person.set({ age : '20' }, { validate : true });
   </script>
   </body>
</html>
```

Output

Let us carry out the following steps to see how the above code works:

- Save the above code in the **validate.htm** file.
- Open this HTML file in a browser.



```
Person age is less than 25, please enter the correct age!!!
```

BackboneJS – Model validationError

Description

It is the returned value by **validate()** method which displays a validation error, if the validation fails or after the **invalid** event is triggered.

Syntax

```
model.validationError
```

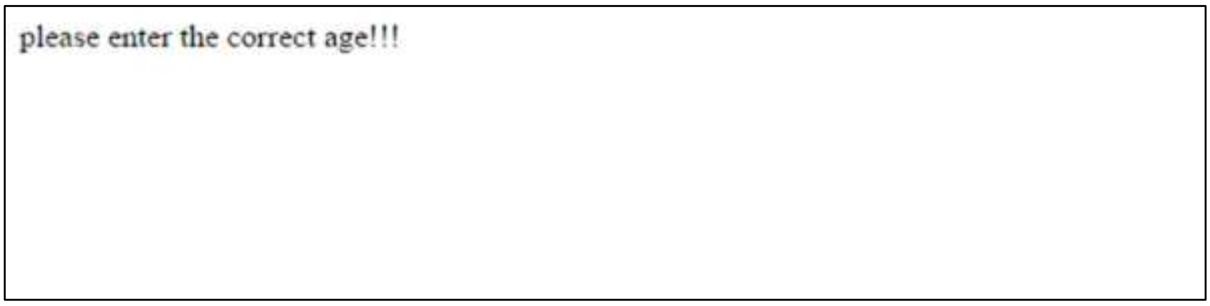
Example

```
<!DOCTYPE html>
   <head>
      <title>Model Example</title>
         <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js"
type="text/javascript"></script>
   </head>
   <body>
   <script type="text/javascript">
      var Person = Backbone.Model.extend({
         defaults: {
            name: 'john',
            age: 25,
            occupation: 'working'
         },
         initialize : function(){
            this.on("invalid",function(model,error){

            document.write(error);
         });
         },
         validate: function(attributes) {
            if ( attributes.age < 25 ) {
               return 'please enter the correct age!!! ';
            }
           if ( ! attributes.name ) {
              return 'please enter the name!!!';
           }
         },
   });
   var person = new Person();
   person.on('invalid', function() {
      this.arguments;
   });
   person.set({ age : '20' }, { validate : true });
   </script>
   </body>
</html>
```

Output

Let us carry out the following steps to see how the above code works:

- Save the above code in the **validationerror.htm** file.
- Open this HTML file in a browser.



```
please enter the correct age!!!
```

BackboneJS – Model isValid

Description

It checks the model state by using the validate() method and also checks validations for each attribute.

Syntax

```
model.isValid()
```

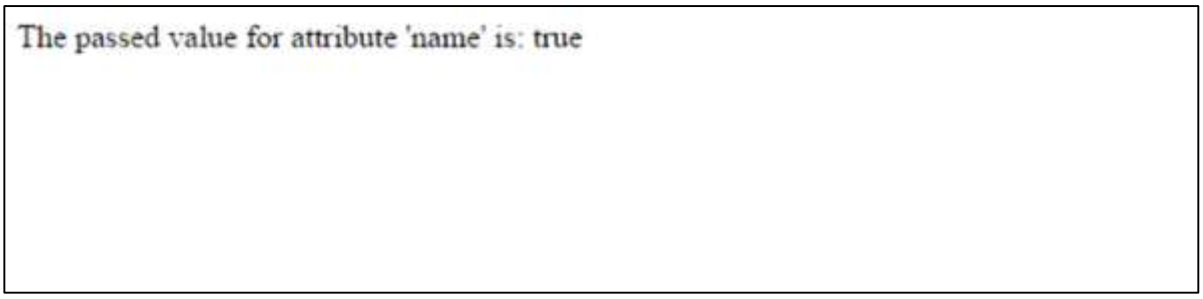
Example

```
<!DOCTYPE html>
   <head>
      <title> Model Example</title>
         <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js"
type="text/javascript"></script>
         <script type="text/javascript">
            var Person = Backbone.Model.extend({
               defaults: {
                  name: '',
                  age: 25,
               },
            validate: function(attributes) {
               if (attributes.name == '') {
               document.write("please enter the name!!!!");
               }
            }
            });
            var person = new Person({
               name: 'John'
            });
            document.write("The passed value for attribute 'name' is: ",
person.isValid());


         </script>
   </head>
   <body></body>
</html>
```

Output

Let us carry out the following steps to see how the above code works:

- Save the above code in the **isvalid.htm** file.
- Open this HTML file in a browser.



```
The passed value for attribute 'name' is: true
```

BackboneJS – Model Url

Description

It is used for the instance of the model and returns a url where the model's resource is located.

Syntax

```
model.url()
```

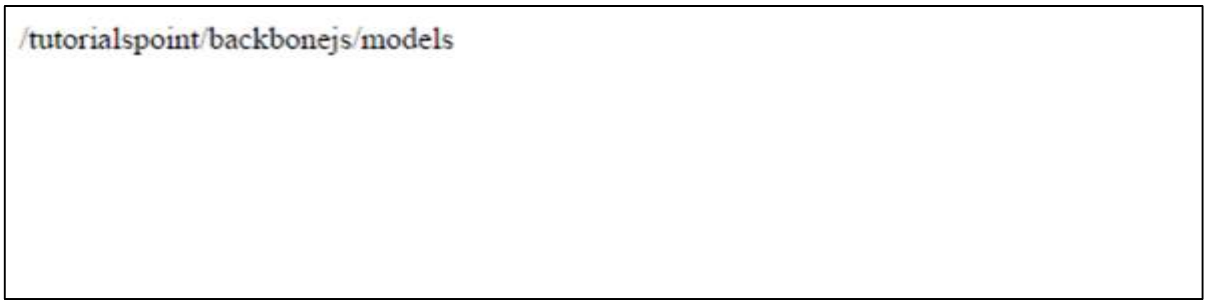
Example

```
<!DOCTYPE html>
   <head>
      <title> Model Example</title>
         <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js"
type="text/javascript"></script>
         <script type="text/javascript">
            var MyModel = Backbone.Model.extend({
               urlRoot: '/tutorialspoint/backbonejs'
            });
            var mymodel = new MyModel({ id: "models" });
            document.write(mymodel.url());
         </script>

   </head>
   <body></body>
</html>
```

Output

Let us carry out the following steps to see how the above code works:

- Save the above code in the **url.htm** file.
- Open this HTML file in a browser.



```
/tutorialspoint/backbonejs/models
```

BackboneJS – Model urlRoot

Description

It enables the url function by using the model id to generate the URL.

Syntax

```
model.urlRoot()
```

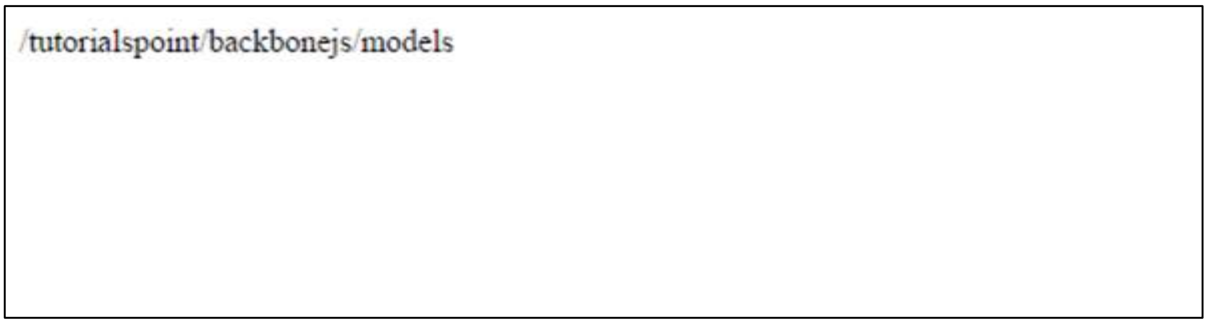
Example

```
<!DOCTYPE html>
   <head>
      <title> Model Example</title>
         <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js"
type="text/javascript"></script>
         <script type="text/javascript">
            var MyModel = Backbone.Model.extend({

               urlRoot: '/tutorialspoint/backbonejs'
            });
            var mymodel = new MyModel({ id: "models" });
            document.write(mymodel.url());
         </script>
   </head>
   <body></body>
</html>
```

Output

Let us carry out the following steps to see how the above code works:

- Save the above code in the **urlroot.htm** file.
- Open this HTML file in a browser.



```
/tutorialspoint/backbonejs/models
```

BackboneJS – Model Parse

Description

It is used by the server and returns the model's data by passing through the response object and represents the data in JSON format.

Syntax

```
model.parse(response,options)
```

Parameters

- **response:** It is passed using response raw object and returns attributes to be set on the model.
- **options:** It includes true as an option which represents data in JSON format.

Example

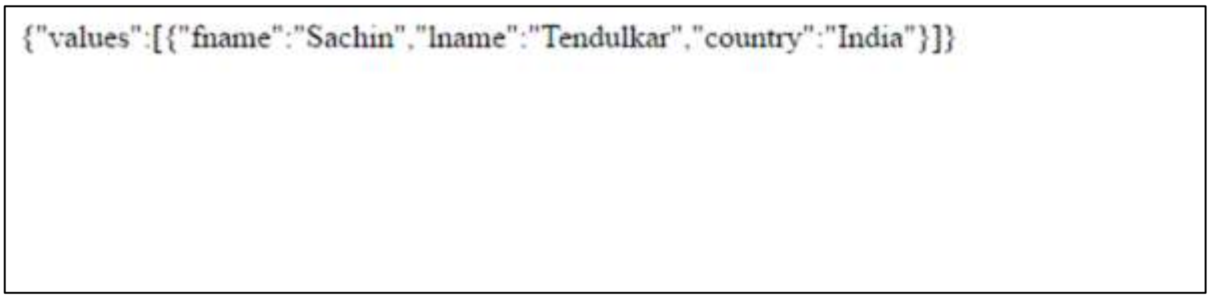
```
<!DOCTYPE html>
   <html>
      <head>
         <title>Model Example</title>
            <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>
```

```
            <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>
            <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js"
type="text/javascript"></script>
      </head>
   <body>
   <script type="text/javascript">
      var myData ={
         "values": [{
             "fname": "Sachin",
             "lname": "Tendulkar",
             "country": "India"
          }]
      };
      var Person  = Backbone.Model.extend({
         parse : function(response, options){
            document.write(JSON.stringify(response));
         }
      });
      var person = new Person(myData, {parse: true});
   </script>
   </body>
</html>
```

Output

Let us carry out the following steps to see how the above code works:

- Save the above code in the **parse.htm** file.
- Open this HTML file in a browser.



```
{"values":[{"fname":"Sachin","lname":"Tendulkar","country":"India"}]}
```

BackboneJS – Model Clone

Description

A Model Clone is used to create a deep copy of a model or to copy one model object to another object.

Syntax

```
model.clone()
```

Example

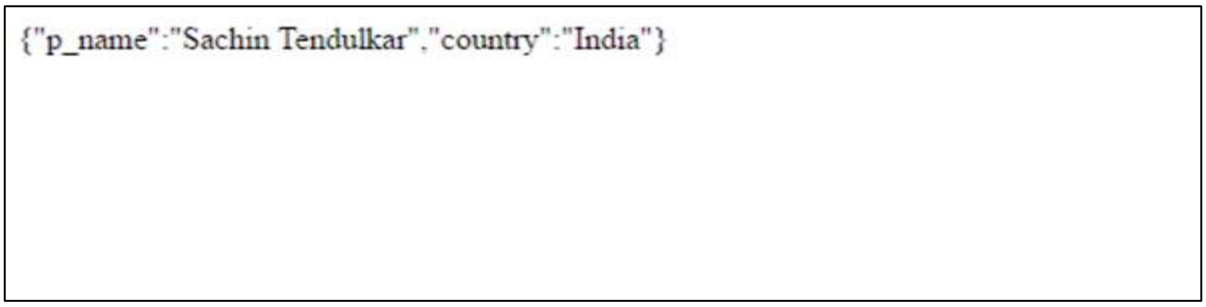
```
<!DOCTYPE html>
   <head>
      <title> Model Example</title>
         <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js"
type="text/javascript"></script>
         <script type="text/javascript">
            var Person  = Backbone.Model.extend();
            var person=new Person({
               p_name: 'Sachin Tendulkar',
               country: 'India'
            });

            var details=person.clone();
            //output would be a deep clone of Person Model
            document.write(JSON.stringify(details));
         </script>
   </head>

   <body></body>
</html>
```

Output

Let us carry out the following steps to see how the above code works:

- Save the above code in the **clone.htm** file.

- Open this HTML file in a browser.



```
{"p_name":"Sachin Tendulkar","country":"India"}
```

BackboneJS – Model hasChanged

Description

It returns true if the attribute gets changed since the last *set*.

Syntax

```
model.hasChanged(attribute)
```

Parameters

- **attribute:** It defines the property of a model.

Example

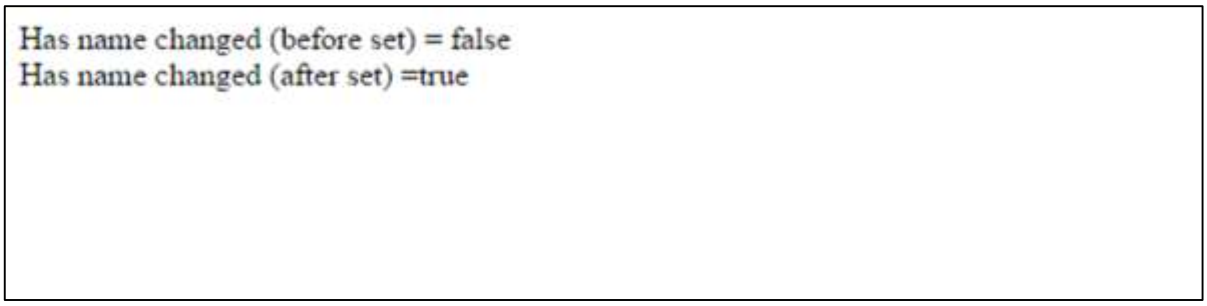
```
<!DOCTYPE html>
   <head>
      <title> Model Example</title>
         <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js"
type="text/javascript"></script>

         <script type="text/javascript">
            var person = new Backbone.Model({
               name: 'john'
            });

            document.write('Has name changed (before set) = ' +
person.hasChanged());
```

```
            person.set('name', 'smith', {silent: true});

            document.write('<br>Has name changed (after set) =' +
person.hasChanged());

         </script>

   </head>

   <body></body>

</html>
```

Output

Let us carry out the following steps to see how the above code works:

- Save the above code in the **haschanged.htm** file.
- Open this HTML file in a browser.



```
Has name changed (before set) = false
Has name changed (after set) =true
```

BackboneJS – Model isNew

Description

If the model is not saved to the server and does not yet have an id, it is considered to be new. This method helps to determine this state.

Syntax

```
model.isNew()
```

Example

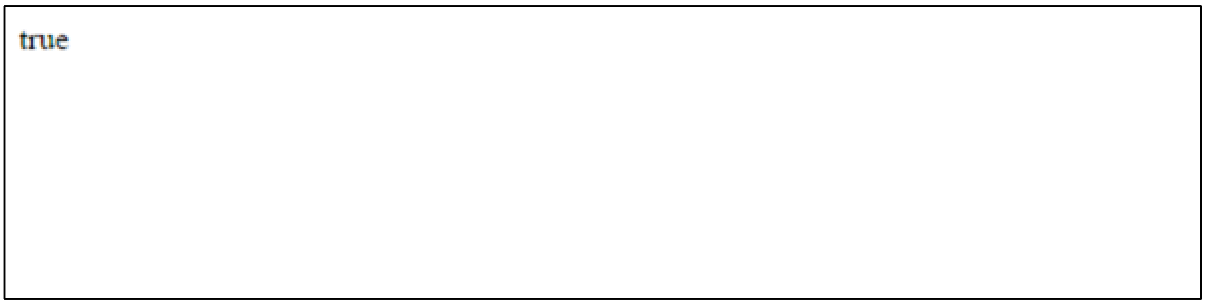
```
<!DOCTYPE html>

   <head>

      <title> Model Example</title>

         <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>

         <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>

         <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js"
type="text/javascript"></script>
```

```
          <script type="text/javascript">
             var Person = Backbone.Model.extend({
                defaults: {
                   title: 'TutorialsPoint'
                }
             });
             var person = new Person();

             document.write(person.isNew());
          </script>
    </head>
    <body></body>
</html>
```

Output

Let us carry out the following steps to see how the above code works:

- Save the above code in the **isnew.htm** file.
- Open this HTML file in a browser.



```
true
```

BackboneJS – Model changedAttributes

Description

It returns a hash of only the model's attributes that have changed since the last set becomes false, if there are no attributes.

Syntax

```
model.changedAttributes(attributes)
```

Parameters

- **attributes:** Attributes define the property of a model.

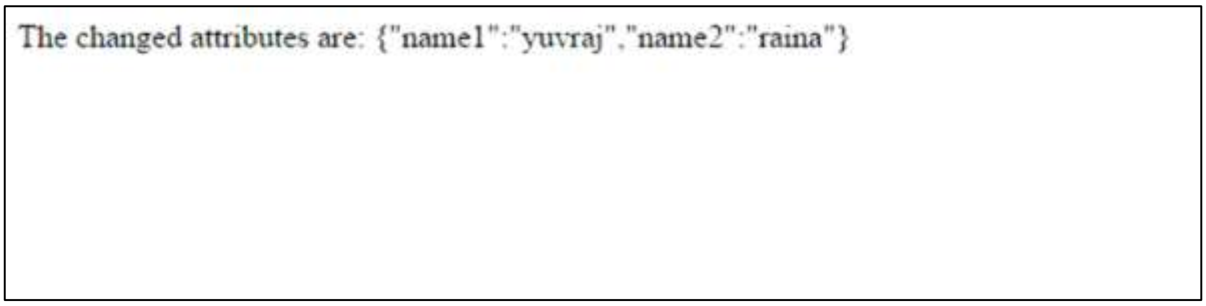
Example

```
<!DOCTYPE html>
   <head>
      <title>Model Example</title>
         <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js"
type="text/javascript"></script>
   </head>
   <body>
   <script type="text/javascript">
      var values = new Backbone.Model({
         name1: 'sachin',
         name2: 'dhoni',
         name3: 'sehwag'
      });
      values.on('change', function() {
         document.write("The changed attributes are: ");
         document.write(JSON.stringify(values.changedAttributes()));
      });
      values.set({
         name1: 'yuvraj',
         name2: 'raina'


      });
   </script>
   </body>
</html>
```

Output

Let us carry out the following steps to see how the above code works:

- Save the above code in the **changedattributes.htm** file.
- Open this HTML file in a browser.



```
The changed attributes are: {"name1":"yuvraj","name2":"raina"}
```

BackboneJS – Model Previous

Description

It determines the previous value of the changed attribute.

Syntax

```
model.previous(attribute)
```

Parameters

- **attribute:** It represents the property of a model.

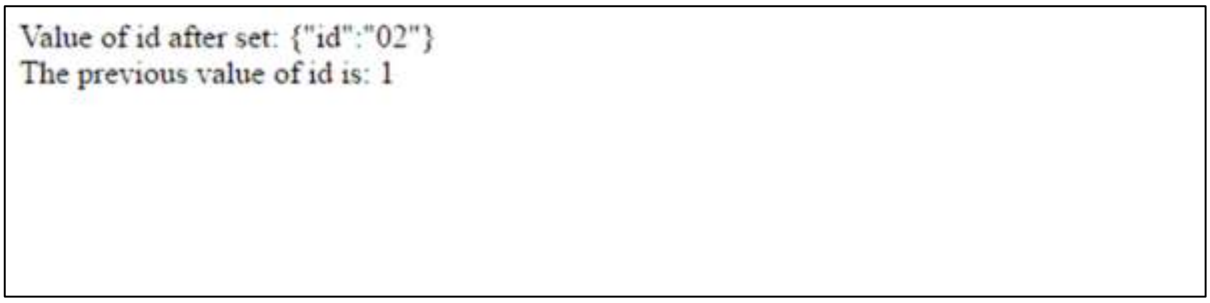
Example

```
<!DOCTYPE html>

   <head>

      <title>Model Example</title>

         <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>

         <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>

         <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js"
type="text/javascript"></script>


   </head>

   <body>

   <script type="text/javascript">

      var model = new Backbone.Model({

         id:01,

         player:'Sachin'

      });

      model.set('id', '02');
      document.write("Value of id after set: ",
JSON.stringify(model.changedAttributes()));

      document.write("<br>");

      document.write("The previous value of id is: ", model.previous('id'));

   </script>

   </body>

</html>
```

Output

Let us carry out the following steps to see how the above code works:

- Save the above code in the **previous.htm** file.
- Open this HTML file in a browser.



```
Value of id after set: {"id":"02"}
The previous value of id is: 1
```

BackboneJS – Model previousAttributes

Description

It returns a copy of the model's previous attributes prior to the last change event. This is useful for getting a difference between various versions of a model, or getting back to a valid state after an error occurs.

Syntax

```
model.previousAttributes()
```

Example

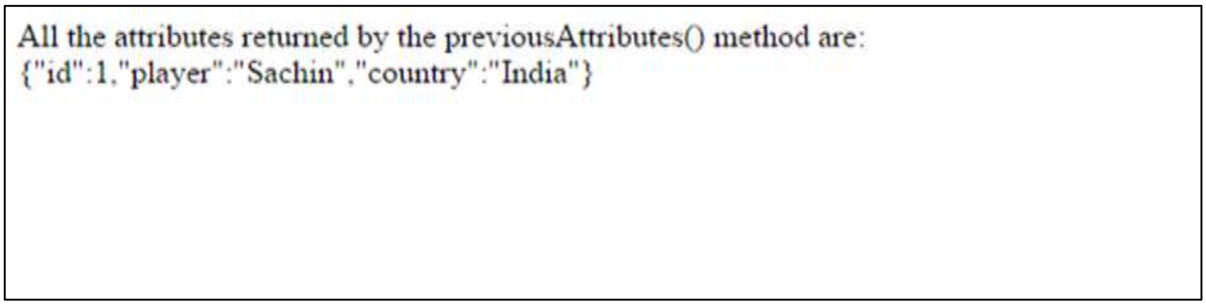
```
<!DOCTYPE html>

   <head>

      <title>Model Example</title>

         <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>

         <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>
```

```
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js"
type="text/javascript"></script>

   </head>

   <body>

   <script type="text/javascript">

      var model = new Backbone.Model({

         id:01,

         player:'Sachin',

         country:'India'


      });

      model.set('id', '02');

      document.write("All the attributes returned by the previousAttributes()
method are: ");

      document.write("<br>");

      document.write(JSON.stringify(model.previousAttributes()));

   </script>

   </body>

</html>
```

Output

Let us carry out the following steps to see how the above code works:

- Save the above code in the **previousattributes.htm** file.
- Open this HTML file in a browser.



```
All the attributes returned by the previousAttributes() method are:
{"id":1,"player":"Sachin","country":"India"}
```

Underscore Methods

There are six **Underscore.js** methods which provides their functionality to be used on the Backbone.Model.

S.No.	Methods & Description
1	**_.keys(object)** It is used to access the object's enumerable properties.
2	**_.values(object)** It is used to get values of object's properties.
3	**_.pairs(object)** It describes the object's properties in terms of key value pairs.
4	**_.invert(object)** It returns the copy of object, in which keys have become the values and vice versa.
5	**_.pick(object, *keys)** It returns the copy of object and indicates which keys to pick up.
6	**_.omit(object, *keys)** It returns the copy of object and indicates which keys to omit.

6. BackboneJS – Collection

Collections are ordered sets of Models. We just need to extend the backbone's collection class to create our own collection. Any event that is triggered on a model in a collection will also be triggered on the collection directly. This allows you to listen for changes to specific attributes in any model in a collection.

The following table lists down all the methods which you can use to manipulate the BackboneJS-Collection:

S.No.	Methods & Description
1	**extend** Extends the backbone's collection class to create a collection.
2	**model** To specify the model class, we need to override the model property of the collection class.
3	**initialize** When a model instance is created, it is invoked by defining the initialize function when the collection is created.
4	**models** Array of models which are created inside the collection.
5	**toJSON** Returns the copy of the attributes of a model using the JSON format in the collection.
6	**sync** It represents the state of the model and uses the Backbone.sync to display the state of the collection.
7	**add** Add a model or array of models to the collection.
8	**remove** Removes a model or array of models from the collection.

9	**reset** It resets the collection and populates with new array of models or will empty the entire collection.
10	**set** It is used to update the collection with a set of items in a model. If any new model is found, the items will be added to that model.
11	**get** It is used to retrieve the model from a collection by using the **idor cid**.
12	**at** Retrieve the model from a collection by using specified index.
13	**push** It is similar to the add() method which takes the array of models and pushes the models to the collection.
14	**pop** It is similar to the remove() method which takes the array of models and removes the models from the collection.
15	**unshift** Add a specified model at the beginning of a collection.
16	**shift** It removes the first item from the collection.
17	**slice** Displays the shallow copy of the elements from the collection model.
18	**length** Counts the number of models in the collection.
19	**comparator** It is used to sort the items in the collection.

20	**sort** Sorts the items in the collection and uses comparator property in order to sort the items.
21	**pluck** Retrieves the attributes from the model in the collection.
22	**where** It is used to display the model by using the matched attribute in the collection.
23	**findWhere** It returns the model, that matches the specified attribute in the collection.
24	**url** It creates an instance of the collection and returns where resources are located.
25	**parse** Returns the collection's data by passing through the response object and represents the data in JSON format.
26	**clone** It returns the shallow copy of the specified object.
27	**fetch** It extracts the data from the model in the collection using the sync method.
28	**create** It creates a new instance of the model in the collection.

BackboneJS – Collection Extend

Description

It extends the backbone's collection class to create an own *collection*.

Syntax

```
Backbone.Collection.extend(properties, classProperties)
```

Parameters

- **properties:** It provides instance properties for the collection class.

- **classProperties:** Class properties are attached to the collection's constructor function.

Example

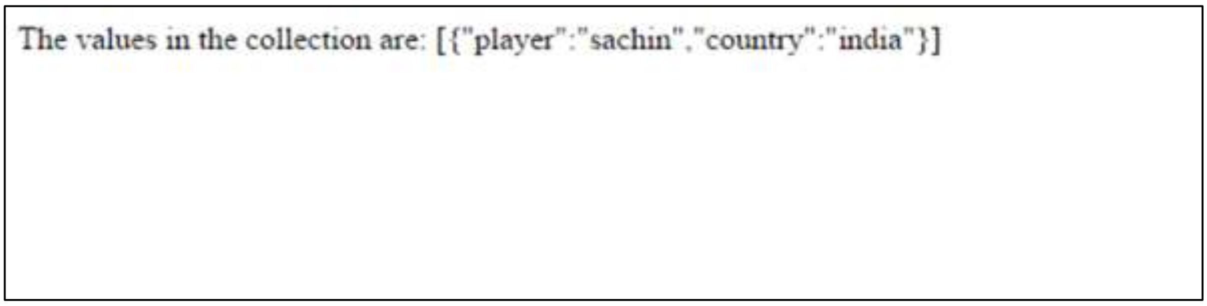
```
<!DOCTYPE html>
   <head>
      <title>Collection Example</title>
         <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js"
type="text/javascript"></script>
   </head>
   <body>
      <script type="text/javascript">
            //The model 'MyTeam' includes default values and  extended using
the Backbone.Model class
         var MyTeam = Backbone.Model.extend({
            defaults: {

               player: "sachin",
               country: "india"
            },
         });

         //'MyTeam1' is an instance of the collection
         var MyTeam1 = Backbone.Collection.extend({
            model: MyTeam  //model 'MyTeam' is specified for a collection by
overriding the 'model' property
         });

         //The collection 'MyTeam1' is instantiated by using new keyword
         var myval=new MyTeam1({});

         //The JSON.stringify() method returns values of collection in the
JSON format
```

```
         document.write("The values in the collection are:
",JSON.stringify(myval));

      </script>

   </body>

</html>
```

Output

Let us carry out the following steps to see how the above code works:

- Save the above code in the **extend.htm** file.
- Open this HTML file in a browser.



```
The values in the collection are: [{"player":"sachin","country":"india"}]
```

BackboneJS – Collection Model

Description

To specify the model class, we need to override the model property of the collection class.

Syntax

```
Backbone.Collection.model
```

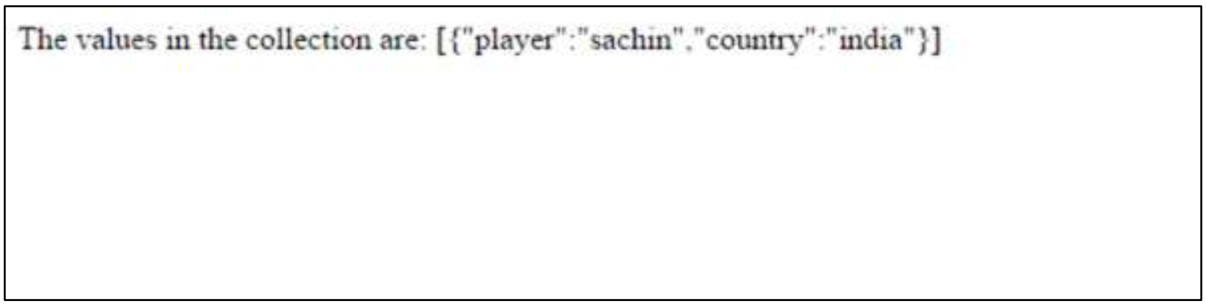
Example

```
<!DOCTYPE html>

   <head>

      <title>Collection Example</title>

         <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>

         <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>

         <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js"
type="text/javascript"></script>

   </head>
   <body>
      <script type="text/javascript">
        //The model 'MyTeam' includes default values and extended using the
Backbone.Model class
         var MyTeam = Backbone.Model.extend({
            defaults: {
               player: "sachin",
               country: "india"
            },
         });

         //'MyTeam1' is an instance of the collection
         var MyTeam1 = Backbone.Collection.extend({
            model: MyTeam  //model 'MyTeam'is specified for a collection by
overriding the 'model' property
         });

         //The collection 'MyTeam1' is instantiated by using new keyword
         var myval=new MyTeam1({});

         //The JSON.stringify() method returns values of collection in the JSON
format

         document.write("The values in the collection are:
",JSON.stringify(myval));
      </script>
   </body>
</html>
```

Output

Let us carry out the following steps to see how the above code works:

- Save the above code in the **model.htm** file.
- Open this HTML file in a browser.



```
The values in the collection are: [{"player":"sachin","country":"india"}]
```

BackboneJS – Collection Initialize

Description

When the model instance is created, it is invoked by defining the **initialize** function when the collection is created.

Syntax

```
new Backbone.Collection(models, options)
```

Parameters

- **models:** It specifies the initial array of models.
- **options:** These are the collection types attached to the collection directly by the passing model object.

Example

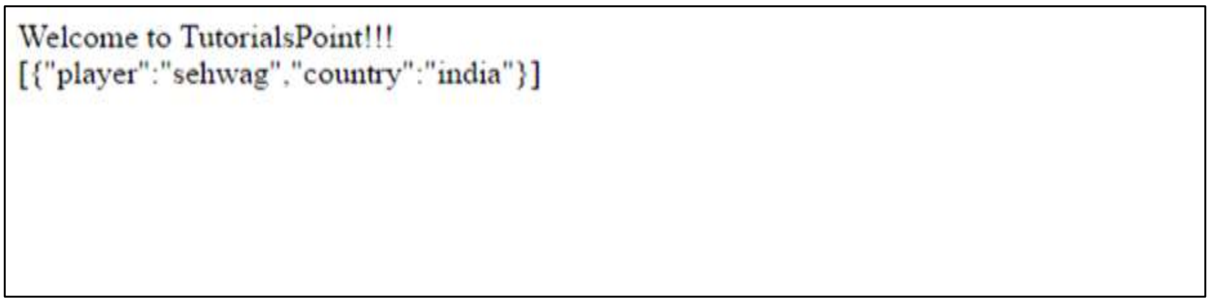
```
<!DOCTYPE html>
   <head>
      <title>Collection Example</title>
         <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>


         <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js"
type="text/javascript"></script>
   </head>
   <body>
      <script type="text/javascript">
            //The model 'MyTeam' contains default values and extended using the
Backbone.Model class
         var MyTeam = Backbone.Model.extend({
            defaults: {
               player: "sachin",
               country: "india"
            },

            //The model instance is invoked by defining initialize function
            initialize: function(){
               document.write("Welcome to TutorialsPoint!!!");
            }
        });

        //The 'MyTeam1' is a collection instance and model 'MyTeam' is
specified by overriding the 'model' property
        var MyTeam1 = Backbone.Collection.extend({
           model: MyTeam
        });
        var player1 = new MyTeam({
           player: "sehwag",
           country: "india"
        });

        //The 'player1' is a type of collection by passing model object in the
collection
        var myval=new MyTeam1([player1]);



        //The 'myval.models' define the array of models inside the collection
        document.write("<br>"+JSON.stringify(myval.models));
      </script>
   </body>
</html>
```

Output

Let us carry out the following steps to see how the above code works:

- Save the above code in the **initialize.htm** file.

- Open this HTML file in a browser.



```
Welcome to TutorialsPoint!!!
[{"player":"sehwag","country":"india"}]
```

BackboneJS – Collection Models

Description

These are the array of models which are created inside the collection.

Syntax

```
collection.models
```

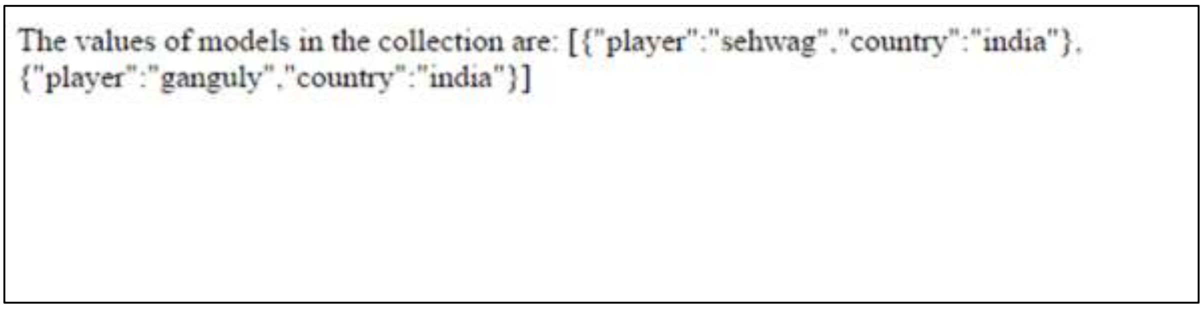
Example

```
<!DOCTYPE html>
   <head>
      <title>Collection Example</title>
         <script src="https://code.jquery.com/jquery-2.1.3.min.js" 
type="text/javascript"></script>
         <script 
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>
         <script 
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js" 
type="text/javascript"></script>
   </head>
   <body>
      <script type="text/javascript">
            //The model 'MyTeam' contains default values and extended using the 
Backbone.Model class
         var MyTeam = Backbone.Model.extend({
            defaults: {
               player: "sachin",
               country: "india"
            }
        });
        //The 'MyTeam1' is a collection instance and model 'MyTeam' is
specified by overriding the 'model' property
        var MyTeam1 = Backbone.Collection.extend({
           model: MyTeam
        });
        var player1 = new MyTeam({
           player: "sehwag",
           country: "india"
        });
        var player2 = new MyTeam({
           player: "ganguly",
           country: "india"
        });

        //Passing the objects 'player1' and 'player2' to the collection
        var myval=new MyTeam1([player1,player2]);

        //The 'myval.models' define the array of models inside the collection
        document.write("The values of models in the collection are:
"+JSON.stringify(myval.models));
      </script>
   </body>
</html>
```

Output

Let us carry out the following steps to see how the above code works:

- Save the above code in the **models.htm** file
- Open this HTML file in a browser.



```
The values of models in the collection are: [{"player":"sehwag","country":"india"},
{"player":"ganguly","country":"india"}]
```

BackboneJS – Collection toJSON

Description

It returns the copy of the attributes hash of each model in the collection using JSON format.

Syntax

```
collection.toJSON(options)
```

Parameters

- **options:** It includes options as the collection instance and converts it into JSON format.

Example

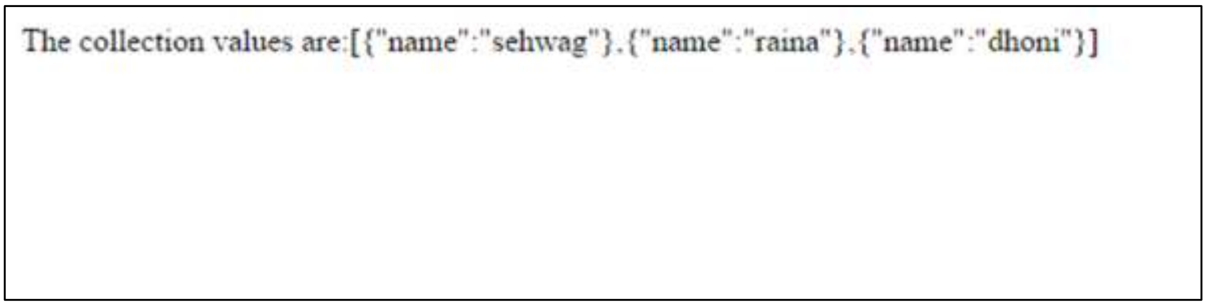
```
<!DOCTYPE html>
   <head>
      <title>Collection Example</title>
         <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js"
type="text/javascript"></script>
   </head>
   <body>
      <script type="text/javascript">
             //'Player' is a model and contains defualt value for the model
         var Player = Backbone.Model.extend({
            defaults: {

              name: "sachin"
            }
         });

         //The 'PlayersCollection' is a collection instance and model 'Player'
is specified by overriding the 'model' property
         var PlayersCollection = Backbone.Collection.extend({
            model: Player
```

```
          });
          $(function(){
             var mycollection = new PlayersCollection();

             //The set() method sets the values for the 'name' attribute
             mycollection.set([{name: 'sehwag'},
                              {name: 'raina'},
                              {name: 'dhoni'}
                          ]);
             //The JSON.stringify() method returns values of the collection in
the JSON format
             document.write("The collection values are:",
JSON.stringify(mycollection.toJSON()));
         });
      </script>
   </body>
</html>
```

Output

Let us carry out the following steps to see how the above code works:

- Save the above code in the **tojson.htm** file.
- Open this HTML file in a browser.



```
The collection values are:[{"name":"sehwag"},{"name":"raina"},{"name":"dhoni"}]
```

BackboneJS – Collection Sync

Description

It uses **Backbone.sync** to persist the state of a collection to the server.

Syntax

```
collection.sync(method, collection, options)
```

Parameters

- **method:** It represents the CRUD operations such as create, read, update and delete.
- **collection:** It contains a set of models to save the data in the collection.
- **options:** It fires success or error message depending on the method succeeded.

Example

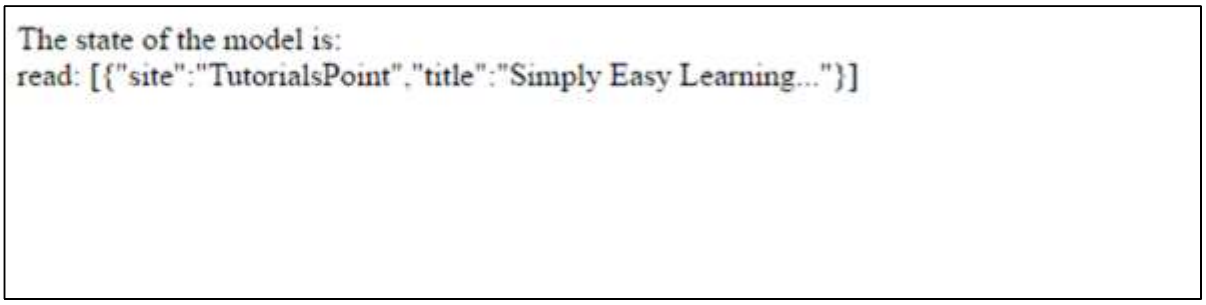
```
<!DOCTYPE html>
   <head>
      <title>Collection Example</title>
         <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js"
type="text/javascript"></script>
   </head>
   <body>
      <script type="text/javascript">
             //The sync() method reads and fetches the model data
         Backbone.sync = function(method, model) {
            document.write("The state of the model is:");
            document.write("<br>");

            //The 'method' specifies state of the model
            document.write(method + ": " + JSON.stringify(model));
         };

         //The 'myval' is collection instance and contains the values which
are to be fetched in the collection
         var myval = new Backbone.Collection({
            site:"TutorialsPoint",
            title:"Simply Easy Learning..."
         });
         //The myval.fetch() method display the model's state by delegating
the sync() method
```

```
         myval.fetch();
      </script>
   </body>
</html>
```

Output

Let us carry out the following steps to see how the above code works:

- Save the above code in the **sync.htm** file.
- Open this HTML file in a browser.



```
The state of the model is:
read: [{"site":"TutorialsPoint","title":"Simply Easy Learning..."}]
```

BackboneJS –Collection Add

Description

It is used to add a model or array of models to the collection.

Syntax

```
collection.add(models,options)
```

Parameters

- **models:** It contains the names of the collection instances, which need to be added in the collection.
- **options:** It includes model types and adds them to the collection instance.

Example

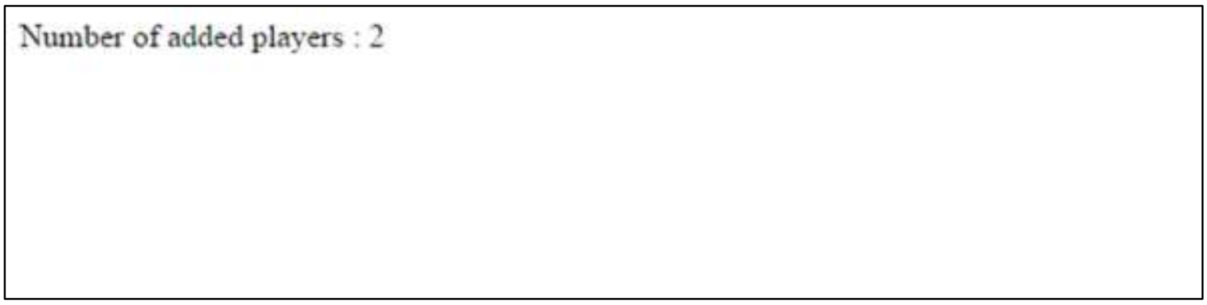
```
<!DOCTYPE html>
   <head>
      <title>Collection Example</title>
         <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>
```

```
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js"
type="text/javascript"></script>

   </head>

   <body>

      <script type="text/javascript">

         //'Player' is a model and contain defualt value for model

         var Player = Backbone.Model.extend({

            defaults: {

               name: "sachin",

               country:"india"

            }

         });


         //The 'PlayersCollection' is a collection instance and model 'Player'
is specified by overriding the 'model' property

         var PlayersCollection = Backbone.Collection.extend({

            model: Player

         });


         //The instances "player1" and "player2" are created for the model
"Player"

         var player1 = new Player({name: "dhoni",  country:"india"});

         var player2 = new Player({name: "raina",  country:"india"});



         //The add() method adds the models 'player1' and 'player2' to the
collection instance 'mycollection'

         var mycollection = new PlayersCollection();

         mycollection.add([player1,player2]);


         //The 'length' property deteremines length of the collection

         document.write('Number of added players : ' + mycollection.length);

      </script>

   </body>

</html>
```

Output

Let us carry out the following steps to see how the above code works:

- Save the above code in the **add.htm** file.
- Open this HTML file in a browser.



```
Number of added players : 2
```

BackboneJS – Collection Remove

Description

It is used to remove a model or array of models from the collection.

Syntax

```
collection.remove(models,options)
```

Parameters

- **models:** It contains the names of the collection instances, which need to be removed from the collection.
- **options:** It includes the model type which will be removed from the collection.

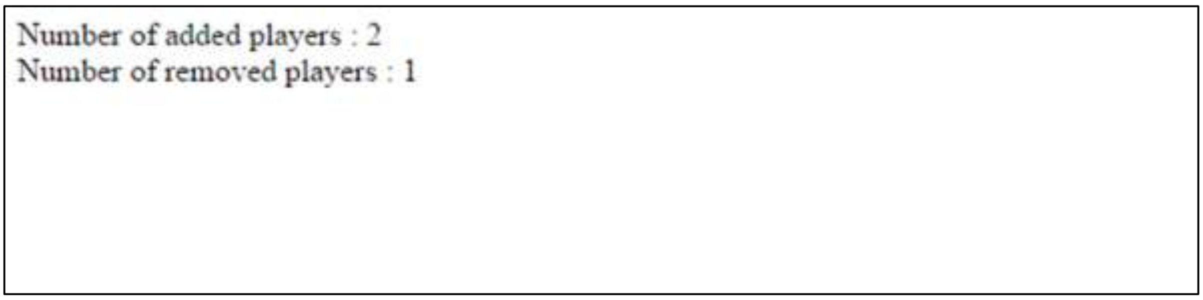
Example

```
<!DOCTYPE html>
   <head>
      <title>Collection Example</title>
         <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js"
type="text/javascript"></script>
   </head>
   <body>
      <script type="text/javascript">
         //'Player' is a model and contain defualt value for model
         var Player = Backbone.Model.extend({
            defaults: {
               name: "sachin",
               country:"india"
            }
         });

         //The 'PlayersCollection' is a collection instance and model 'Player'
is specified by overriding the 'model' property
         var PlayersCollection = Backbone.Collection.extend({
            model: Player
         });

         //The instances "player1" and "player2" are created for the model
"Player"
         var player1 = new Player({name: "dhoni",  country:"india"});
         var player2 = new Player({name: "raina",  country:"india"});

         //The add() method adds the models 'player1' and 'player2' to the
collection instance 'mycollection'
         var mycollection = new PlayersCollection();
         mycollection.add([player1,player2]);

          //The 'length' property deteremines length of the collection
         document.write('Number of added players : ' + mycollection.length);
         document.write("<br>");

         //The remove() method removes the 'player1' model from the collection
         mycollection.remove([player1]);
         document.write('Number of removed players : ' + mycollection.length);
      </script>
   </body>
</html>
```

Output

Let us carry out the following steps to see how the above code works:

- Save the above code in the **remove.htm** file.
- Open this HTML file in a browser.



```
Number of added players : 2
Number of removed players : 1
```

BackboneJS – Collection Reset

Description

It resets the collection and populates with the new array of models or will empty the entire collection.

Syntax

```
collection.reset(models,options)
```

Parameters

- **models:** It contains the names of the collection instances, which need to be reset in the collection.
- **options:** The options include null value to empty the collection.

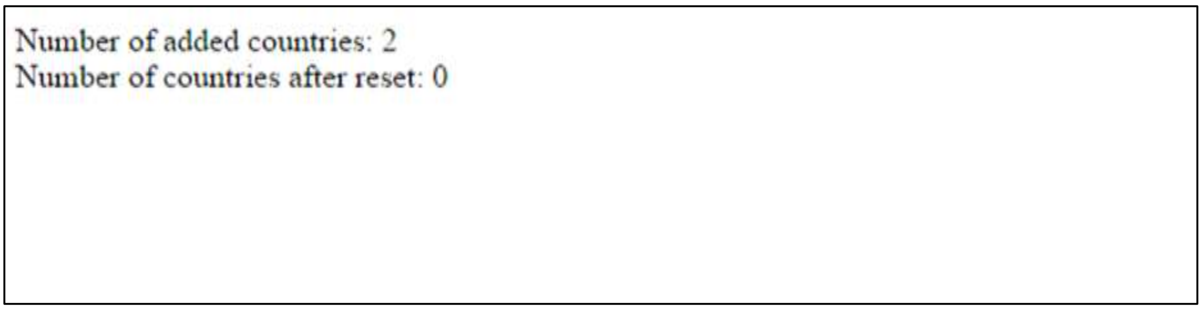
Example

```
<!DOCTYPE html>
   <head>
      <title>Collection Example</title>
         <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js"
type="text/javascript"></script>
   </head>
   <body>
      <script type="text/javascript">
         //The 'C_Name' is a model name and includes default value
         var C_Name = Backbone.Model.extend({
            defaults: {
                country: "sachin"
            }
         });


         //'PlayersCollection' is an instance of collection and model 'C_Name'
is specified by using model property
         var PlayersCollection = Backbone.Collection.extend({
            model: C_Name
         });


         //The 'country1' and 'country2' are the instances of the model
'C_name'
         var country1 = new C_Name({country: "australia"});
         var country2 = new C_Name({country: "england"});


         //Add the model instances to the collection using 'mycollection'
collection instance
         var mycollection = new PlayersCollection();
         mycollection.add([country1,country2]);



         //The 'length' property defines length of the collection
         document.write('Number of added countries: ' + mycollection.length);
         document.write("<br>");


         //Here, the reset() method resets the collection otherwise empties
the collection
         mycollection.reset();
         document.write('Number of countries after reset: ' +
mycollection.length);
      </script>
   </body>
</html>
```

Output

Let us carry out the following steps to see how the above code works:

- Save the above code in the **reset.htm** file.
- Open this HTML file in a browser.



```
Number of added countries: 2
Number of countries after reset: 0
```

BackboneJS – Collection Set

Description

It is used to update the collection with a set of items in a model. If any new model is found, the items will be added to that model.

Syntax

```
collection.set(models,options)
```

Parameters

- **models:** It includes an instance of the collection along with the values to be set in the collection.
- **options:** It includes parameters such as id, name, etc., to set the values in the collection.

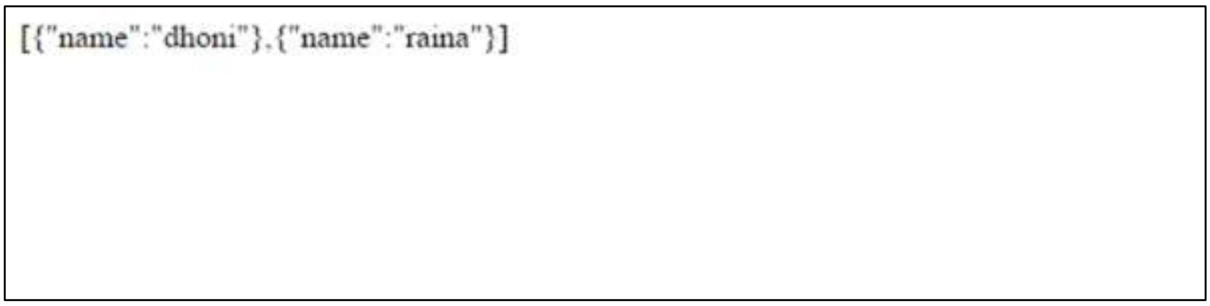
Example

```
<!DOCTYPE html>
   <head>
      <title>Collection Example</title>
         <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js"
type="text/javascript"></script>
   </head>
   <body>
      <script type="text/javascript">
         //Here the model name is 'Player' and contains default value
         var Player = Backbone.Model.extend({
            defaults: {
               name: 'sachin'
            },
         });
         //'PlayersCollection' is an instance of collection
         var PlayersCollection = Backbone.Collection.extend({
            model: Player  //model 'Player' is specified by using model
property
         });
         var player1 = new Player({ name: "dhoni" });  //'player1' is instance
of the model
         var mycollection = new PlayersCollection();   //'mycollection' is
instance of the collection
         mycollection.add(player1);                    //adding model instance
'player1' along with value to the collection
         //The set() method update the 'player1' model by passing this value
in the collection
         mycollection.set([player1, { name: "raina" }]);
         document.write(JSON.stringify(mycollection.toJSON()));
      </script>
   </body>
</html>
```

Output

Let us carry out the following steps to see how the above code works:

- Save the above code in the **set.htm** file.
- Open this HTML file in a browser.



```
[{"name":"dhoni"},{"name":"raina"}]
```

BackboneJS – Collection Get

Description

It is used to retrieve the model from a collection by using the **id** or **cid**.

Syntax

```
collection.get(id)
```

Parameters

- **id:** It is used to get the model from the collection.

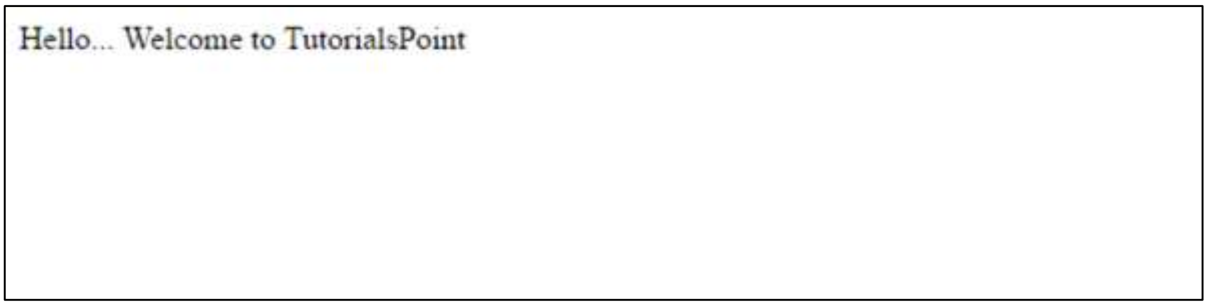
Example

```
<!DOCTYPE html>
   <head>
      <title>Collection Example</title>
         <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js"
type="text/javascript"></script>
   </head>
   <body>
      <script type="text/javascript">
         var MyCollection = new Backbone.Collection();

         //The on() method binds callback function to 'MyCollection' instance
and invokes whenever an event triggers
         MyCollection.on("change:title", function(model) {

            //When event triggers, it retrieves the title from the collection
            document.write("Hello... " + model.get('title'));
         });
         MyCollection.add({id: 2});  //adds id value as 2 in the collection
         //The 'myvalues' object gets the 'MyCollection' with id = 2
         var myvalues = MyCollection.get(2);

         //Sets the value for the title
         myvalues.set('title', 'Welcome to TutorialsPoint');
      </script>
   </body>
</html>
```

Output

Let us carry out the following steps to see how the above code works:

- Save the above code in the **get.htm** file.
- Open this HTML file in a browser.


Hello... Welcome to TutorialsPoint

BackboneJS – Collection At

Description

It is used to retrieve the model from a collection by using the specified index.

Syntax

```
collection.at(index)
```

Parameters

- **index:** It is an index position where we could get the model from a collection.

Example

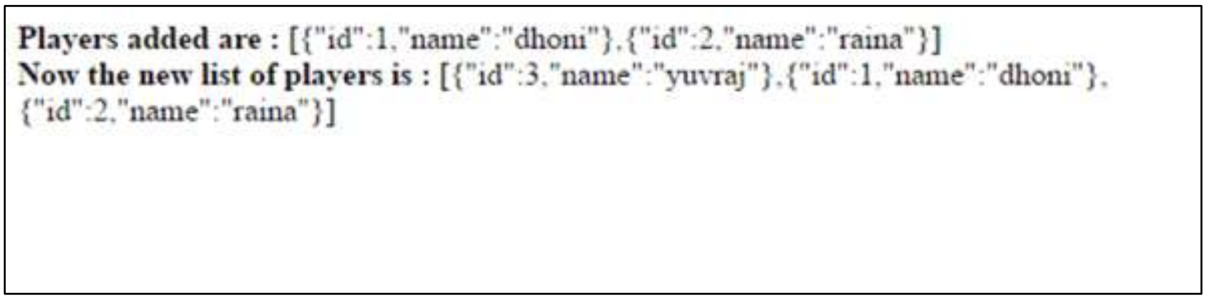
```
<!DOCTYPE html>
   <head>
      <title>Collection Example</title>
         <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js"
type="text/javascript"></script>
```

```
   </head>
   <body>
      <script type="text/javascript">
         //The model name is 'Player' and contains default values
         var Player = Backbone.Model.extend({
            defaults: {
               id:"",
               name: ""
            }
         });

         //'PlayersCollection' is an instance of the collection
         var PlayersCollection = Backbone.Collection.extend({
            model: Player  //model 'Player' is specified by using model
property
         });
         var player1 = new Player({id:1, name: "dhoni" });
         var player2 = new Player({id:2, name: "raina"});

           //The add() method adds the models 'player1' and 'player2' to the
collection instance 'mycollection'
         var mycollection = new PlayersCollection();
         mycollection.add([player1,player2]);
         document.write('<b>Players added are :</b> ' +
JSON.stringify(mycollection.toJSON()));
         var player3 = new Player({id:3, name: "yuvraj" });

         //Here, adding the model 'player3' at 0th index of the collection
         mycollection.add(player3,{at:0});

         //display all the models added. player3 will be added at the 0th
position
         document.write('<br><b>Now the new list of players is :</b> ' +
JSON.stringify(mycollection.toJSON()));
      </script>
   </body>
</html>
```

Output

Let us carry out the following steps to see how the above code works:

- Save the above code in the **at.htm** file.
- Open this HTML file in a browser.



Players added are : [{"id":1,"name":"dhoni"},{"id":2,"name":"raina"}]
Now the new list of players is : [{"id":3,"name":"yuvraj"},{"id":1,"name":"dhoni"},{"id":2,"name":"raina"}]

BackboneJS – Collection Push

Description

It is similar to the add() method. It adds a model at the end of a collection.

Syntax

```
collection.push(models, options)
```

Parameters

- **models:** It contains the names of the collection instances, which are pushed in the collection.
- **options:** It includes the model types and adds them to the collection instance.

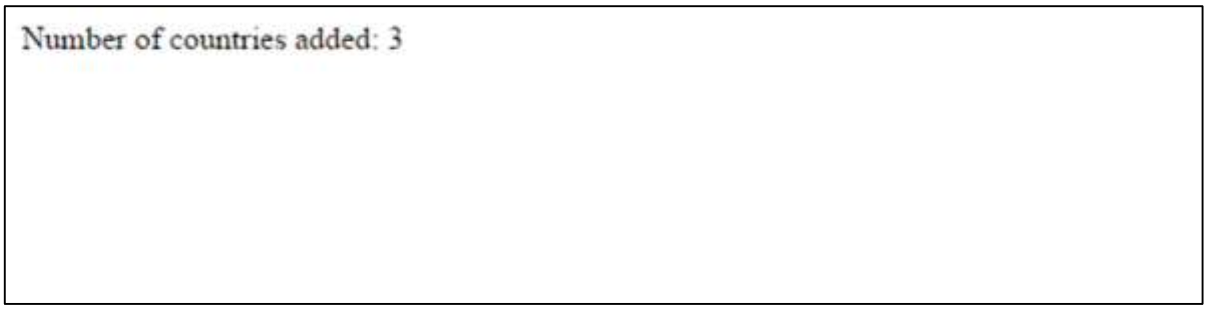
Example

```
<!DOCTYPE html>
   <head>
      <title>Collection Example</title>
         <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js"
type="text/javascript"></script>
   </head>
   <body>
      <script type="text/javascript">
         //The 'Player' is a model name and includes the default values
         var Player = Backbone.Model.extend({
            defaults: {
               c_id:"",
               country: ""
            }
         });

         //The 'PlayersCollection' is a collection instance and model 'Player'
is specified by using model property
         var PlayersCollection = Backbone.Collection.extend({
            model: Player
         });

         //The 'country1', 'country2' and 'country3' are instances of the
model 'Player'
         var country1 = new Player({c_id:1, country: "australia" });
         var country2 = new Player({c_id:2, country: "england"});
         var country3 = new Player({c_id:3, country: "india"});
         var mycollection = new PlayersCollection();

         //Here, the push() method adds the above models to the collection
         mycollection.push(country1);

         mycollection.push(country2);
         mycollection.push(country3);

         //'length' property defines total number of models in the collection
         document.write('Number of countries added: ' + mycollection.length);
      </script>
   </body>
</html>
```

Output

Let us carry out the following steps to see how the above code works:

- Save the above code in the **push.htm** file.
- Open this HTML file in a browser.



```
Number of countries added: 3
```

BackboneJS – Collection Pop

Description

It is similar to the **remove()** method which takes the array of models and remove the models from the collection.

Syntax

```
collection.pop(models, options)
```

Parameters

- **models:** It contains the names of the collection instances, which are needed to be popped out from the collection.
- **options:** It includes the model type which will be removed from the collection.

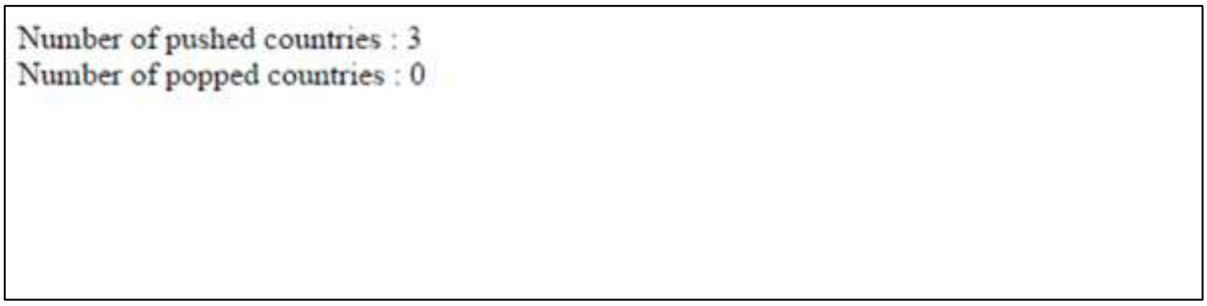
Example

```
<!DOCTYPE html>
   <head>
      <title>Collection Example</title>
         <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js"
type="text/javascript"></script>
   </head>
   <body>
      <script type="text/javascript">
         //The 'Player' is a model name and includes the default values
         var Player = Backbone.Model.extend({
            defaults: {
               c_id:"",
               country: ""
            }
         });

         //The 'PlayersCollection' is a collection instance and model 'Player'
is specified by using model property
         var PlayersCollection = Backbone.Collection.extend({
            model: Player
         });

         //The 'country1', 'country2' and 'country3' are instances of the model
'Player'
         var country1 = new Player({c_id:1, country: "australia" });
         var country2 = new Player({c_id:2, country: "england"});
         var country3 = new Player({c_id:3, country: "india"});
         var mycollection = new PlayersCollection();

         //Here, the push() method adds the above models to the collection
         mycollection.push(country1);

         mycollection.push(country2);
         mycollection.push(country3);
         document.write('Number of pushed countries : ' + mycollection.length);

        document.write("<br>");

         //The pop() method removes the models from the collection
         mycollection.pop(country1);
         mycollection.pop(country2);
         mycollection.pop(country3);
         document.write('Number of popped countries : ' + mycollection.length);
      </script>
   </body>
</html>
```

Output

Let us carry out the following steps to see how the above code works:

- Save the above code in the **pop.htm** file.
- Open this HTML file in a browser.



```
Number of pushed countries : 3
Number of popped countries : 0
```

BackboneJS – Collection Unshift

Description

It is used to add the specified model at the beginning of a collection.

Syntax

```
collection.unshift(models, options)
```

Parameters

- **models:** It contains the names of the collection instances, which is to be added at the beginning of a collection.
- **options:** It includes the model types and adds them to the collection instance.

Example

```
<!DOCTYPE html>
   <head>
      <title>Collection Example</title>
         <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js"
type="text/javascript"></script>
   </head>
   <body>
      <script type="text/javascript">
```

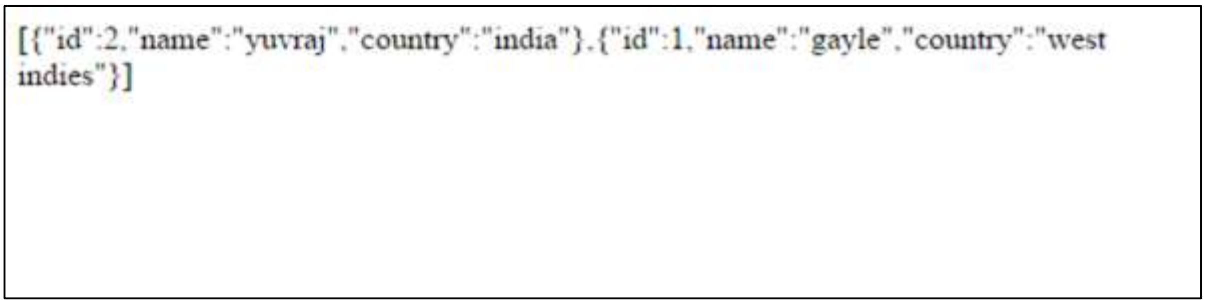
```
          //'Player' is a model and contains defualt values for the model
          var Player = Backbone.Model.extend({
            defaults: {
               name: 'sachin',
               country: 'india'
             }
          });

          //'Players' is an instance of the collection
          var Players = Backbone.Collection.extend({
              model: Player  //model 'Player' is specified by using model
property
          });

          //Here, instantiating models along with "new" keyword and store them
in the collection instance
          var player1 = new Player({ id: 1, name: 'gayle', country: 'west
indies'});
          var player2 = new Player({ id: 2, name: 'yuvraj', country:
'india'});
          var teamArray = [player1, player2];



          //The unshift() method adds the 'player2' model to the beginning of
the collection
          teamArray.unshift(player2);

          //Instantiate new collection by passing in an array of models
          var players = new Players(teamArray);
          document.write(JSON.stringify(players));
      </script>
   </body>
</html>
```

Output

Let us carry out the following steps to see how the above code works:

- Save the above code in the **unshift.htm** file.

- Open this HTML file in a browser.



```
[{"id":2,"name":"yuvraj","country":"india"},{"id":1,"name":"gayle","country":"west
indies"}]
```

BackboneJS – Collection Shift

Description

It removes the first item from the collection.

Syntax

```
collection.shift(options)
```

Parameters

- **options:** It includes the model type which will be removed from the collection.

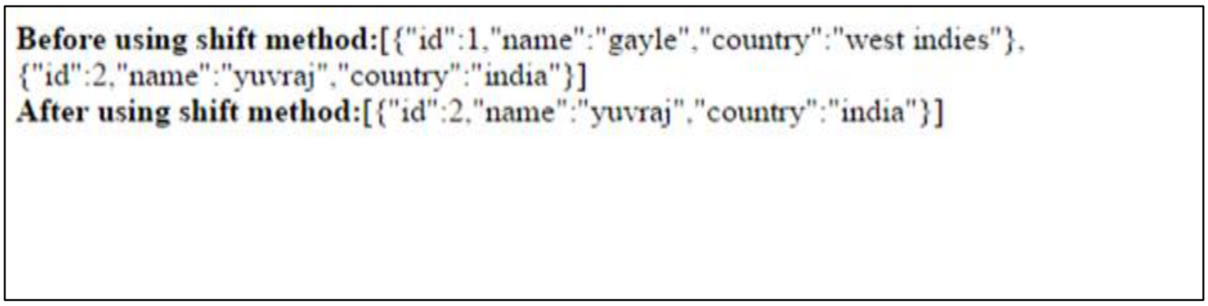
Example

```
<!DOCTYPE html>
   <head>
      <title>Collection Example</title>

         <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js"
type="text/javascript"></script>
   </head>
   <body>
      <script type="text/javascript">
          //'Player' is a model and contains defualt values for the model
          var Player = Backbone.Model.extend({
            defaults: {
               name: 'sachin',
                country: 'india'
              }
          });

          //'Players' is an instance of the collection
          var Players = Backbone.Collection.extend({
              model: Player
          });

          //Instantiate the models along with "new" keyword and store them in
the collection instance
          var player1 = new Player({ id: 1, name: 'gayle', country: 'west
indies'});
          var player2 = new Player({ id: 2, name: 'yuvraj', country: 'india'});
          var teamArray = [player1, player2];

          document.write("<b>Before using shift
method:</b>",JSON.stringify(teamArray));
          //The shift() method removes the model 'player1' from the collection
          teamArray.shift();

          //The 'players' is the collection instance and contains array of
models
          var players = new Players(teamArray);

          document.write("<br><b>After using shift
method:</b>",JSON.stringify(players));
      </script >
   </body>
</html>
```

Output

Let us carry out the following steps to see how the above code works:

- Save the above code in the **shift.htm** file.
- Open this HTML file in a browser.


Before using shift method:[{"id":1,"name":"gayle","country":"west indies"},{"id":2,"name":"yuvraj","country":"india"}]
After using shift method:[{"id":2,"name":"yuvraj","country":"india"}]

BackboneJS – Collection Slice

Description

It returns the shallow copy of the elements from the collection model.

Syntax

```
collection.slice(begin, end)
```

Parameters

- **begin:** It specifies where to begin the extraction of elements.
- **end:** It specifies where to end the extraction of elements.

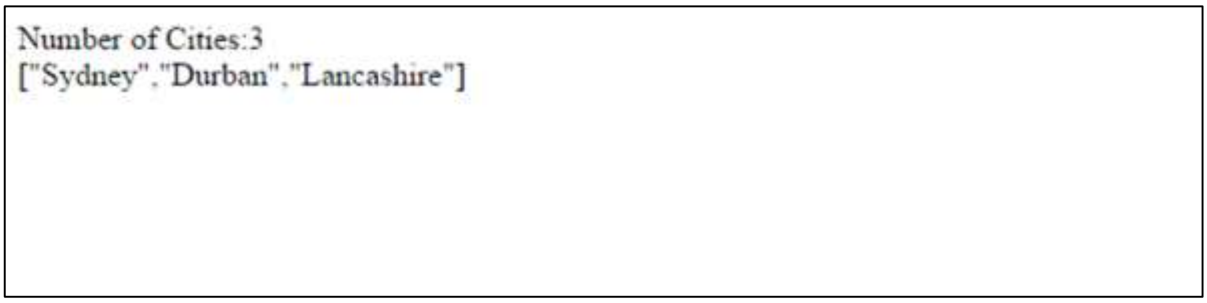
Example

```
<!DOCTYPE html>
   <head>
      <title>Collection Example</title>
         <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js"
type="text/javascript"></script>
   </head>
   <body>
      <script type="text/javascript">
         var City = Backbone.Model.extend();


         //The variable "Cities" contains the elements which are to be
displayed when 'slice()' method is called
         var Cities =['NewYork', 'Sydney', 'Durban', 'Lancashire', 'Kent'];
         //The collection instance "cities" uses the "Cities" variable to
represent the model elements in the collection
         var cities = new Backbone.Collection(Cities,{
            model:City
         });


         //The slice(1,4) method begins the element extraction at index "1"
and ends the extraction of elements up to the index "4"
         var mycity = Cities.slice(1,4);


         //The 'length' property specifies the total number of sliced elements
between 1 and 4
         document.write("Number of Cities:", +mycity.length);
         document.write("<br>");
         document.write(JSON.stringify(mycity));
      </script>
   </body>
</html>
```

Output

Let us carry out the following steps to see how the above code works:

- Save the above code in the **slice.htm** file.
- Open this HTML file in a browser.



```
Number of Cities:3
["Sydney","Durban","Lancashire"]
```

BackboneJS – Collection Length

Description

It is a property that counts the number of models in the collection.

Syntax

```
collection.length
```

Example

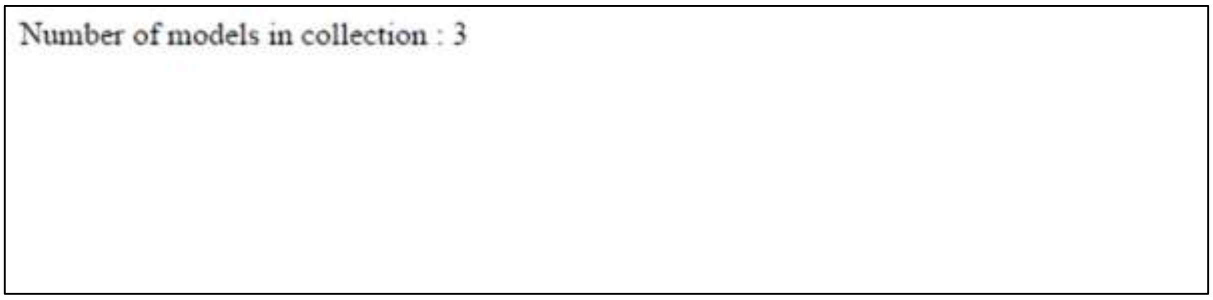
```
<!DOCTYPE html>
   <head>
      <title>Collection Example</title>
         <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js"
type="text/javascript"></script>
   </head>
   <body>
      <script type="text/javascript">
         //'Cities' is a model and contains defualt value for the model
         var Cities = Backbone.Model.extend({
            defaults: {
               city: "Hyderabad"
            }
         });

         //'CityCollection' is an instance of the collection
         var CityCollection = Backbone.Collection.extend({
            model: Cities  //model 'Cities' is specified by using model
property
         });

         //The add() method adds the models 'city1', 'city2' and 'city3' to
the collection instance 'mycollection'
         var city1 = new Cities({city: "Banglore"});
         var city2 = new Cities({city: "Delhi"});
         var city3 = new Cities({city: "Mumbai"});
         var mycollection = new CityCollection();
         mycollection.add([city1,city2,city3]);
         document.write('Number of models in collection : ' +
mycollection.length);
      </script>
   </body>
```

```
</html>
```

Output

Let us carry out the following steps to see how the above code works:

- Save the above code in the **length.htm** file.
- Open this HTML file in a browser.



```
Number of models in collection : 3
```

BackboneJS – Collection Comparator

Description

There is no comparator for a collection. To maintain the collection in a sorted order, we use the comparator property.

Syntax

```
collection.comparator
```

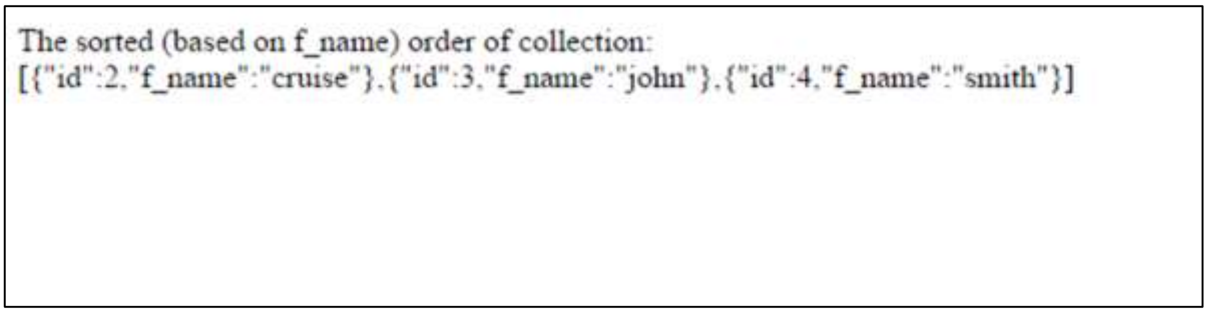
Example

```
<!DOCTYPE html>
   <head>
      <title>Collection Example</title>
         <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js"
type="text/javascript"></script>
   </head>
   <body>
      <script type="text/javascript">
         var MyModel = Backbone.Model.extend();    //'LangNames' is a model name
         //'mydata' variable contains values to be displayed in sorted order
         var mydata = [
            {id:4, f_name: 'smith'},
            {id:2, f_name: 'cruise'},
            {id:3, f_name: 'john'}
         ];

         //'myval' is a collection instance and includes array of values stored
in 'mydata' variable
         var myval = new Backbone.Collection(mydata,{
            model:MyModel,   //The model 'MyModel' is specified by overriding
the "model" property
            comparator: 'f_name'   //The 'comparator' maintain the collection
in sorted order
         });

         //Here, displaying the array of values using collection instance
'myval' and 'models' method
          document.write("The sorted (based on f_name) order of collection:
",
            JSON.stringify(myval)

         );

      </script>
   </body>
</html>
```

Output

Let us carry out the following steps to see how the above code works:

- Save the above code in the **comparator.htm** file.
- Open this HTML file in a browser.



```
The sorted (based on f_name) order of collection:
[{"id":2,"f_name":"cruise"},{"id":3,"f_name":"john"},{"id":4,"f_name":"smith"}]
```

BackboneJS – Collection Sort

Description

The collection sort command sorts the items in the collection and uses the **comparator** property to sort the items.

Syntax

```
collection.sort(options)
```

Parameters

- **options:** It contains optional parameters such as true or false to disable or enable the sorting.

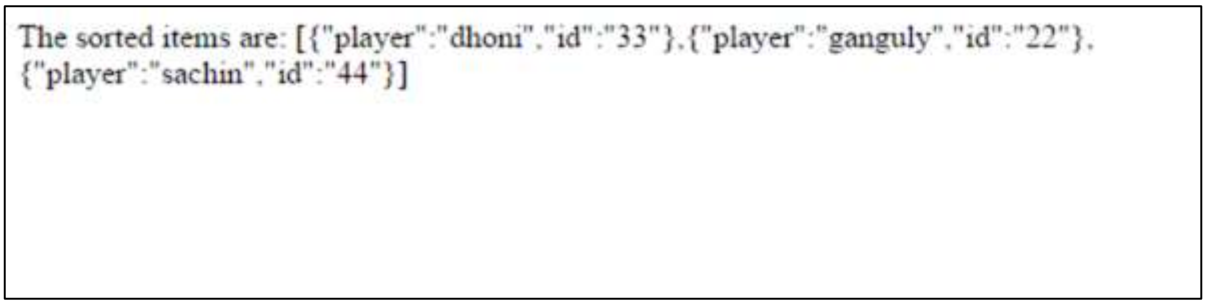
Example

```
<!DOCTYPE html>
   <head>
      <title>Collection Example</title>
         <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>


         <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js"
type="text/javascript"></script>
   </head>
   <body>
      <script type="text/javascript">
         var Player = Backbone.Model.extend();  //'Player' is a model name


         //The variable 'Players' contains the items which are to be displayed
in the ascending order
         var Players = [
            {player: 'sachin',id: '44'},
            {player: 'ganguly',id: '22'},
            {player: 'dhoni',id:'33'}
          ];
          //'myteam' is a collection instance and model 'Player' is specified
using model property
          var myteam = new Backbone.Collection(Players,{
             model:Player,

             //The comparator method is used to maintain the collection in
sorted order
             comparator: 'player'
          });

          //display the sorted records, records are sorted based on player as
we have set comparator to 'player'
          document.write("The sorted items are:
",JSON.stringify(myteam.toJSON()));
      </script>
   </body>
</html>
```

Output

Let us carry out the following steps to see how the above code works:

- Save the above code in the **sort.htm** file
- Open this HTML file in a browser.



```
The sorted items are: [{"player":"dhoni","id":"33"},{"player":"ganguly","id":"22"},
{"player":"sachin","id":"44"}]
```

BackboneJS – Collection Pluck

Description

It retrieves the attributes from each model in the collection.

Syntax

```
collection.pluck(attribute)
```

Parameters

- **attribute:** It represents the property of a defined model.

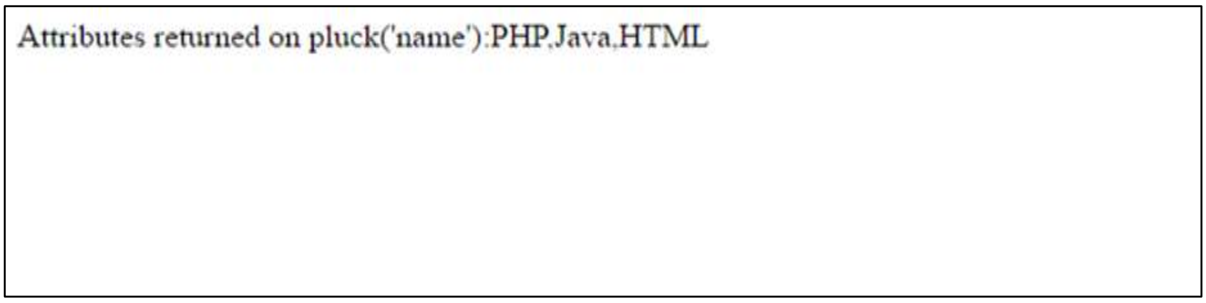
Example

```
<!DOCTYPE html>
   <head>
      <title>Collection Example</title>
         <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js"
type="text/javascript"></script>
   </head>
   <body>
      <script type="text/javascript">
         var LangNames  = Backbone.Model;          //'LangNames' is a model name
         var langauges = new Backbone.Collection;  //'langauges' is a
collection instance

         //The collection instance used with comparator() method to compare
the items and display them in descending order

         langauges.comparator = function(value1, value2) {

            //The comparator returns -1, when the first model should display
before the second model
            if (value1.get('name') > value2.get('name')) return -1;

            //It returns -1, when the first model should display after the
second model
            if (value2.get('name') > value1.get('name')) return 1;
            return 0;
         };
         langauges.add(new LangNames({name: "Java"}));
         langauges.add(new LangNames({name: "PHP"}));
         langauges.add(new LangNames({name: "HTML"}));
         //The pluck() method returns an attribute, by using "name" attribute
from the model in the collection

         document.write("Attributes returned on pluck('name'):
",langauges.pluck('name'));

      </script>

   </body>

</html>
```

Output

Let us carry out the following steps to see how the above code works:

- Save the above code in the **pluck.htm** file.
- Open this HTML file in a browser.



```
Attributes returned on pluck('name'):PHP,Java,HTML
```

BackboneJS – Collection Where

Description

It is used to display the model by using the matched attribute in the collection.

Syntax

```
collection.where(attribute)
```

Parameters

- **attribute:** It represents the property of a defined model.

Example

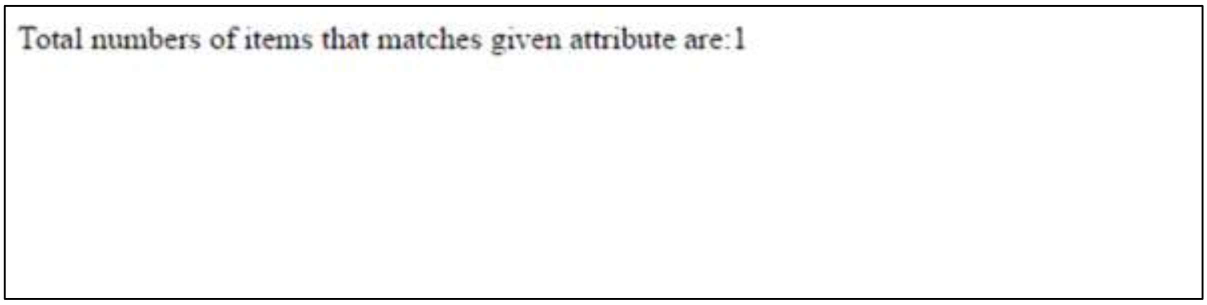
```
<!DOCTYPE html>

   <head>

      <title>Collection Example</title>

         <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>

         <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>
```

```
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js"
type="text/javascript"></script>
   </head>
   <body>
      <script type="text/javascript">
         //'Player' is a model name
         Player = Backbone.Model.extend({
            name: ""
         });

         //The 'PlayersCollection' is an instance of the collection
         PlayersCollection = Backbone.Collection.extend({
            model: Player  //The model 'Player' is specified by overriding the
"model" property of the collection
         });
         var player1 = new Player({ name: "Dravid" });
         var player2 = new Player({ name: "Raina"});
         var player3 = new Player({ name: "Jadeja"});
         var mycollection = new PlayersCollection();

         //The 'player1','player2' and 'player3' are 3 instances added to the
collection by using 'mycollection' instance

         mycollection.add(player1);
         mycollection.add(player2);
         mycollection.add(player3);

         //The where() method returns the model, which contains the name with
"Raina" in the collection
         var myteam = mycollection.where({ name: 'Raina' });
         document.write("Total numbers of items that matches given attribute
are:", +myteam.length);
      </script>
   </body>
</html>
```

Output

Let us carry out the following steps to see how the above code works:

- Save the above code in the **where.htm** file.
- Open this HTML file in a browser.



```
Total numbers of items that matches given attribute are:1
```

BackboneJS – Collection findWhere

Description

It is like the **where** method, but directly returns only the first model in the collection that matches the passed attributes.

Syntax

```
collection.findWhere(attributes)
```

Parameters

- **attributes:** They represent the property of a defined model.

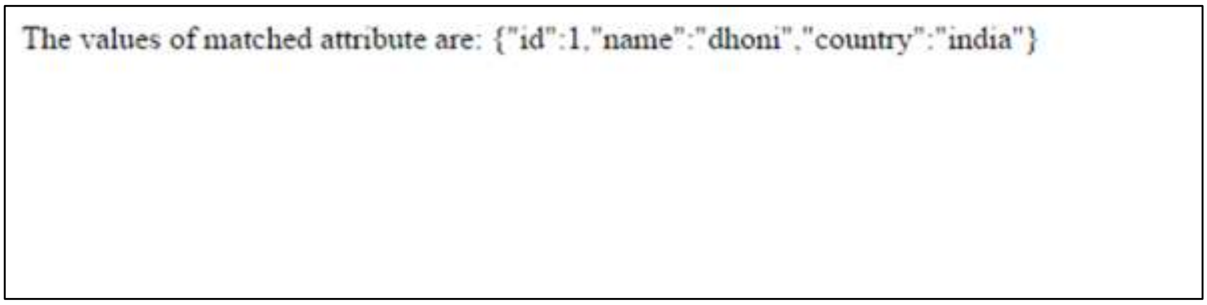
Example

```
<!DOCTYPE html>
   <head>
      <title>Collection Example</title>
         <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js"
type="text/javascript"></script>
   </head>
   <body>
      <script type="text/javascript">
         //'Players' is a model name and containa default values
         var Players = Backbone.Model.extend({
            defaults: {
               id:"",
               name: "",
               country:""
            }
         });

         //The 'PlayersCollection' is an instance of the collection
         var PlayersCollection = Backbone.Collection.extend({
            model: Players   //The model 'Players' is specified by overriding
the "model" property of the collection
         });
         $(function(){
            var mycollection = new PlayersCollection();

            // The set() method to sets the values for 'id', 'name' and
'country' attributes, specified in the model "Players"
            mycollection.set([{id:1, name: 'dhoni', country:'india'},
               {id:2, name:'gayle', country:'west indies'},
               {id:3, name: 'maxwell', country:'australia'},
               {id:4, name: 'duminy', country:'south africa'}

            ]);

            // The findWhere() method finds the model containing with the id
'1'
            var res=mycollection.findWhere({id:1});

            //Display the result in  the JSON format
            document.write("The values of matched attribute are:
",JSON.stringify(res));
         });
      </script>
   </body>
</html>
```

Output

Let us carry out the following steps to see how the above code works:

- Save the above code in the **findwhere.htm** file.
- Open this HTML file in a browser.



```
The values of matched attribute are: {"id":1,"name":"dhoni","country":"india"}
```

BackboneJS – Collection Url

Description

It creates an instance of the collection and returns where the resource is located.

Syntax

```
collection.url()
```

Example

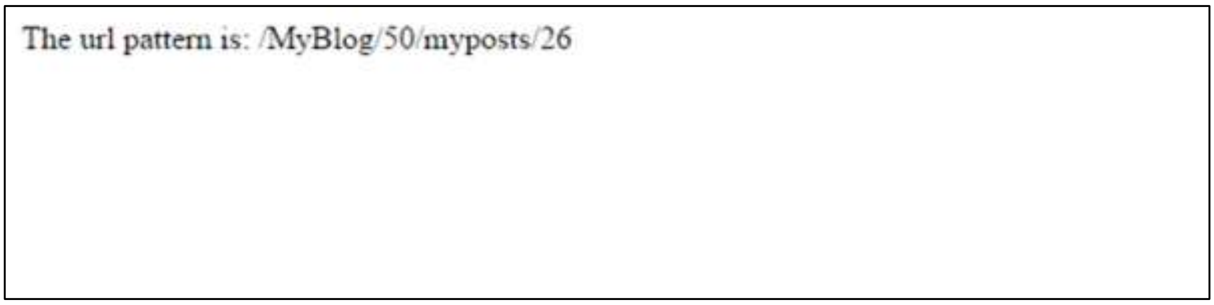
```
<!DOCTYPE html>
   <head>
      <title>Collection Example</title>
         <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js"
type="text/javascript"></script>
   </head>
   <body>
      <script type="text/javascript">
         var MyModel = Backbone.Model.extend({});  ////'MyModel' is a model name


         //The 'MyCollection' is an instance of the collection
         var MyCollection = Backbone.Collection.extend({
            model: MyModel   //The model 'MyModel' is specified by overriding
the "model" property

         });

         //The model "MyBlog" contains default values for 'user' and 'myposts'
attributes

         var MyBlog = Backbone.Model.extend({

            defaults: {

            user: null,

            myposts: []

         },

         initialize: function () {

            var myval = this;


            //Model 'MyModel' gets the 'user' and 'myposts' from the model
'MyBlog' by referring to the current object

            this.MyModel = new MyModel(this.get('user'));

            this.posts = new MyCollection(this.get('myposts'));

            this.posts.url = function () {

               return myval.url() + '/myposts';

            };

         },


         //It enables the url() function by using the id attribute to generate
the URL as "/MyBlog/50/myposts/26"

         urlRoot: '/MyBlog/'

         });

         var attributes = {

            id: 50,

            myposts:[{id: 26}]

         }


         //The model "MyBlog" will access the attributes and display the url
using 'url()' function

         val = new MyBlog(attributes);

         val.posts.each(function (MyModel) {

            document.write("The url pattern is: ",MyModel.url());

         });

      </script>
```

```
   </body>
</html>
```

Output

Let us carry out the following steps to see how the above code works:

- Save the above code in the **url.htm** file.
- Open this HTML file in a browser.



```
The url pattern is: /MyBlog/50/myposts/26
```

BackboneJS – Collection Parse

Description

It returns the collection's data by passing through the response object and represents the data in JSON format.

Syntax

```
collection.parse(response, options)
```

Parameters

- **response:** It returns the array of model attributes to the collection.
- **options:** It includes true as an option which represents data in the JSON format.

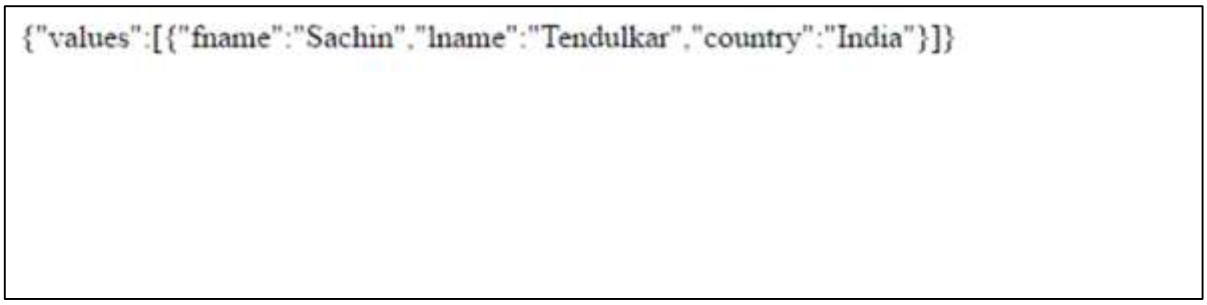
Example

```
<!DOCTYPE html>
   <head>
      <title>Collection Example</title>
         <script src="https://code.jquery.com/jquery-2.1.3.min.js" 
type="text/javascript"></script>
         <script 
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>
         <script 
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js" 
type="text/javascript"></script>
   </head>
   <body>
      <script type="text/javascript">
         //'MyModel' is a model name and  extended using the Backbone.Model class
         var MyModel = Backbone.Model.extend();

         // The variable 'myData' contains the values which are need to be
parsed in the collection
         var myData ={
            "values": [{
               "fname": "Sachin",
               "lname": "Tendulkar",
               "country": "India"
            }]
         };

         //'MyCollection' is a collection name
         var MyCollection = Backbone.Collection.extend({
            model: MyModel,   //The model 'MyModel' is specified by overriding
the 'model' property

            parse : function(response, options){
               document.write(JSON.stringify(response));
            }
         });

         //The collection instance 'myCollection' extracts the values of
'myData'only if parse is set to true
         var mycollection = new MyCollection(myData, { parse: true });
      </script>
   </body>
</html>
```

Output

Let us carry out the following steps to see how the above code works:

- Save the above code in the **parse.htm** file.
- Open this HTML file in a browser.



```
{"values":[{"fname":"Sachin","lname":"Tendulkar","country":"India"}]}
```

BackboneJS – Collection Clone

Description

It returns a new instance of the collection with an identical list of models.

Syntax

```
collection.clone()
```

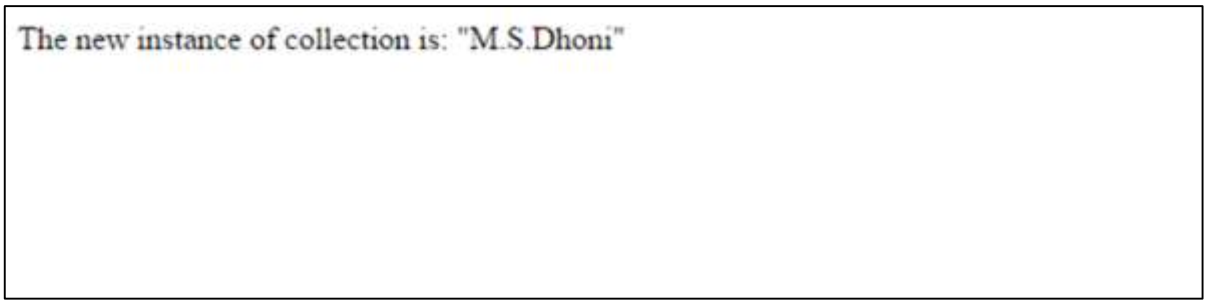
Example

```
<!DOCTYPE html>
   <head>
      <title>Collection Example</title>


         <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js"
type="text/javascript"></script>
   </head>
   <body>
      <script type="text/javascript">
         //'Person' is a model name
         var Person  = Backbone.Model.extend();


         //The model instance 'person' contains 'name' attribute
         var person=new Person({
            name: 'Sachin Tendulkar'
         });
         var MyCollection = Backbone.Collection.extend({
            model: Person   //model is override by specifying the "model"
property of collection class
         });
         var myCollection = new MyCollection();


         // The clone() method uses get method to retrieve the 'name'
attribute
         var details = myCollection.clone(person.get('name'));


         //The variable 'details' assigns the value for 'name' as 'M.S.Dhoni'
         details.name="M.S.Dhoni";
         document.write("The new instance of collection is:
",JSON.stringify(details.name));
      </script>
   </body>
</html>
```

Output

Let us carry out the following steps to see how the above code works:

- Save the above code in the **clone.htm** file.
- Open this HTML file in a browser.



```
The new instance of collection is: "M.S.Dhoni"
```

BackboneJS – Collection Fetch

Description

It extracts the data from the model in the collection using the **sync** method.

Syntax

```
collection.fetch(options)
```

Parameters

- **options:** It takes the **success** and **error** callbacks which will be both passed as arguments.

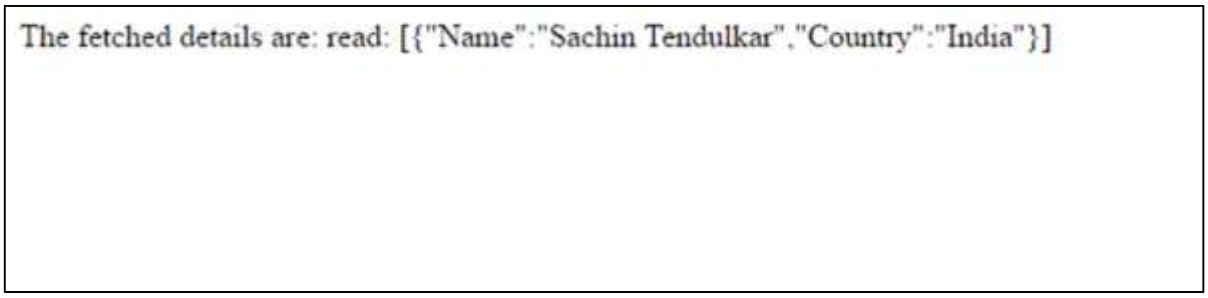
Example

```
<!DOCTYPE html>
   <head>
      <title>Collection Example</title>
         <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js"
type="text/javascript"></script>
   </head>
   <body>
      <script type="text/javascript">
         //The sync() method represents state of a model
         Backbone.sync = function(method, model) {

            //The 'method' gives 'read' state of the model by representing data
in JSON format
            document.write("The fetched details are: ",method + ": " +
JSON.stringify(model));
         };

         //The collection instance 'details' contains the values which are to
be fetched in the collection
         var details = new Backbone.Collection({
            Name:"Sachin Tendulkar",
            Country:"India"
         });

         //This will display the model state by delegating the 'sync()' method
         details.fetch();
      </script>
   </body>
</html>
```

Output

Let us carry out the following steps to see how the above code works:

- Save the above code in the **fetch.htm** file.
- Open this HTML file in a browser.



```
The fetched details are: read: [{"Name":"Sachin Tendulkar","Country":"India"}]
```

BackboneJS – Collection Create

Description

It creates the new instance of the model in the collection.

Syntax

```
collection.create(attributes,options)
```

Parameters

- **attributes:** They represent the property of a defined model.
- **options:** It takes the id, name, etc., parameters to create the collection instance.

Example

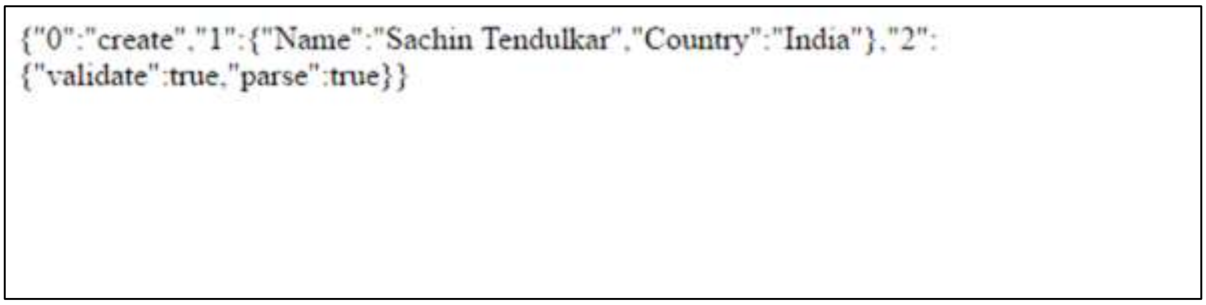
```
<!DOCTYPE html>
   <head>
      <title>Collection Example</title>
         <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js"
type="text/javascript"></script>
   </head>
   <body>
      <script type="text/javascript">
```

```
         //The model 'ModelDemo' uses the sync method to display the model
state such as create, read, update etc
         var ModelDemo = Backbone.Model.extend({
            sync : function (method, model, options) {
               document.write(JSON.stringify(arguments));
            }
         });

         //'CollectionDemo' is an instance of the collection
        var CollectionDemo = Backbone.Collection.extend({
           model : ModelDemo   //The model 'ModelDemo' is specified by
overriding the 'model' property
        });

        //'ViewDemo' is the name of the view
        var ViewDemo = Backbone.View.extend({

        //The instance of the collection 'collectiondemo' is created within the
'initialize()' function
           initialize : function () {
              var collectiondemo = new CollectionDemo();

              collectiondemo.create({
                 Name:"Sachin Tendulkar",
                 Country:"India"
              });
           }
        });
        new ViewDemo();  //The view instance 'ViewDemo' is created using the
'new' keyword.
      </script>
   </body>
</html>
```

Output

Let us carry out the following steps to see how the above code works:

- Save the above code in the **create.htm** file.
- Open this HTML file in a browser.



```
{"0":"create","1":{"Name":"Sachin Tendulkar","Country":"India"},"2":
{"validate":true,"parse":true}}
```

Underscore Methods

The following table lists down the **Underscore.js** methods which provides their functionality to be used on the **Backbone.Collection**.

S.No.	Methods & Description
1	**_.each(list, iteratee, [context])** Iterates each of the elements in the collection using the iteratee function.
2	**_.map(list, iteratee, [context])** It maps each value and displays them in a new array of values using the **iteratee** function.
3	**_.reduce(list, iteratee, memo, [context])** It reduces the list of values into a single value and it also known as **inject** and **foldl**.
4	**_.reduceRight(list, iteratee, memo, [context])** It is the right associative version of **reduce**.
5	**_.find(list, predicate, [context])** It finds each value and returns the first one which passes the predicate or test.
6	**_.filter(list, predicate, [context])** It filters each value and returns the array of values which passes the predicate or test.
7	**_.reject(list, predicate, [context])** It returns the rejected elements in the list which do not pass the predicted values.
8	**_.every(list, predicate, [context])** It returns true, if elements in the list pass the predicted values.

9	**_.some(list, predicate, [context])** It returns true, if elements in the list pass the predicted values.
10	**_.contains(list, value, [fromIndex])** It returns true, if a value is present in the list.
11	**_.invoke(list, methodName, *arguments)** It invokes the method name using **methodName()** on each value in the list.
12	**_.max(list, [iteratee], [context])** It specifies the maximum value in the list.
13	**_.min(list, [iteratee], [context])** It specifies the minimum value in the list.
14	**_.sortBy(list, [iteratee], [context])** It returns the sorted elements in the ascending order by using iteratee in the list.
15	**_.groupBy(list, [iteratee], [context])** It divides the collection values into the sets, grouped by using the **iteratee** in the list.
16	**_.shuffle(list)** It returns the shuffled copy of the list.
17	**_.toArray(list)** It defines an array of the list.
18	**_.size(list)** It defines the number of values in the list.
19	**_.first(array, [n])** It specifies the first element of the array in the list.
20	**_.initial(array, [n])** It returns everything, but specifies the last entry of the array in the list.

21	**_.last(array, [n])** It specifies the last element of the array in the list.
22	**_.rest(array, [index])** It defines the remaining elements in the array.
23	**_.without(array, *values)** It returns the values of all instances which are removed in the list.
24	**_.indexOf(array, value, [isSorted])** It returns the value if it is found at a specified index or returns -1, if it is not found.
25	**_.indexOf(array, value, [fromIndex])** It returns the last occurrence of the value in the array or returns -1, if it is not found.
26	**_.isEmpty(object)** It returns true if there are no values in the list.
27	**_.chain(obj)** It returns a wrapped object.

7. BackboneJS – Router

Router is used for routing the client side applications and defines the URL representation of the application's object. A router is required when web applications provide linkable, bookmarkable and shareable URL's for important locations in the app.

The following table lists down the methods which can be used to manipulate the **BackboneJS - Router**:

S.No.	Methods & Description
1	**extend** It extends the backbone's router class.
2	**routes** It defines the URL representation of applications objects.
3	**initialize** It creates a new constructor for the router instantiation.
4	**route** It creates a route for the router.
5	**navigate** It is used to update the URL in the applications.
6	**execute** It is used when a route matches its corresponding callback.

BackboneJS – Router Extend

Description

It extends the backbone's router class and creates a new constructor which inherits from the **Backbone.Router**.

Syntax

```
Backbone.Router.extend(properties, classProperties)
```

Parameters

- **properties:** It provides instance properties for the router class.

- **classProperties:** The class properties are attached to the router's constructor function.

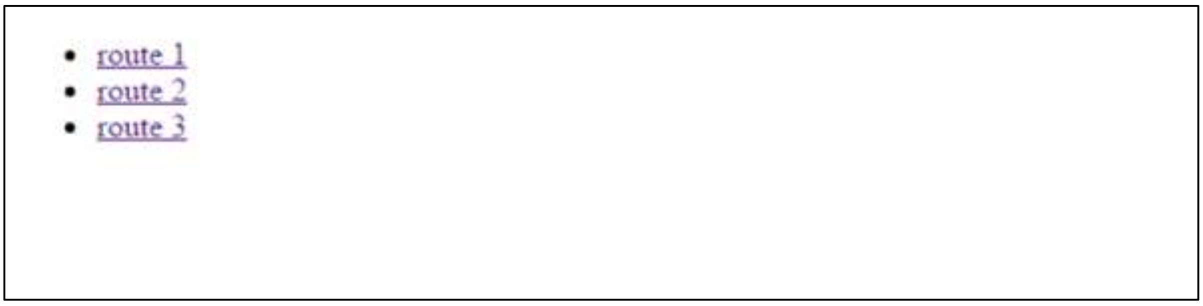
Example

```
<!DOCTYPE html>
   <head>
      <title>Router Example</title>
         <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js"
type="text/javascript"></script>

            <script type="text/javascript">
         //'RouteMenu' is a name of the view class
         var RouteMenu = Backbone.View.extend({
            el: '#routemenu',  //'el' defines which element to be used as the
view reference

            //defines a click event to occur on link
            events: {
               'click a' : 'onClick'
            },

            //After clicking on a link, router calls 'navigate' to update URL
            onClick: function( e ) {
               router.navigate('/');
            }
        });

        //'Router' is a name of the router class
        var Router = Backbone.Router.extend({

        //The 'routes' maps URLs with parameters to functions on your router
           routes: {
              'route/:id' : 'defaultRoute'
           },
        });


        //'routemenu' is an instance of the view class
        var routemenu = new RouteMenu();


        //It starts listening to the routes and manages the history for
bookmarkable URL's
        Backbone.history.start();
     </script>
     </head>
  <body>


     <section id="routemenu">
        <ul>
           <li> <a href="#/route/1">route 1 </a> </li>
           <li> <a href="#/route/2">route 2 </a> </li>
           <li> <a href="#/route/3">route 3 </a> </li>
       </ul>
    </section>
  </body>
</html>
```

Output

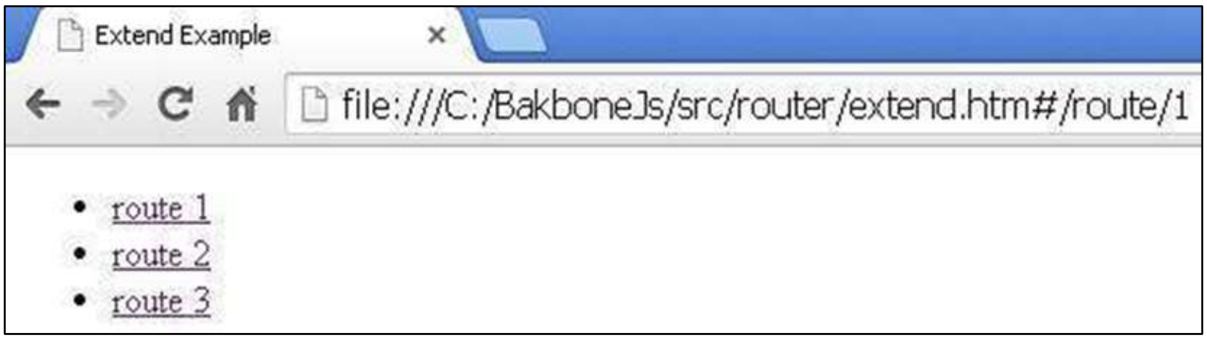
Let us carry out the following steps to see how the above code works:

- Save the above code in the **extend.htm** file.
- Open this HTML file in a browser.



- route 1
- route 2
- route 3

NOTE: The above functionality is related to the address bar. So, when you open the above code in the browser, it will show as follows.



- route 1
- route 2
- route 3

Click here for the demo – (Clicking on this link takes you to – https://www.tutorialspoint.com/backbonejs/src/router/extend.htm)

BackboneJS – Router Routes

Description

It defines the URL representation of the application objects on the router and contains the incoming route value from the URL.

Syntax

```
router.routes
```

Example

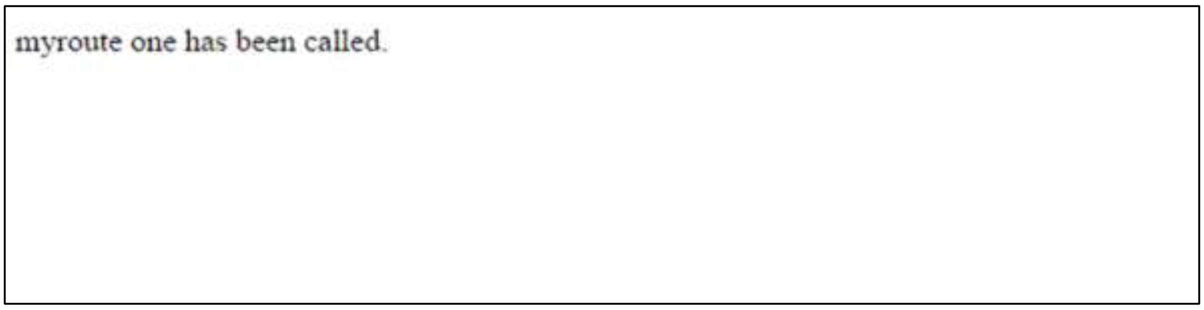
```
<!DOCTYPE html>
   <head>
      <title>Router Example</title>
         <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js"
type="text/javascript"></script>
   </head>
   <body>
      <script type="text/javascript">
          //'Router' is a name of the router class
         var Router = Backbone.Router.extend({


         // The 'routes' maps URLs with parameters to functions on your router
```

```
            routes: {
               '': 'myroute_1',
               'myroute_2': 'myroute_2'
            },

            //After executing the code, it will display this line
            myroute_1: function(){
               document.write("myroute one has been called.");
            },

            //When you enter the #myroute_2 at the end of url, it will display
this line
            myroute_2: function(){
                document.write("myroute two has been called.");
            },
         });
         var appRouter=new Router;   //It is an instantiation of the router

         //It start listening to the routes and manages the history for
bookmarkable URL's
         Backbone.history.start();
     </script>
  </body>
</html>
```

Output

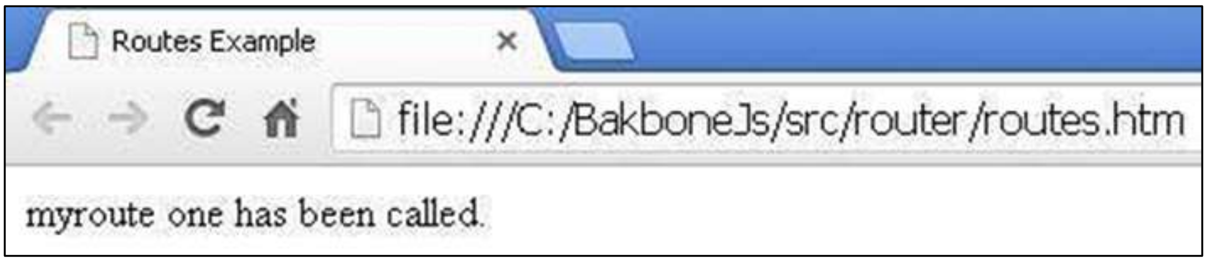
Let us carry out the following steps to see how the above code works:

- Save the above code in the **routes.htm** file.
- Open this HTML file in a browser.



```
myroute one has been called.
```

NOTE: The above functionality is related to the address bar. So, when you open the above code in the browser, it will show as follows.



myroute one has been called.

Click here for the demo (Clicking on this link will take you to – https://www.tutorialspoint.com/backbonejs/src/router/routes.htm)

BackboneJS – Router Initialize

Description

It creates a new constructor for the router instantiation.

Syntax

```
new Router(options)
```

Parameters

- **options:** These are passed to the initialize function.

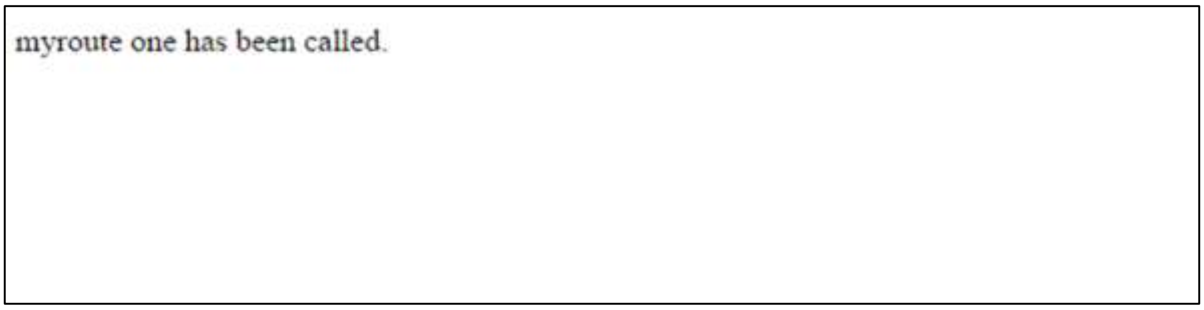
Example

```
<!DOCTYPE html>
   <head>
      <title>Router Example</title>
         <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js"
type="text/javascript"></script>
   </head>
   <body>
      <script type="text/javascript">
         //'Router' is a name of the router class
         var Router = Backbone.Router.extend({


         //The 'routes' maps URLs with parameters to functions on your router
            routes: {
               '': 'myroute_1',
               'myroute_2': 'myroute_2'
            },

            //After executing the code, it will display this line
            myroute_1: function(){
               document.write("myroute one has been called.");
            },

            //When you enter the #myroute_2 at the end of url, it will display
this line
            myroute_2: function(){
                document.write("myroute two has been called.");
            },
         });
         var appRouter=new Router;  //It is an instantiation of the router
using the 'new' keyword

         //It start listening to the routes and manages the history for
bookmarkable URL's
         Backbone.history.start();
     </script>
  </body>
</html>
```

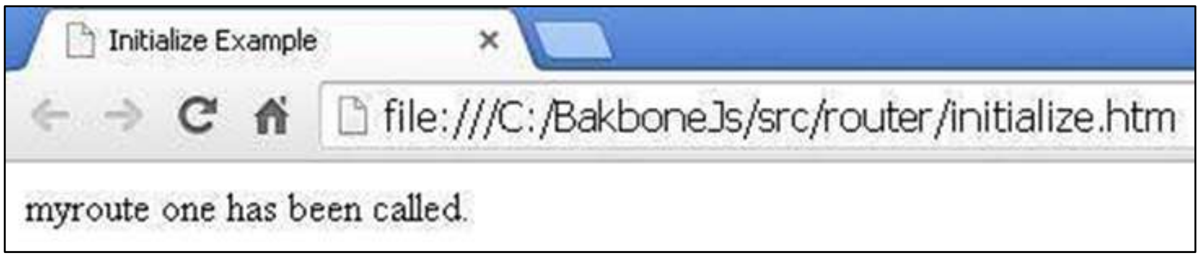
Output

Let us carry out the following steps to see how the above code works:

- Save the above code in the **initialize.htm** file.
- Open this HTML file in a browser.


```
myroute one has been called.
```

NOTE: The above functionality is related to the address bar. So, when you open the above code in a browser, it will show result as follows.



myroute one has been called.

[Click here for the demo](https://www.tutorialspoint.com/backbonejs/src/router/initialize.htm) – (Clicking on this link will take you to – https://www.tutorialspoint.com/backbonejs/src/router/initialize.htm)

BackboneJS – Router Route

Description

It provides a route for the router and appends the router's parameter using a **slash** followed by **colons** and the parameter's name.

Syntax

```
router.route(route, name, [callback])
```

Parameters

- **route:** It may be a routing string or a regular expression.
- **name:** It is the name of a router parameter.
- **callback:** It is the name of the router, if callback argument is omitted.

Example

```
<!DOCTYPE html>

   <head>

      <title>Router Example</title>

         <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>

         <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>

         <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js"
type="text/javascript"></script>

   </head>

      <script type="text/javascript">

         //'RouteMenu' is a name of the view class

         var RouteMenu = Backbone.View.extend({

            el: '#routemenu',   //'el' defines which element to be used as the
view reference
```

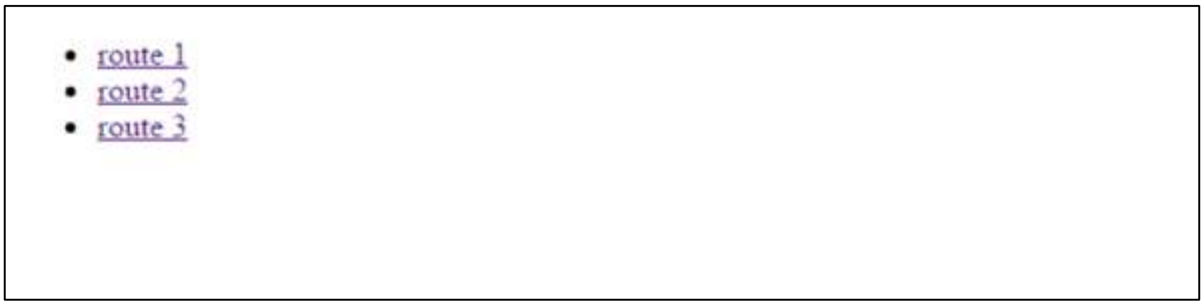
```
            //defines a click event to be occur on link
            events: {
               'click a' : 'onClick'
            },

            //After clicking on a link, router calls 'navigate' to update URL
            onClick: function( e ) {
               router.navigate('/');
            }
        });

        //'Router' is a name of the router class
        var Router = Backbone.Router.extend({

           //The 'routes' maps URLs with parameters to functions on your
router
           routes: {
              'route/:id' : 'defaultRoute'
           },
        });

        //'routemenu' is an instance of the view class
        var routemenu = new RouteMenu();

        //It start listening to the routes and manages the history for
bookmarkable URL's
        Backbone.history.start();
     </script>
  <body>
     <section id="routemenu">
        <ul>
           <li> <a href="#/route/1">route 1 </a> </li>
           <li> <a href="#/route/2">route 2 </a> </li>
           <li> <a href="#/route/3">route 3 </a> </li>
       </ul>
```

```
</section>
  </body>
</html>
```

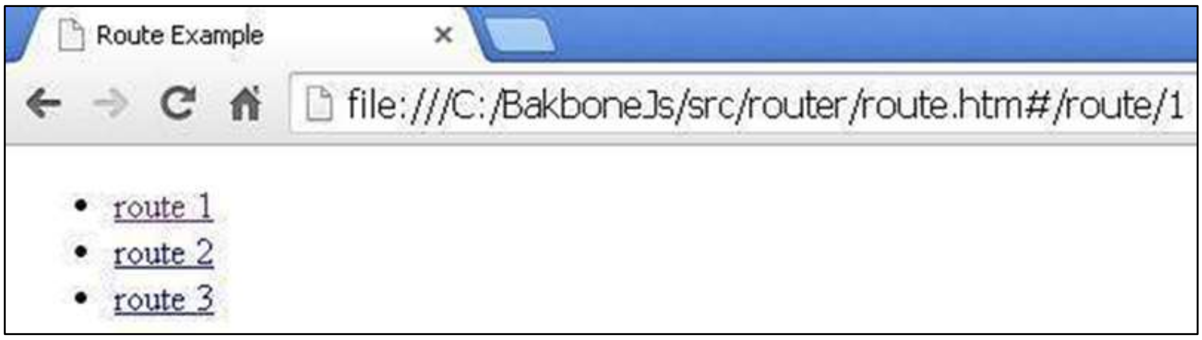
Output

Let us carry out the following steps to see how the above code works:

- Save the above code in the **route.htm** file.
- Open this HTML file in a browser.


- route 1
- route 2
- route 3

NOTE: The above functionality is related to the address bar. So, when you open the above code in the browser, it will show the result as follows.



- route 1
- route 2
- route 3

Click here for the demo – (Clicking on this link takes you to – https://www.tutorialspoint.com/backbonejs/src/router/route.htm)

BackboneJS – Router Navigate

Description

To save the application as a URL, you need to use the navigate method to update the URL.

Syntax

```
router.navigate(fragment, options)
```

Parameters

- **fragment:** It is the name of the parameter in which the url will be displayed after this parameter.

- **options:** The options such as **trigger** and **replace** to call the route function and to update the URL.

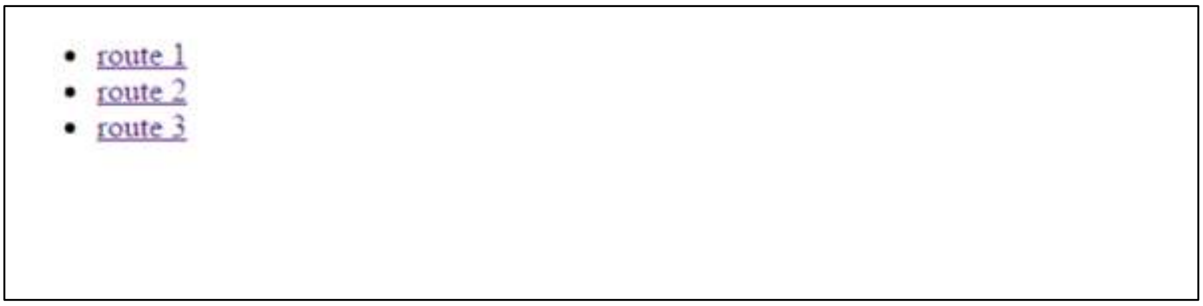
Example

```
<!DOCTYPE html>
   <head>
      <title>Router Example</title>
         <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js"
type="text/javascript"></script>
   </head>
      <script type="text/javascript">
         //'RouteMenu' is a name of the view class
         var RouteMenu = Backbone.View.extend({
            el: '#routemenu',   //'el' defines which element to be used as the
view reference

             //defines a click event to be occur on link
            events: {
               'click a' : 'onClick'
            },

            //After clicking on a link, router calls 'navigate' to update URL
            onClick: function( e ) {

               //Uses the navigate() method save the application as URL
               router.navigate('/');
            }
        });
        var Router = Backbone.Router.extend({

           //The 'routes' maps URLs with parameters to functions on your router
           routes: {
              'route/:id' : 'defaultRoute'
           },
        });

        //'routemenu' is an instance of the view class
        var routemenu = new RouteMenu();

         //It start listening to the routes and manages the history for
bookmarkable URL's
        Backbone.history.start();
     </script>
  <body>
     //It refers to the view class 'RouteMenu' and creates the 3 links which
changes the url when you click on the links
     <section id="routemenu">
        <ul>
           <li> <a href="#/route/1">route 1 </a> </li>
           <li> <a href="#/route/2">route 2 </a> </li>
           <li> <a href="#/route/3">route 3 </a> </li>
       </ul>
    </section>
  </body>
</html>
```

Output

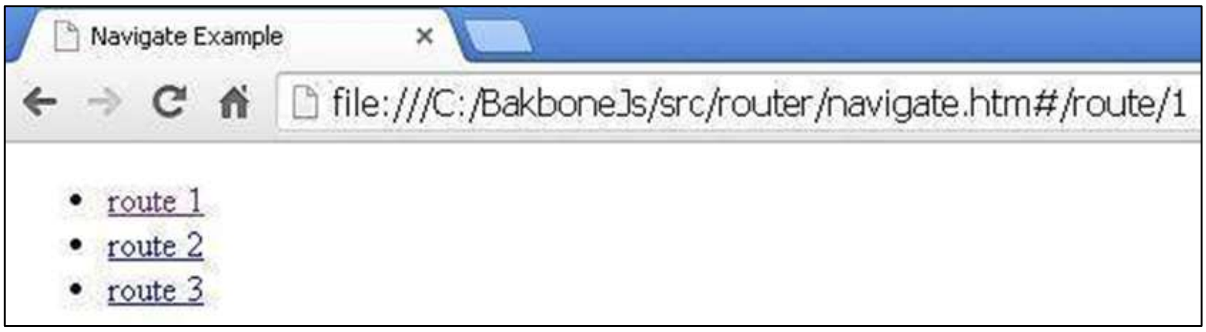
Let us carry out the following steps to see how the above code works:

- Save the above code in the **navigate.htm** file.
- Open this HTML file in a browser.



- route 1
- route 2
- route 3

NOTE: The above functionality is related to the address bar. So, when you will open the above code in the browser, it will show the result as follows.



- route 1
- route 2
- route 3

Click here for the demo – (Clicking on this link takes you to – https://www.tutorialspoint.com/backbonejs/src/router/navigate.htm)

BackboneJS – Router Execute

Description

It is used when a route matches its corresponding callback.

Syntax

```
router.execute(callback, args)
```

Parameters

- **callback:** It executes when there is a match with route.
- **args:** Arguments passed within the execute method.

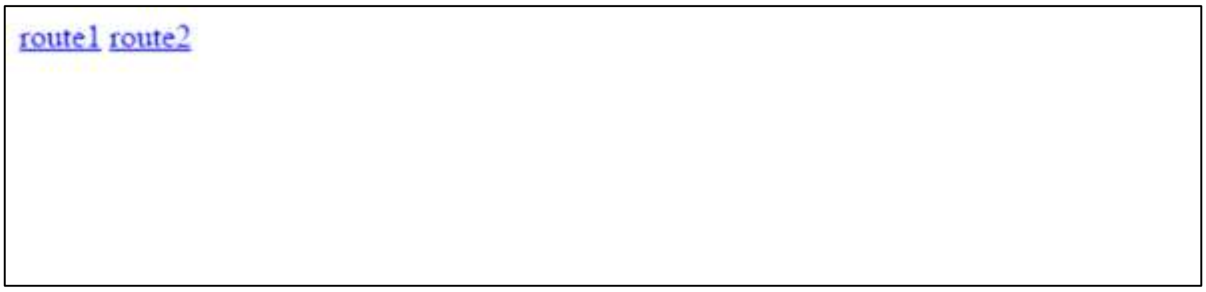
Example

```
<!DOCTYPE html>
   <head>
      <title>Router Example</title>
         <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js"
type="text/javascript"></script>
   </head>
      <script type="text/javascript">
         //'RouteMenu' is a name of the view class
         var Route1 = Backbone.View.extend({
            //Creates the route1 link for the text to be change after
triggerring the click event
            template: '<b>This is route 1</b>',


               //The 'initialize' function creates new constructor for the
router instantiation
               initialize: function () {
                  this.execute();
               },


               //This is called when a route matches its corresponding callback
               execute: function () {
                  this.$el.html(this.template);
               }
         });
         var Route2 = Backbone.View.extend({
            template: '<b>This is route 2</b>',
               initialize: function () {
                  this.execute();
               },
               execute: function () {
                  this.$el.html(this.template);
               }
        });


        //'AppRouter' is a name of the router class
        var AppRouter = Backbone.Router.extend({
           routes: {
              '': 'homeRoute',
              'route/1': 'homeRoute',
              'route/2': 'aboutRoute',
           },
           //When you click on route1, it will navigate to the custom view
class 'Route1'
           homeRoute: function () {
              var route1 = new Route1();
              $("#content").html(route1.el);
           },

           //When you click on route2, it will navigate to the custom view
class 'Route2'
           aboutRoute: function () {
              var route2 = new Route2();
              $("#content").html(route2.el);
           }
       });
       var appRouter = new AppRouter();   //It is an instantiation of the
router


       //It start listening to the routes and manages the history for
bookmarkable URL's
       Backbone.history.start();
     </script>
   <body>
     <div id="navigation">
       <a href="#/route/1">route1</a>
       <a href="#/route/2">route2</a>
     </div>
     <div id="content></div>
   </body>
</html>
```

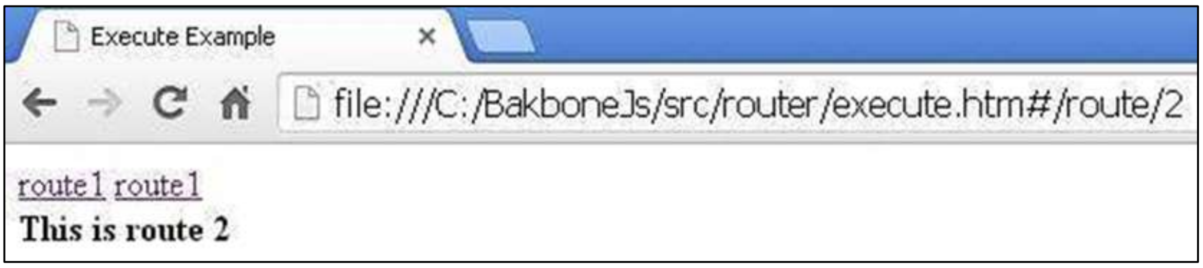
Output

Let us carry out the following steps to see how the above code works:

- Save the above code in the **execute.htm** file.
- Open this HTML file in a browser.


route1 route2

NOTE: The above functionality is related to the address bar. So, when you open the above code in the browser, it will show the result as follows.



route1 route1

This is route 2

Click here for the demo – (Clicking on this link takes you to – https://www.tutorialspoint.com/backbonejs/src/router/execute.htm)

8. BackboneJS – History

It keeps a track of the history, matches the appropriate route, fires callbacks to handle events and enables the routing in the application.

start

This is the only method which can be used to manipulate the **BackboneJS-History**. It starts listening to routes and manages the history for bookmarkable URL's.

Syntax

```
Backbone.history.start(options)
```

Parameters

- **options:** The options include the parameters such as **pushState** and **hashChange** used with history.

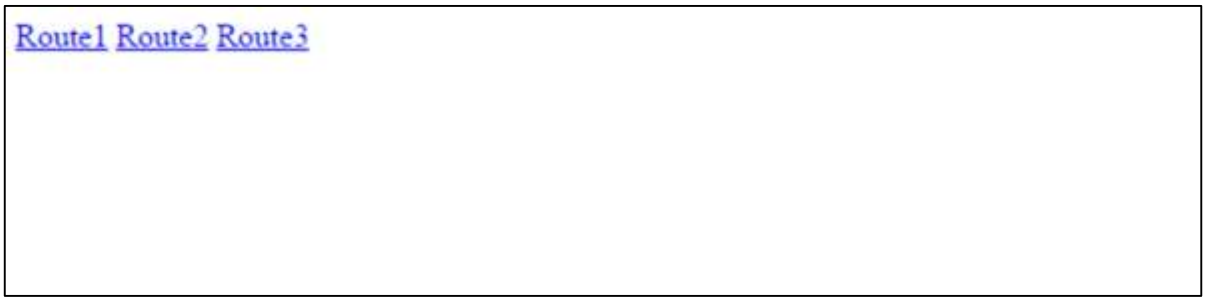
Example

```
<!DOCTYPE html>
   <head>
      <title>History Example</title>
         <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js"
type="text/javascript"></script>
   </head>
   <script type="text/javascript">
       //'Router' is a name of the router class
      var Router = Backbone.Router.extend({

         //The 'routes' maps URLs with parameters to functions on your router
         routes: {
            "myroute" : "myFunc"
         },
         //'The function 'myFunc' defines the links for the route on the
browser
         myFunc: function (myroute) {
            document.write(myroute);
         }
      });

      //'router' is an instance of the Router
      var router = new Router();

      //Start listening to the routes and manages the history for bookmarkable
URL's
      Backbone.history.start();
  </script>
  <body>

     <a href="#route1">Route1 </a>
     <a href="#route2">Route2 </a>
     <a href="#route3">Route3 </a>
  </body>
</html>
```

Output

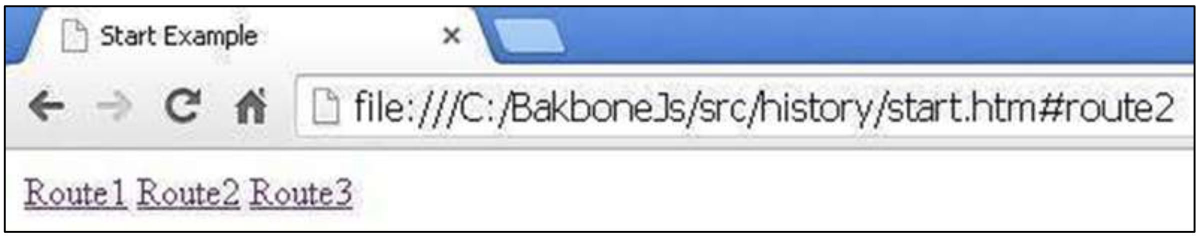
Let us carry out the following steps to see how the above code works:

- Save the above code in the **start.htm** file.
- Open this HTML file in a browser.



Route1 Route2 Route3

NOTE: The above functionality is related to the address bar. So, when you open the above code in the browser, it will show the result as follows.



Click here for the demo – (Clicking on this link takes you to – https://www.tutorialspoint.com/backbonejs/src/history/start.htm)

9. BackboneJS — Sync

It is used to persist the state of the model to the server. The following table lists down the methods which can be used to manipulate the **BackboneJS-Sync**:

S.No.	Methods & Description
1	**Backbone.sync** It persists the state of the model to the server.
2	**Backbone.ajax** It defines the custom ajax function.
3	**Backbone.emulateHTTP** If your web server does not support REST or HTTP approach, then turn on the Backbone.emulateHTTP.
4	**Backbone.emulateJSON** It is used to handle the requests encoded with **application/json** by setting the method to **true**.

Backbone.sync

Description

It is a function that Backbone calls every time to read or save the model to the server. It represents the state of the model.

Syntax

```
sync.(method, model, options)
```

Parameters

- **method:** It represents the CRUD operations such as create, read, update and delete.
- **model:** It includes the model to be saved.
- **options:** It fires success or error messages depending on the method succeeded.

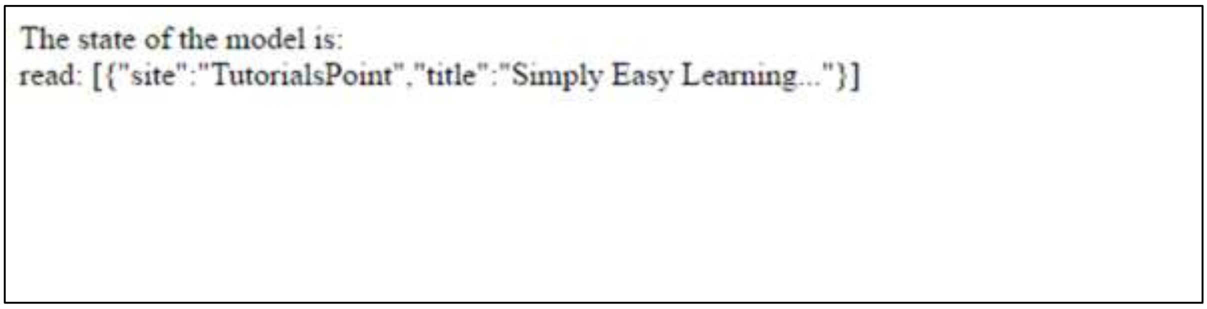
Example

```
<!DOCTYPE html>
   <head>
      <title>Sync Example</title>
         <script src="https://code.jquery.com/jquery-2.1.3.min.js" 
type="text/javascript"></script>
         <script 
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>
         <script 
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js" 
type="text/javascript"></script>
   </head>
   <body>
      <script type="text/javascript">
         //The sync() method reads and fetched the model data
         Backbone.sync = function(method, model) {
            document.write("The state of the model is:");
            document.write("<br>");

            //The 'method' specifies state of the model
            document.write(method + ": " + JSON.stringify(model));
         };

         //'myval' is a collection instance and contains the values which are 
to be fetched in the collection
         var myval = new Backbone.Collection({
            site:"TutorialsPoint",
            title:"Simply Easy Learning..."
         });

         //The myval.fetch() method displays the model's state by delegating 
to sync() method
         myval.fetch();
      </script>
   </body>
</html>
```

Output

Let us carry out the following steps to see how the above code works:

- Save the above code in the **backbone-sync.htm** file.
- Open this HTML file in a browser.



```
The state of the model is:
read: [{"site":"TutorialsPoint","title":"Simply Easy Learning..."}]
```

BackboneJS-Sync Backbone.emulateHTTP

Description

If you are using a legacy web server that doesn't support Backbone's default REST/HTTP approach, you may choose to turn on the **Backbone.emulateHTTP**. Setting this option to **true** will fake PUT, PATCH and DELETE requests with a HTTP POST, setting the X-HTTP-Method-Override header with the true method.

If **emulateJSON** is also on, the true method will be passed as an **additional _method** parameter.

Syntax

```
Backbone.emulateHTTP = true
```

Example

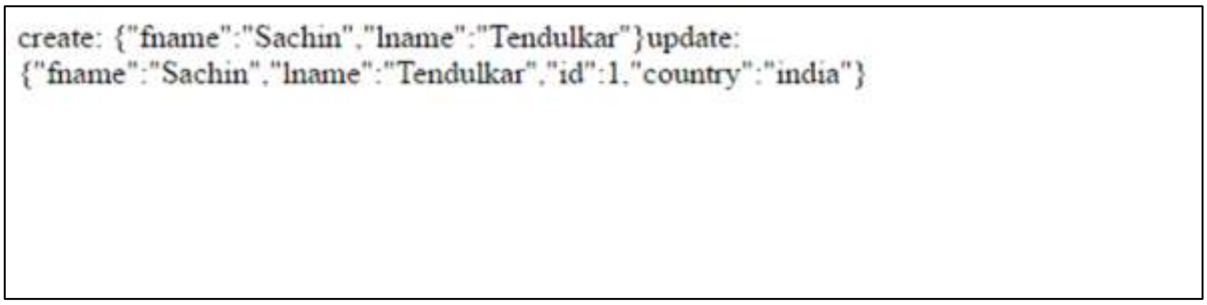
```
<!DOCTYPE html>
   <head>
      <title>Sync Example</title>
         <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js"
type="text/javascript"></script>
   </head>
   <body>
      <script type="text/javascript">
```

```
         //If web server that doesn't support Backbone's REST/HTTP approach,
then turn on 'Backbone.emulateHTTP'
         Backbone.emulateHTTP = true;


         //If web server can't handle requests encoded as application/json,
then set the 'Backbone.emulateJSON' to true
         Backbone.emulateJSON = true;


         //The sync() method reads and fetch the model data
         Backbone.sync = function(method, model) {
            document.write(method + ": " + JSON.stringify(model));
            model.set('id', 1);   //Set the model with id as '1'
         };


         //'Player' is a model name and contains the values to be displayed
when you save the model
         var Player = new Backbone.Model({
            fname:"Sachin",
            lname:"Tendulkar"
         });


         //The 'save()' method saves data of the model by delegating to sync()
method
         Player.save();


         //Update the model with a value
         Player.save({country: "india"});
      </script>
   </body>
</html>
```

Output

Let us carry out the following steps to see how the above code works:

- Save the above code in the **backbone-emulatehttp.htm** file.
- Open this HTML file in a browser.



create: {"fname":"Sachin","lname":"Tendulkar"}update: {"fname":"Sachin","lname":"Tendulkar","id":1,"country":"india"}

BackboneJS-Sync Backbone.emulateJSON

Description

It is used to handle the requests encoded with **application/json** by setting the method to **true**.

Syntax

```
Backbone.emulateJSON=true
```

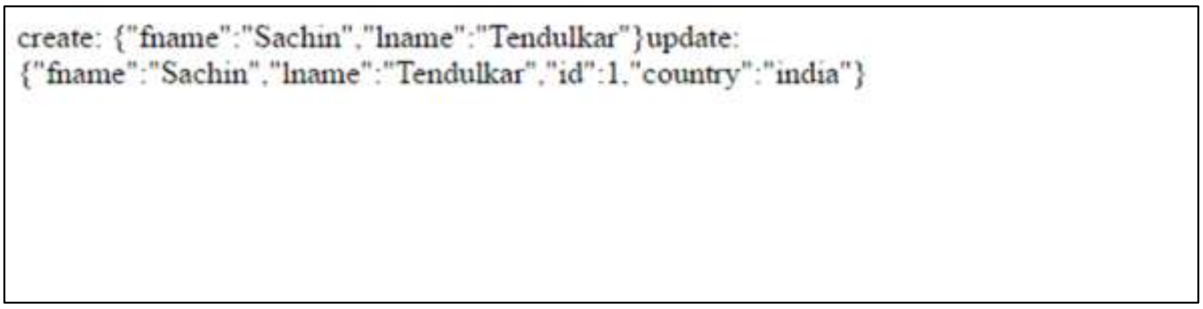
Example

```
<!DOCTYPE html>
   <head>
      <title>Sync Example</title>
         <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js"
type="text/javascript"></script>
   </head>
   <body>
      <script type="text/javascript">
         //If web server that doesn't support Backbone's REST/HTTP approach,
then turn on 'Backbone.emulateHTTP'
         Backbone.emulateHTTP = true;


         //If web server can't handle requests encoded as application/json,
then set the 'Backbone.emulateJSON' to true
         Backbone.emulateJSON = true;


         //The sync() method reads and fetch the model data
         Backbone.sync = function(method, model) {
            document.write(method + ": " + JSON.stringify(model));
            model.set('id', 1);   //Set the model with id as '1'
         };

         //'Player' is a model name and contains the values to be displayed
when you save the model
         var Player = new Backbone.Model({
            fname:"Sachin",
            lname:"Tendulkar"
         });

         //The 'save()' method saves data of the model by delegating to sync()
method
         Player.save();

         //When you save the model, it updates the model along with this value
         Player.save({country: "india"});
      </script>
   </body>
</html>
```

Output

Let us carry out the following steps to see how the above code works:

- Save the above code in the **backbone-emulatejson.htm** file.
- Open this HTML file in a browser.



```
create: {"fname":"Sachin","lname":"Tendulkar"}update:
{"fname":"Sachin","lname":"Tendulkar","id":1,"country":"india"}
```

10. BackboneJS – View

Views are used to reflect "how your data model looks like". They represent the model's data to the user. They provide the idea behind the presentation of the model's data to the user. It handles the user input events, binds events and methods, renders model or collection and interacts with the user.

The following table lists down the methods which can be used to manipulate the **BackboneJS-Views**.

S.No.	Methods & Description
1	**extend** It extends the **Backbone.View** class to create a custom view class.
2	**initialize** It instantiates the view by using the new keyword.
3	**el** It defines which element to be used as the view reference.
4	**$el** It represents the jQuery object for the view's element.
5	**setElement** It specifies the existing DOM element to a different DOM element.
6	**attributes** They can be used as DOM element attributes on the view class.
7	$(**jQuery**) It is used as a selector that contains the $ function and runs queries within the view's element.
8	**template** While rendering the view, template creates reusable copies of markup and provides access to instance data.

9	**render** It contains the logic for rendering a template.
10	**remove** Removes a view from the DOM.
11	**delegateEvents** Binds elements to the specified DOM elements with callback methods to handle events.
12	**undelegateEvents** It removes delegate events from the view.

BackboneJS – View Extend

Description

It extends the **Backbone.View** class to create a custom view class.

Syntax

```
Backbone.View.extend(properties, classProperties)
```

Parameters

- **properties:** It provides instance properties for the view class.
- **classProperties:** The classProperties attached to the view's constructor function.

Example

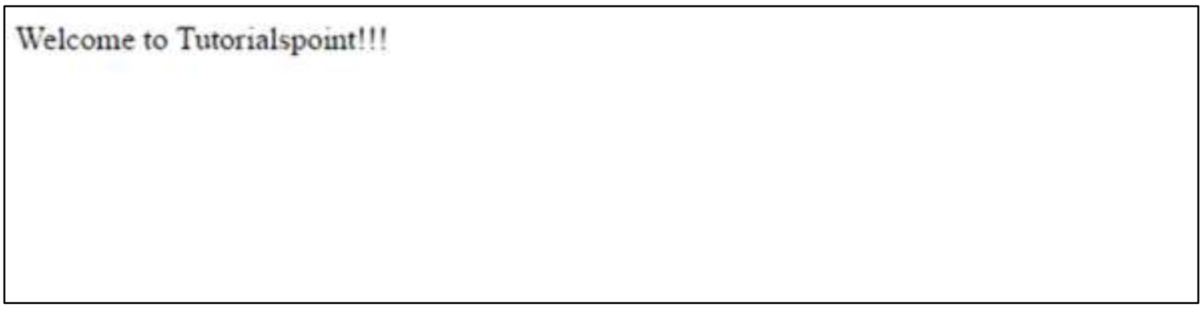
```
<!DOCTYPE html>
   <head>
      <title>View Example</title>
         <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js"
type="text/javascript"></script>
   </head>
   <body>
```

```
      <script type="text/javascript">
         //'ViewDemo' is a name of the view class
         var ViewDemo = Backbone.View.extend({

         //This function is called when the view is instantiated
            initialize:function(){
               document.write('Welcome to Tutorialspoint!!!');
             }
         });

         //'myview' is an instance of the view 'ViewDemo'
         var myview = new ViewDemo();
      </script>
   </body>
</html>
```

Output

Let us carry out the following steps to see how the above code works:

- Save the above code in the **extend.htm** file.
- Open this HTML file in a browser.



```
Welcome to Tutorialspoint!!!
```

BackboneJS – View Initialize

Description

It instantiates the view by using the **new** keyword and gets called when the view is first created.

Syntax

```
new View(options)
```

Parameters

- **options:** These are optional parameters which can be model, collection, el, id, className, tagName, attributes and events.

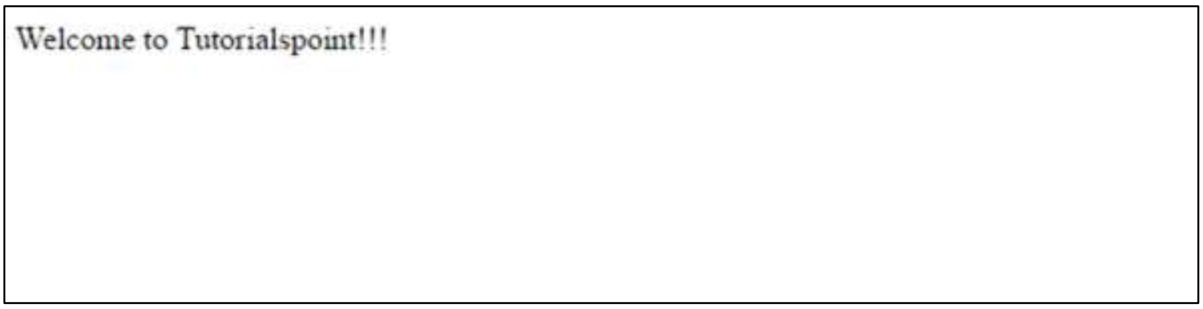
Example

```
<!DOCTYPE html>
   <head>
      <title>View Example</title>
         <script src="https://code.jquery.com/jquery-2.1.3.min.js" 
type="text/javascript"></script>
         <script 
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>
         <script 
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js" 
type="text/javascript"></script>
   </head>
   <body>
      <script type="text/javascript">
         //'ViewDemo' is a name of the view class
         var ViewDemo = Backbone.View.extend({

         //This function is called when the view is instantiated
            initialize:function(){
               document.write('Welcome to Tutorialspoint!!!');
             }
         });

         //'myview' is an instance of the view 'ViewDemo'
         var myview = new ViewDemo();
      </script>
   </body>
</html>
```

Output

Let us carry out the following steps to see how the above code works:

- Save the above code in the **initialize.htm** file.
- Open this HTML file in a browser.


Welcome to Tutorialspoint!!!

BackboneJS – View El

Description

It defines which element to be used as the view reference. The **this.el** is created from the view's **tagName**, **className**, **id** and **attributes** properties, if specified. If not, **el** is an empty **div**.

Syntax

```
view.el
```

Example

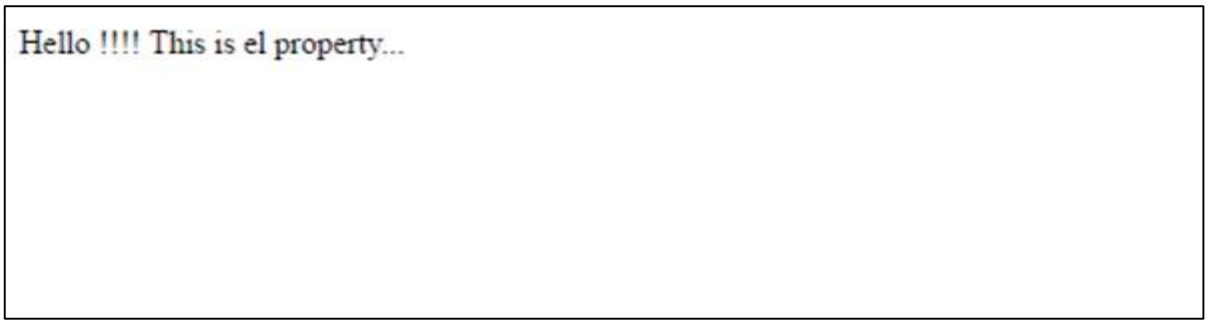
```
<!DOCTYPE html>
   <head>
      <title>View Example</title>
         <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js"
type="text/javascript"></script>
   </head>
      <script type="text/javascript">
         //'ViewDemo' is a name of the view class
         ViewDemo = Backbone.View.extend({

         //This function gets called when the view is instantiated
            initialize: function(){
               document.write("Hello !!!! This is el property...");
             }
         });
```

```
          //'viewdemo' is a instance of the 'ViewDemo' class
          var viewdemo = new ViewDemo({ el: $("myapp") });
       </script>
    <body>
        <div id="myapp"></div>
    </body>
</html>
```

Output

Let us carry out the following steps to see how the above code works:

- Save the above code in the **el.htm** file.
- Open this HTML file in a browser.



```
Hello !!!! This is el property...
```

BackboneJS – View $el

Description

It represents a **cached jQuery object** for the view's element. A handy reference instead of re-wrapping the DOM element all the time.

Syntax

```
view.$el
```

Example

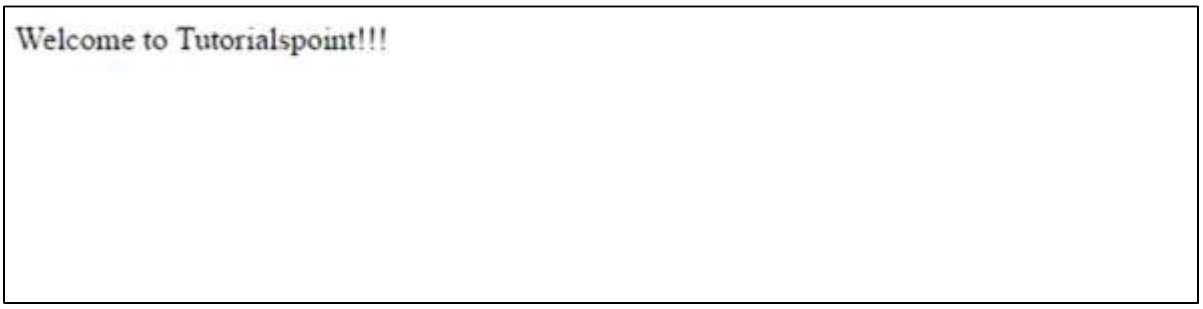
```
<!DOCTYPE html>
   <head>
      <title>View Example</title>
         <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>
```

```
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js"
type="text/javascript"></script>
   </head>
   <body>
      <div id="myapp"></div>
      <script type="text/javascript">
         //'ViewDemo' is a name of the view class
         var ViewDemo = Backbone.View.extend({

         //'el' is a view reference in which every element of view associate
with HTML content will be rendered
            el: '/backbonejs/backbonejs_view.htmmyapp',

            //This function is called when the view is instantiated
               initialize: function(){
               this.render();  //render method specifies how to handle and
what to display in the view
            },

            //'$el' is cached object which pushes the content defined within it
            render: function(){
               this.$el.html("Welcome to Tutorialspoint!!!");
            }
         });

         //'myview' is a instance of the 'ViewDemo' class
         var myview = new ViewDemo();
      </script>
   </body>
</html>
```

Output

Let us carry out the following steps to see how the above code works:

- Save the above code in the **dollar-el.htm** file.
- Open this HTML file in a browser.



Welcome to Tutorialspoint!!!

BackboneJS – View setElement

Description

If you'd like to apply a Backbone view to a different DOM element, use **setElement**, which will also create the cached **$el** reference and move the view's delegated events from the old element to the new one.

Syntax

```
view.setElement(element)
```

Parameters

- **element:** It is the element which can be changed from the existing element to a different element.

Example

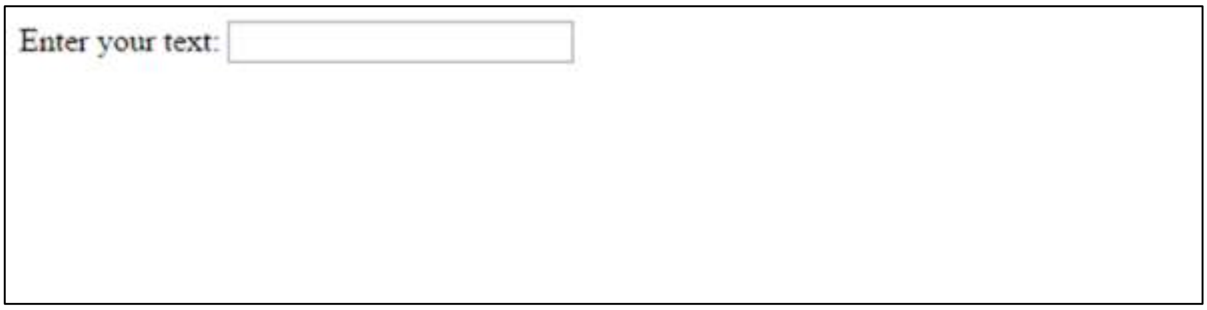
```
<!DOCTYPE html>
   <head>
      <title>View Example</title>
         <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js"
type="text/javascript"></script>
   </head>
   <body>
      <div id="myview">
          Enter your text: <input type="text"/>
      </div>
      <div id="myapp"></div>
      <script type="text/javascript">
```

```
         //'ViewDemo' is a name of the view class
         var ViewDemo = Backbone.View.extend({

            //Event triggers 'sayHi' function when you enter the text in input tag
            events: {
               'change input': 'sayHi'
            },

            //This function is called when the view is instantiated
            initialize: function() {
                this.setElement($('#myview'));   //'setElement' changes the
element associated with the view
            },

            //when you enter the text, it displays the below line on the screen
            sayHi: function() {
               document.write('Welcome to Tutorialspoint!!!');
            }
         });

          //'viewdemo' is a instance of the 'ViewDemo' class
         var viewdemo = new ViewDemo;
      </script>
   </body>
</html>
```

Output

Let us carry out the following steps to see how the above code works:

- Save the above code in the **setelement.htm** file.
- Open this HTML file in a browser.


Enter your text:

BackboneJS – View Attributes

Description

These are the hash of attributes that will be set as the **HTML DOM** element attributes on the view's **el** (id, class, data-properties, etc.), or a function that returns such a hash.

Syntax

```
view.attributes
```

Example

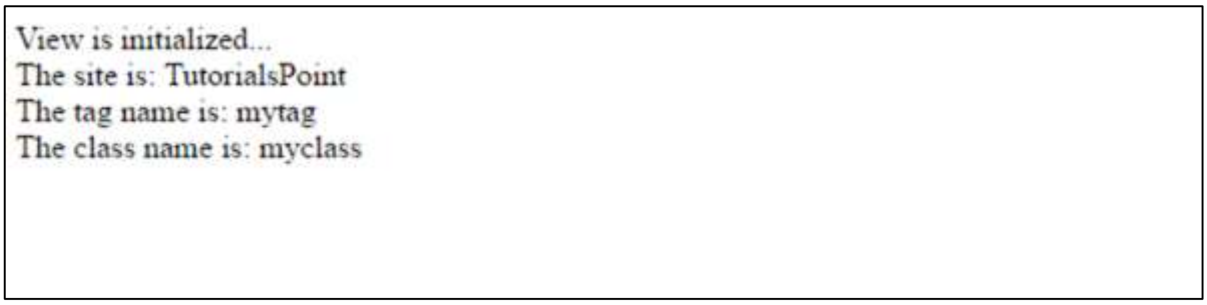
```
<!DOCTYPE html>
   <head>
      <title>View Example</title>
         <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js"
type="text/javascript"></script>
   </head>
   <body>
      <div id="myapp"></div>
      <script type="text/javascript">
         //'MyModel' is a model name and includes default value
         var MyModel = Backbone.Model.extend({
            defaults: {
               name: "sachin tendulkar"
            }
         });
```

```
          //'ViewDemo' is a name of the view class
          var ViewDemo = Backbone.View.extend({

          //This function is called when the view is instantiated
             initialize: function () {
                document.write("View is initialized..."+"<br>");

                //gets site name by using model instance
                document.write("The site is: ",this.model.get("Site")+"<br>");
                document.write("The tag name is: ",this.tagName+"<br>");
                document.write("The class name is: ",this.className);
             },
          });
          $(function () {

             //'mymodel' is an instance of the model
             var mymodel = new MyModel({ Site: "TutorialsPoint" });

             //'myview' is an instance of the view class with attributes model,
tagName, className
             var myview = new ViewDemo({ model: mymodel, tagName: "mytag",
className: "myclass"});
          })
       </script>
   </body>
</html>
```

Output

Let us carry out the following steps to see how the above code works:

- Save the above code in the **attributes.htm** file.
- Open this HTML file in a browser.



```
View is initialized...
The site is: TutorialsPoint
The tag name is: mytag
The class name is: myclass
```

BackboneJS – View $(jQuery)

Description

If **jQuery** is included on the page, each view has a **$** function that runs queries scoped within the view's element. If you use this scoped jQuery function, you don't have to use the model ids as part of your query to pull out specific elements in a list, and can rely much more on the HTML class attributes. It is equivalent to running – ***view.$el.find(selector)***.

Syntax

```
view.$(selector)
```

Parameters

- **selector:** It uses the different types of selectors such as id or class.

Example

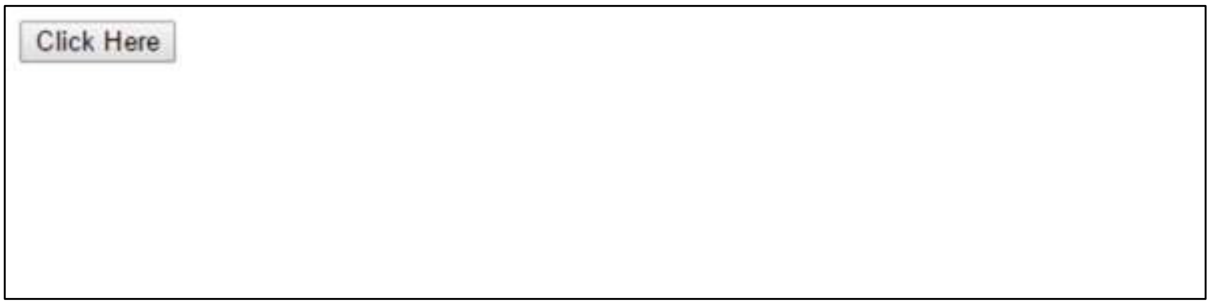
```
<!DOCTYPE html>
   <head>
      <title>View Example</title>
         <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js"
type="text/javascript"></script>
   </head>
   <body>
   <div id="myVal">
        <button id="button" data-test="">Click Here</button>
    </div>
   <span id="myLog"></span>
      <script type="text/javascript">
```

```
         //The variable contains id selector as 'mydata'
          var myLog = $('#mydata');

          //The variable 'data' is used to display the values
          var data = function(val) {
             document.write(val);
          };

          //'ViewDemo' is a name of the view class
          var ViewDemo = Backbone.View.extend({

          //When click event occurs it activates the defined functions
'myFunc1' and 'myFunc2'
             events: {
                'click [data-test]' : 'myFunc1',
                'click *[data-test]': 'myFunc2',
             },
             //'el' uses '#myVal' as the view reference
             el: $('#myVal'),

             //When user clicks the button, it refers to the button defined
within the 'div' tag and
             //it will display the below statements
             myFunc1: function () {
                data('Hello...');
             },
             myFunc2: function () {
                data('Welcome to Tutorialspoint...');
             }
         });

         //'myview' is an instance of the 'ViewDemo' class
         var myview = new ViewDemo();
       </script>
    </body>
</html>
```

Output

Let us carry out the following steps to see how the above code works:

- Save the above code in the **dollar-jquery.htm** file.
- Open this HTML file in a browser.



BackboneJS – View Template

Description

While rendering the view, the template creates reusable copies of markup and provides access to instance data.

Syntax

```
view.template(data)
```

Parameters

- *data:* Data to be accessed when rendering the view.

Example

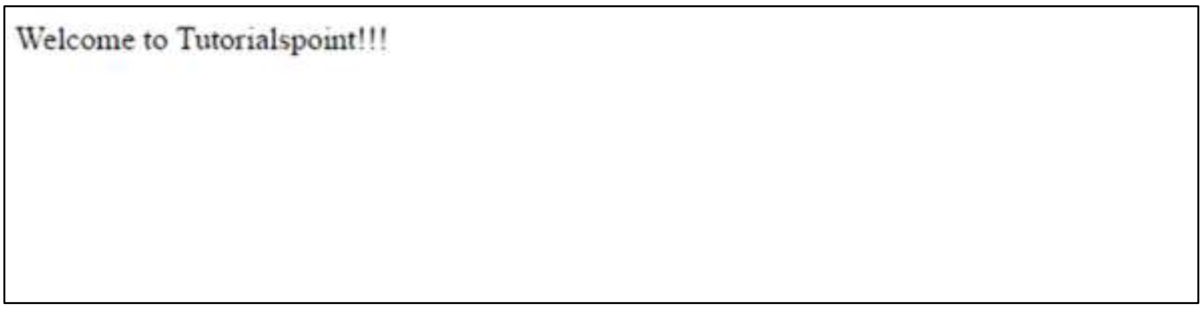
```
<!DOCTYPE html>
   <head>
      <title>View Example</title>
         <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js"
type="text/javascript"></script>
   </head>
   <body>
      <div id="mydiv"></div>
      <script type="text/javascript">
```

```
         //'ViewDemo' is a name of the view class
         var ViewDemo = Backbone.View.extend({

            //'el' uses '#mydiv' as the view reference
            el: $('#mydiv'),

            //'template' provides access to instance data when rendering the view
            template: _.template("Welcome to <%= name %>"),

               //This function is called when the view is instantiated
               initialize: function(){
                  this.render();
               },

               //'render' provides the logic required to construct the view
               render: function(){

                  //'$el' is cached object that push the content defined within
it and
                  //display the value of 'name' when 'template' access the data
                  this.$el.html(this.template({name: 'Tutorialspoint...!'}));
               }
         });

         //'myview' is an instance of the 'ViewDemo' class
         var myview = new ViewDemo();
      </script>
   </body>
</html>
```

Output

Let us carry out the following steps to see how the above code works:

- Save the above code in the **template.htm** file.
- Open this HTML file in a browser.


Welcome to Tutorialspoint!!!

BackboneJS – View Render

Description

It contains the logic for rendering the template which constructs the view.

Syntax

```
view.render()
```

Example

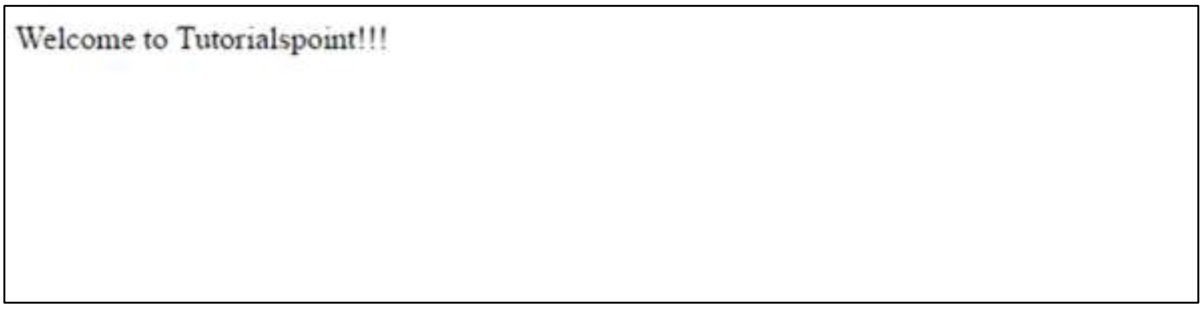
```
<!DOCTYPE html>
   <head>
      <title>View Example</title>
         <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js"
type="text/javascript"></script>
   </head>
   <body>
      <div id="mydiv"></div>
      <script type="text/javascript">
         //'ViewDemo' is a name of the view class
         var ViewDemo = Backbone.View.extend({

            //'el' uses '#mydiv' as the view reference
            el: $('#mydiv'),

            //'template' provides access to instance data when rendering the
view
```

```
            template: _.template("Welcome to <%= name %>"),

               //This function is called when the view is instantiated
               initialize: function(){
                  this.render();
               },

               //'render' provides the logic required to construct the view
               render: function(){

                  //'$el' is cached object that push the content defined
within it and
                  //display the value of 'name' when 'template' access the
data
                  this.$el.html(this.template({name: 'Tutorialspoint...!'}));
               }
         });

         //'myview' is an instance of the 'ViewDemo' class
         var myview = new ViewDemo();
      </script>
   </body>
</html>
```

Output

Let us carry out the following steps to see how the above code works:

- Save the above code in the **render.htm** file.
- Open this HTML file in a browser.



```
Welcome to Tutorialspoint!!!
```

BackboneJS – View Remove

Description

It is used to remove the view from the DOM and calls **stopListening** to remove any bound events that the view has **listenTo'd**.

Syntax

```
view.remove()
```

Example

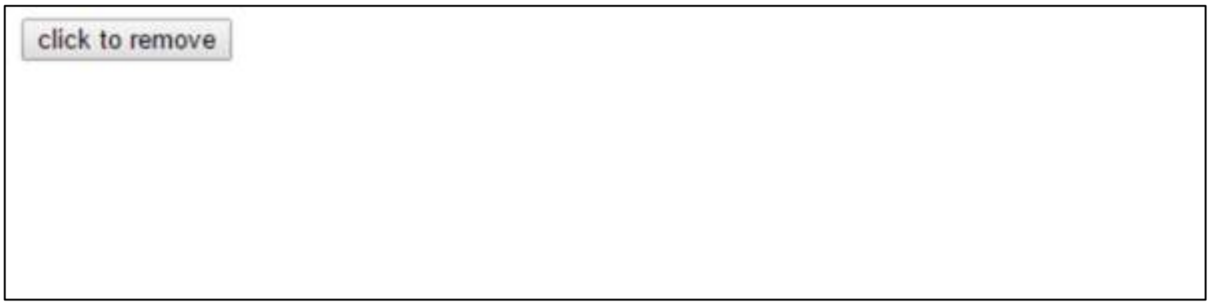
```
<!DOCTYPE html>
   <head>
      <title>View Example</title>
         <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js"
type="text/javascript"></script>
   </head>
   <body>
      <div id="mydiv"></div>
      <script type="text/javascript">
         //'ViewDemo' is a name of the view class
         var ViewDemo = Backbone.View.extend({

            //When click event occurs it activates the defined function
'removeFunc'
            events: {'click button': 'removeFunc'
               },
               removeFunc: function () {

                  //the 'remove()' method removes the view from the DOM
                  this.remove();

                  //After removing the view, it shows length as '0'
                 document.write("After removing, view becomes:
",$('#mydiv').length);
```

```
                },

                //'render' provides the logic required to construct the view
                render: function () {

                   //'$el' is cached object that push the content defined
within it and
                   //display the button which should clicked by the user to
remove the view
                   this.$el.html('<button>click to remove</button>');
                },

                //This function is called when the view is instantiated
                initialize:function(){this.render();}
          });

          //'myview' is an instance of the 'ViewDemo' class
          var myview = new ViewDemo({el: '#mydiv'});
       </script>
    </body>
</html>
```

Output

Let us carry out the following steps to see how the above code works:

- Save the above code in the **remove.htm** file.
- Open this HTML file in a browser.



BackboneJS – View delegateEvents

Description

Binds elements to the specified DOM elements with callback methods to handle events. If events are removed from the view, this method can be used to attach the events to the view.

Syntax

```
delegateEvents(events)
```

Parameters

- **events:** It provides events that are needed for reattaching to the view.

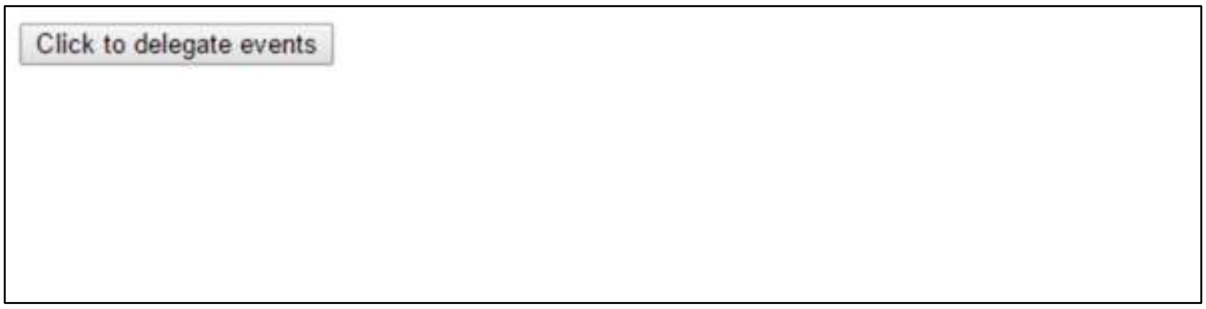
Example

```
<!DOCTYPE html>
   <head>
      <title>View Example</title>
         <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js"
type="text/javascript"></script>
   </head>
   <body>
      <div id="mydiv"></div>
      <script type="text/javascript">
         //'ViewDemo' is a name of the view class
         var ViewDemo = Backbone.View.extend({

            //When click event occurs it activates the defined function
'delegateFunc'
            events: {'click button': 'delegateFunc'},
               delegateFunc: function () {
                  this.remove();   //The 'remove()' method removes the view
from the DOM
                  document.write("Welcome to Tutorialspoint...");
               },
               //'render' provides the logic required to construct the view
               render: function () {

                  //'$el' is cached object that push the content defined
within it and

                  //display the value when user clicks the button
                  this.$el.html('<button>Click to delegate events</button>');
               },

               //This function is called when the view is instantiated
               initialize:function(){this.render();}
         });

         //'myview' is an instance of the 'ViewDemo' class
         var myview = new ViewDemo({el: '#mydiv'});   //'el' defines which
element to be used as the view reference

         //Here defining the events which are reattaching to the view using
'delegateEvents()' method
         myview.delegateEvents();
      </script>
   </body>
</html>
```

Output

Let us carry out the following steps to see how the above code works:

- Save the above code in the **delegateevents.htm** file.
- Open this HTML file in a browser.



BackboneJS – View undelegateEvents

Description

It is used to remove the delegated events of the view from the DOM.

Syntax

```
delegateEvents()
```

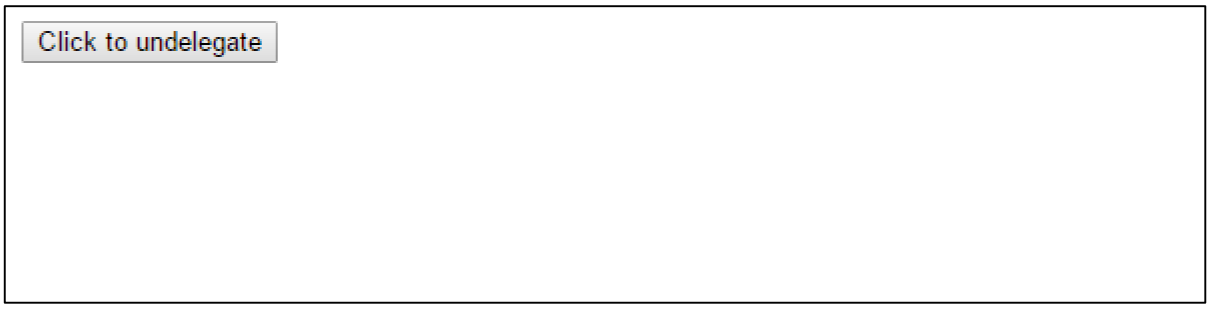
Example

```
<!DOCTYPE html>
   <head>
      <title>View Example</title>
         <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js"
type="text/javascript"></script>
   </head>
   <body>
      <div id="mydiv"></div>
      <script type="text/javascript">
         //'ViewDemo' is a name of the view class
         var ViewDemo = Backbone.View.extend({

            //When click event occurs it activates the defined function
'undelegateFunc'
            events: {'click button': 'undelegateFunc'},
               undelegateFunc: function () {
                  document.write("do something...");
                        $(this.el).undelegate('button', 'click');
               },

               //'render' provides the logic required to construct the view
               render: function () {
                  //'$el' is cached object that push the content defined within
it and
                  //display the value when user clicks the button
                  this.$el.html('<button>Click to undelegate</button>');
               },

               //This function is called when the view is instantiated
               initialize:function(){this.render();}

         });

         //'myview' is an instance of the 'ViewDemo' class
         var myview = new ViewDemo({el: '#mydiv'});
      </script>
   </body>
</html>
```

Output

Let us carry out the following steps to see how the above code works:

- Save the above code in the **undelegateevents.htm** file.
- Open this HTML file in a browser.



11. BackboneJS – Utility

The utility class defines a set of methods used for implementing the backbone utility. The following table lists down the methods which you can use to manipulate the **BackboneJS-Utility**:

S.No.	Methods & Description
1	**Backbone.noConflict** It displays the original value of Backbone object and allows to store reference to a backbone.
2	**Backbone.$** It allows Backbone to use particular object as DOM library.

BackboneJS-Utility Backbone.noConflict

Description

It displays the original value of the Backbone object and allows to store reference to a backbone.

Syntax

```
var backbone=Backbone.noConflict();
```

Example

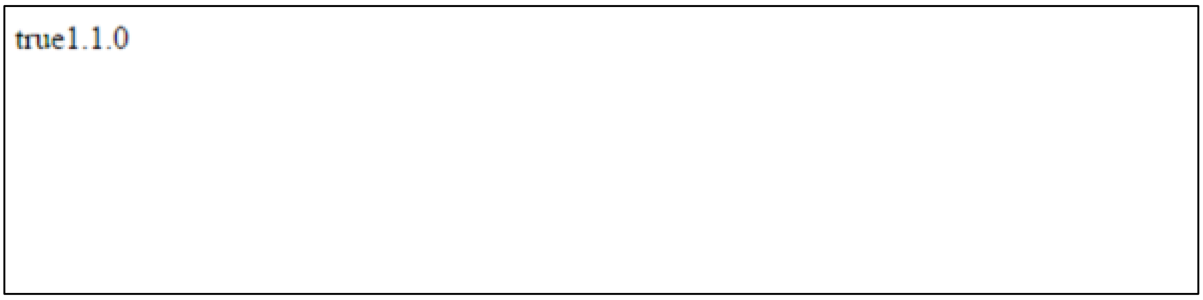
```
<!DOCTYPE html>
   <head>
      <title>Utility Example</title>
         <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js"
type="text/javascript"></script>
   </head>
   <body>
```

```
      <script>
          //It is reference to a last parsed verison of Backbone.js
          this.Backbone = {
             "Example for Backbone utility": true
          };
       </script>
       <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.0/backbone-
min.js"></script>
       <script type="text/javascript">
          //'Backbone.noConflict()' gives reference to a Backbone before last
parsed version of Backbone was loaded
          var backboneParsed = Backbone.noConflict();


          //It looks for the previous version and logs 'true' on the browser
          document.write(Backbone["Example for Backbone utility"]);


          //It displays the version of the Backbone.js contained in the new
'backboneParsed' namespace
          document.write(backboneParsed.VERSION);
       </script>
    </body>
</html>
```

Output

Let us carry out the following steps to see how the above code works:

- Save the above code in the **backbone-noconflict.htm** file.
- Open this HTML file in a browser.



```
true1.1.0
```

BackboneJS-Utility Backbone.$

Description

It allows the Backbone to use an object as the DOM library.

Syntax

```
Backbone.$=$;
```

Example

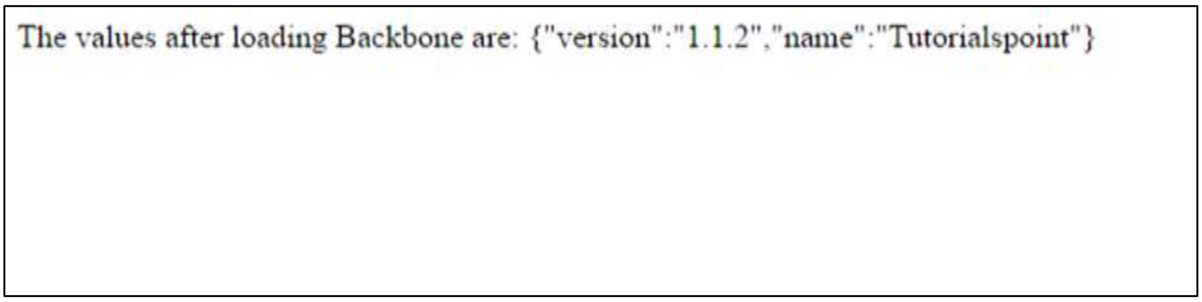
```
<!DOCTYPE html>
   <head>
      <title>Utility Example</title>
         <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-
min.js" type="text/javascript"></script>
   </head>
   <body>
      <script>
         //The value specified for the variable '$' in the global scope
         var $ = {version:'0.1.1',name:'Tutorialspoint'};
      </script>
      <script
src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-min.js"
type="text/javascript"></script>
      <script type="text/javascript">
         //Set the value for the variable after Backbone is loaded
         Backbone.$ = {version:'1.1.2',name:'Tutorialspoint'};


         //It displays the defined values in the JSON format
         document.write("The values after loading Backbone are:
",JSON.stringify(Backbone.$));
      </script>
   </body>
</html>
```

Output

Let us carry out the following steps to see how the above code works:

- Save the above code in the **backbone-dollar.htm** file.

- Open this HTML file in a browser.



```
The values after loading Backbone are: {"version":"1.1.2","name":"Tutorialspoint"}
```