

Knockout.JS

MVVM library

Introduction

Js:

```
var myViewModel = {  
    personName: ko.observable('Bob'),  
    personAge: ko.observable(123)  
};  
ko.applyBindings(myViewModel);
```

Html:

The name is

Generated html:

The name is Bob

Bindings

```
<div data-bind="visible: shouldShowMessage">
<span data-bind="text: myMessage"></span>
<div data-bind="html: details"></div>
<div data-bind="css: { profitWarning: currentProfit() < 0 }">
<div data-bind="style: { color: currentProfit() < 0 ? 'red' : 'black' }">
<a data-bind="attr: { href: url, title: details }">

<input type='checkbox' data-bind="checked: hasCellphone, enable:
hasCellphoneEnabled" />
<input type='text' data-bind="value: cellphone, valueUpdate: 'keyup' " />

<input data-bind="hasFocus: isSelected" />
<select data-bind="options: availableCountries"></select>
```

Bindings

```
<table>
  <tbody data-bind="foreach: people">
    <tr>
      <td data-bind="text: firstName"></td>
      <td data-bind="text: lastName"></td>
    </tr>
  </tbody>
</table>
```

```
ko.applyBindings({
  people: [
    { firstName: 'Bert', lastName: 'Bertington' },
    { firstName: 'Charles', lastName: 'Charlesforth' },
    { firstName: 'Denise', lastName: 'Dentiste' }
  ]
});
```

Bindings

```
<ul data-bind="foreach: planets">
  <li>
    Planet: <b data-bind="text: name"> </b>
    <div data-bind="if: capital">
      Capital: <b data-bind="text: capital.cityName"> </b>
    </div>
  </li>
</ul>
```

```
<script>
  ko.applyBindings({
    planets: [
      { name: 'Mercury', capital: null },
      { name: 'Earth', capital: { cityName: 'Barnsley' } }
    ]
  });
</script>
```

Aliases

```
ko.field = ko.observable;  
ko.array = ko.observableArray;  
  
ko.property = function (getter) {  
  var result;  
  if (getter.read) {  
    result = ko.computed({  
      deferEvaluation: true,  
      read: getter.read,  
      write: getter.write  
    });  
  } else {  
    result = ko.computed({  
      deferEvaluation: true,  
      read: getter  
    });  
  }  
  return result;  
};
```

Aliases

```
function AwardsViewModel() {  
  this.has10KPingsBadge = ko.field(null);  
  this.hasReachedLevel10 = ko.field(null);  
  this.awards = ko.array([]);  
  this.tooltipVisible = ko.field(false);  
  
  this.showTooltip = function () {  
    self.tooltipVisible(true);  
  };  
  
  this.hasAnyBadge = ko.property({  
    read: function() {  
      return self.hasReachedLevel10 () && self.hasReachedLevel10 ();  
    },  
    write: function(val) {  
      self.has10KPingsBadge (val);  
      self.hasReachedLevel10(val);  
    }  
  });  
}
```

Subscriptions

```
myViewModel.personName.subscribe(function(newValue) {  
    alert("The person's new name is " + newValue);  
});
```

```
myViewModel.personName.subscribe(function(oldValue) {  
    alert("The person's previous name is " + oldValue);  
}, null, "beforeChange");
```

```
var subscription = myViewModel.personName.subscribe(function(newValue) { /* do stuff */ });  
// ...then later...  
subscription.dispose(); // I no longer want notifications
```


Computed

```
function AppViewModel() {  
  var self = this;  
  
  self.firstName = ko.observable('Bob');  
  self.lastName = ko.observable('Smith');  
  
  self.fullName = ko.computed(function() {  
    return self.firstName() + " " + self.lastName();  
  });  
}
```

Writable Computed

First name: Planet

Last name: Earth

Hello, Planet Earth

```
<div>First name: <span data-bind="text: firstName"></span></div>
<div>Last name: <span data-bind="text: lastName"></span></div>
<div class="heading">Hello, <input data-bind="textInput: fullName"/></div>
```

```
function MyViewModel() {
  this.firstName = ko.observable('Planet');
  this.lastName = ko.observable('Earth');

  this.fullName = ko.computed({
    read: function () {
      return this.firstName() + " " + this.lastName();
    },
    write: function (value) {
      var lastSpacePos = value.lastIndexOf(" ");
      if (lastSpacePos > 0) { // Ignore values with no space character
        this.firstName(value.substring(0, lastSpacePos)); // Update "firstName"
        this.lastName(value.substring(lastSpacePos + 1)); // Update "lastName"
      }
    },
    owner: this
  });
}
```

Writable Computed

Enter bid price: \$25.99

(Raw value: 25.99)

```
<div>Enter bid price: <input data-bind="textInput: formattedPrice"/></div>  
<div>(Raw value: <span data-bind="text: price"></span>)</div>
```

```
function MyViewModel() {  
    this.price = ko.observable(25.99);  
  
    this.formattedPrice = ko.computed({  
        read: function () {  
            return '$' + this.price().toFixed(2);  
        },  
        write: function (value) {  
            value = parseFloat(value.replace(/^[^\d]/g, ""));  
            this.price(isNaN(value) ? 0 : value); // Write to underlying storage  
        }  
    });  
}
```

Computed

```
ko.computed(function() {  
  var params = {  
    page: this.pageIndex(),  
    selected: this.selectedItem.peek()  
  };  
  $.getJSON('/Some/Json/Service', params);  
}, this);
```

```
var myComputed = ko.computed(function() {  
  var isFirstEvaluation = ko.computedContext.isInitial(),  
      dependencyCount = ko.computedContext.getDependenciesCount(),  
      console.log("Evaluating " + (isFirstEvaluation ? "for the first time" : "again"));  
  console.log("By now, this computed has " + dependencyCount + " dependencies");  
});
```

```
someObservableOrComputed.extend({ rateLimit: { timeout: 500, method:  
"notifyWhenChangesStop" } });
```

```
myViewModel.fullName = ko.computed(function() {  
  return myViewModel.firstName() + " " + myViewModel.lastName();  
}).extend({ notify: 'always', rateLimit: 500 });
```

Computed

```
var upperCaseName = ko.computed(function() {  
    return name().toUpperCase();  
}).extend({ throttle: 500 }); // deprecated
```

```
ko.observableArray.fn.pushAll = function(valuesToPush) {  
    var underlyingArray = this();  
    this.valueWillMutate();  
    ko.utils.arrayPushAll(underlyingArray, valuesToPush);  
    this.valueHasMutated();  
    return this;  
};
```

```
this.users.pushAll(dataFromServer);
```

Validation

ko.validation

```
var myObj = ko.observable().extend({ required: true });  
var myObj = ko.observable().extend({ required: { onlyIf: function() { return true; } } });  
var myObj = ko.observable().extend({ min: 2 });  
var myObj = ko.observable().extend({ maxLength: 12 });  
var myObj = ko.observable().extend({ pattern: '^[a-z0-9].$' });
```

```
ko.validation.rules['exampleRule'] = {  
  validator: function(val, otherVal){  
    /* awesome logic */  
  },  
  message: 'Sorry Chief, {0} this is not Valid'  
};
```

<https://github.com/ericmbarnard/Knockout-Validation>

Custom bindings

```
ko.bindingHandlers.slideVisible = {  
  init: function (element, valueAccessor) {  
    // Initially set the element to be instantly visible/hidden depending on the value  
    var value = valueAccessor();  
    $(element).toggle(ko.unwrap(value)); // Use "unwrap" so we can handle values that may or may not be observable  
  },  
  update: function (element, valueAccessor, allBindingsAccessor) {  
    // First get the latest data that we're bound to  
    var value = valueAccessor(), allBindings = allBindingsAccessor();  
  
    // Next, whether or not the supplied model property is observable, get its current value  
    var valueUnwrapped = ko.utils.unwrapObservable(value);  
  
    // Grab some more data from another binding property  
    var duration = allBindings.slideDuration || 400; // 400ms is default duration unless otherwise specified  
  
    // Now manipulate the DOM element  
    if (valueUnwrapped == true)  
      $(element).slideDown(duration); // Make the element visible  
    else  
      $(element).slideUp(duration); // Make the element invisible  
  }  
};
```

```
<div class="cpm-panel" data-bind="slideVisible: isCpmOpened, slideDuration: 450">
```