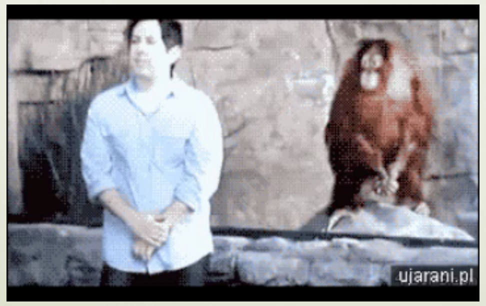
Simple Data Binding

jacques woodcock

When I change, you change.



jacques woodcock

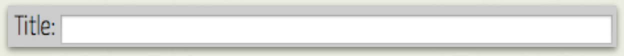
Data binding is simply the act of establishing a contract between an element and a data model.

3 Part Contract

jacques woodcock

Data binding is simply the act of establishing a relationship between an html element and a data model.

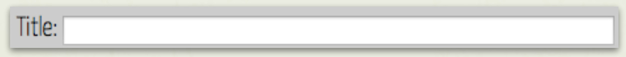
3 Part Contract



jacques woodcock

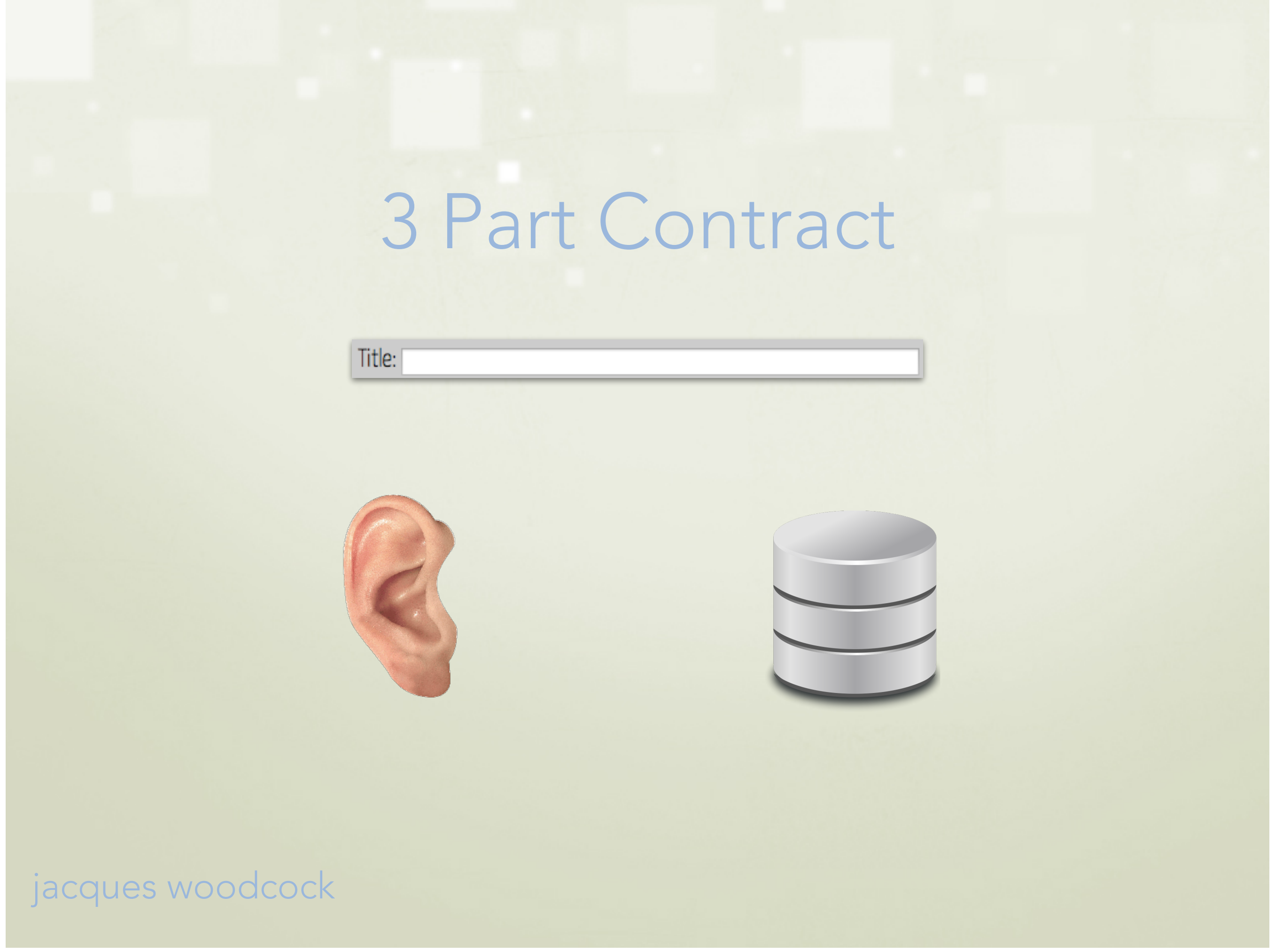
Data binding is simply the act of establishing a relationship between an html element and a data model.

3 Part Contract



jacques woodcock

Data binding is simply the act of establishing a relationship between an html element and a data model.



3 Part Contract

Title:

jacques woodcock

Of course each one of these has some subtasks, but in a nutshell, this is the binding structure.

How do these relate? Let’s start at the bottom and work up in a simple graphic.

jacques woodcock

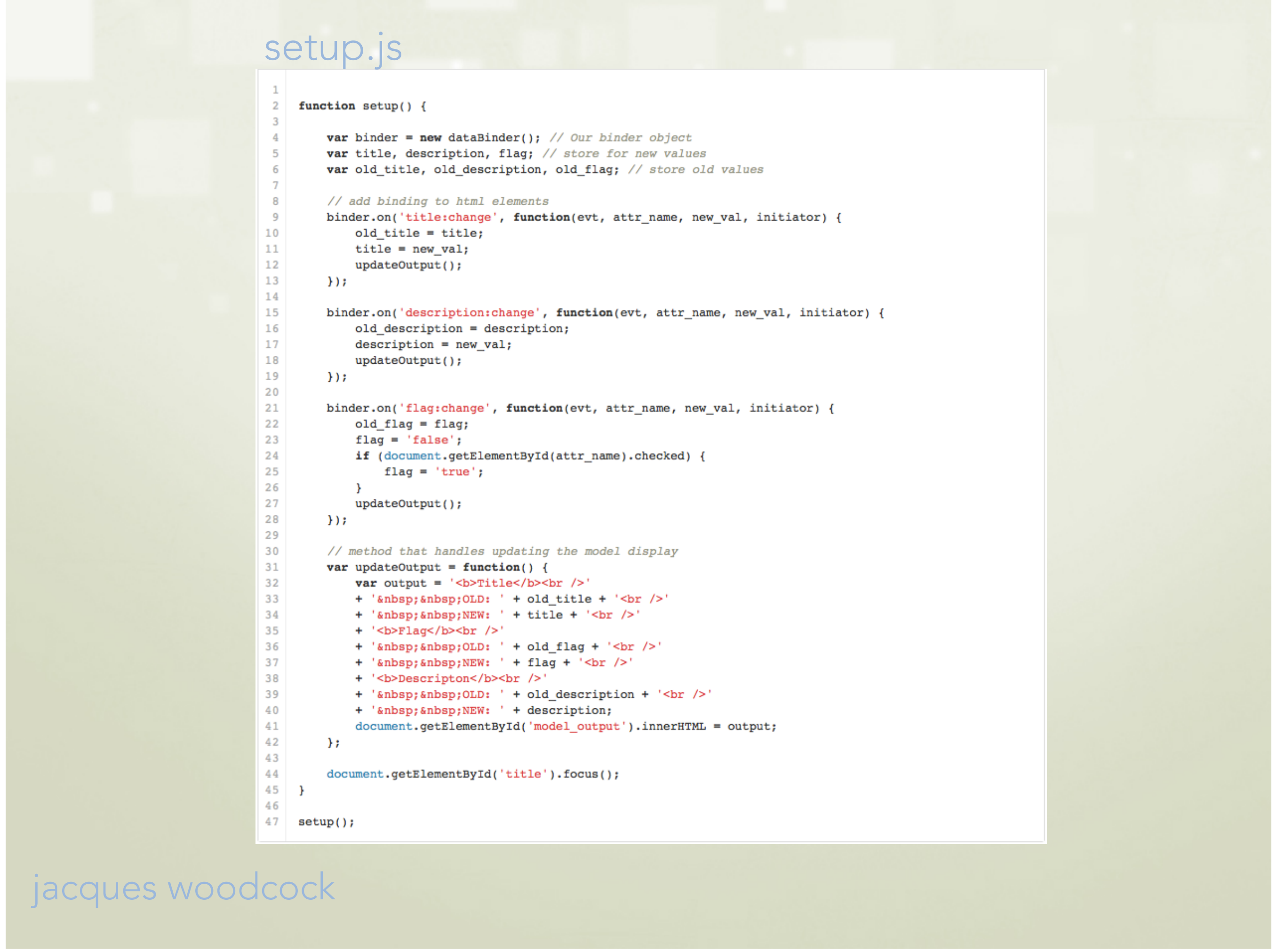
First, you need a mechanism for defining what html elements are tracked. This can happen in a setup script but most often happens in a simple model object. Here we'll just use a setup script, lightening talk you know.

First, you need a mechanism for defining what html elements are tracked. This can happen in a setup script but most often happens in a simple model object. Here we'll just use a setup script, lightening talk you know.

Next, you need an to handle the binding of event to html object. So we’ll add in a handler for that.

Finally, you need your binding data store. The event handler will pass on the event request which will make sure the right event was triggered. The data store itself will make sure there is a valid callback to execute.

So looking at this, you can probably tell the majority of the complexity is in the data store. To see it all, let's look at some code following the same steps.

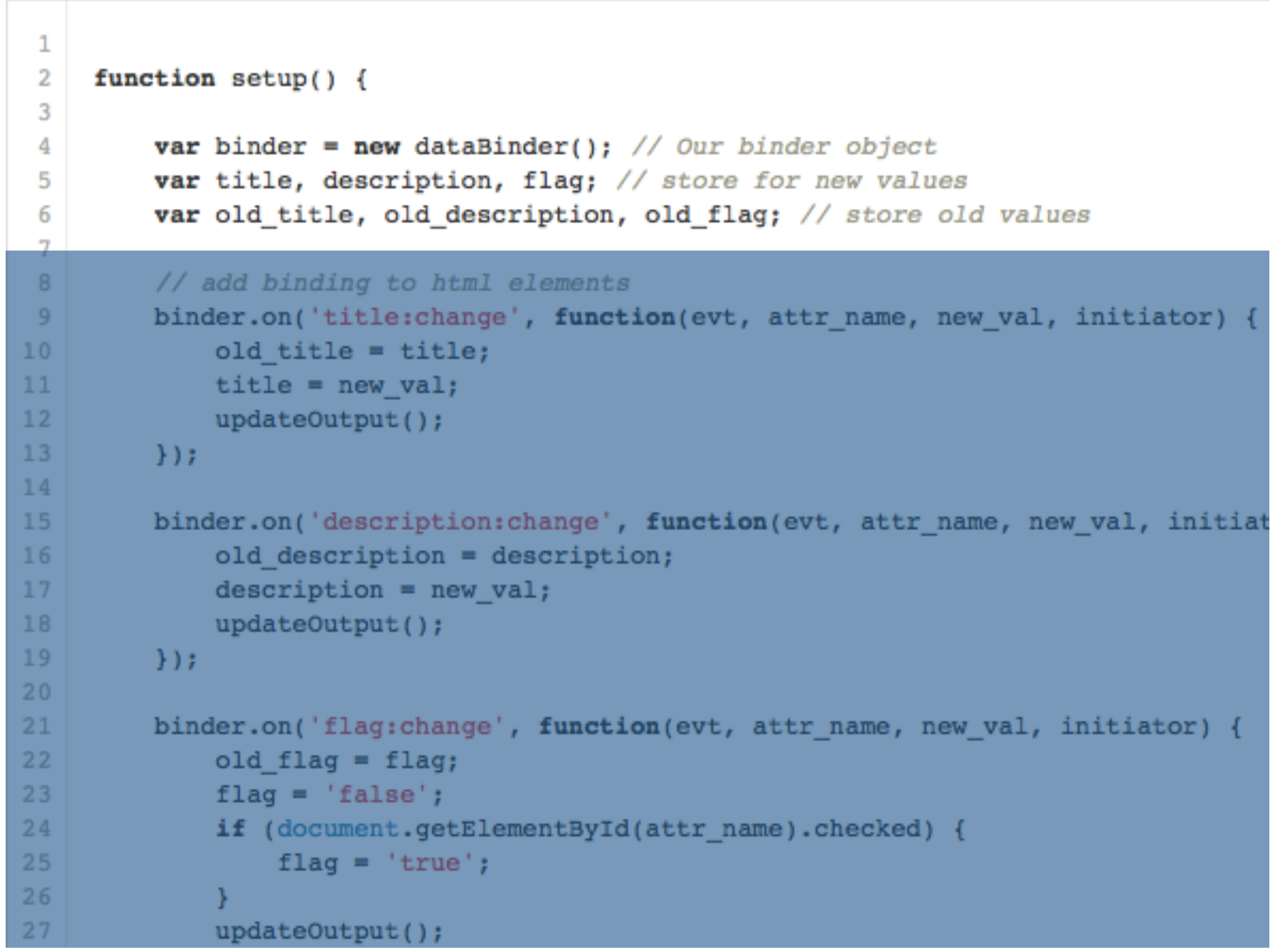


setup.js

```javascript

function setup() {

    var binder = new dataBinder(); // Our binder object
    var title, description, flag; // store for new values
    var old_title, old_description, old_flag; // store old values

    // add binding to html elements
    binder.on('title:change', function(evt, attr_name, new_val, initiator) {
        old_title = title;
        title = new_val;
        updateOutput();
    });

    binder.on('description:change', function(evt, attr_name, new_val, initiator) {
        old_description = description;
        description = new_val;
        updateOutput();
    });

    binder.on('flag:change', function(evt, attr_name, new_val, initiator) {
        old_flag = flag;
        flag = 'false';
        if (document.getElementById(attr_name).checked) {
            flag = 'true';
        }
        updateOutput();
    });

    // method that handles updating the model display
    var updateOutput = function() {
        var output = '<b>Title</b><br />'
        + '&nbsp;&nbsp;OLD: ' + old_title + '<br />'
        + '&nbsp;&nbsp;NEW: ' + title + '<br />'
        + '<b>Flag</b><br />'
        + '&nbsp;&nbsp;OLD: ' + old_flag + '<br />'
        + '&nbsp;&nbsp;NEW: ' + flag + '<br />'
        + '<b>Descripton</b><br />'
        + '&nbsp;&nbsp;OLD: ' + old_description + '<br />'
        + '&nbsp;&nbsp;NEW: ' + description;
        document.getElementById('model_output').innerHTML = output;
    };

    document.getElementById('title').focus();
}

setup();
```

jacques woodcock

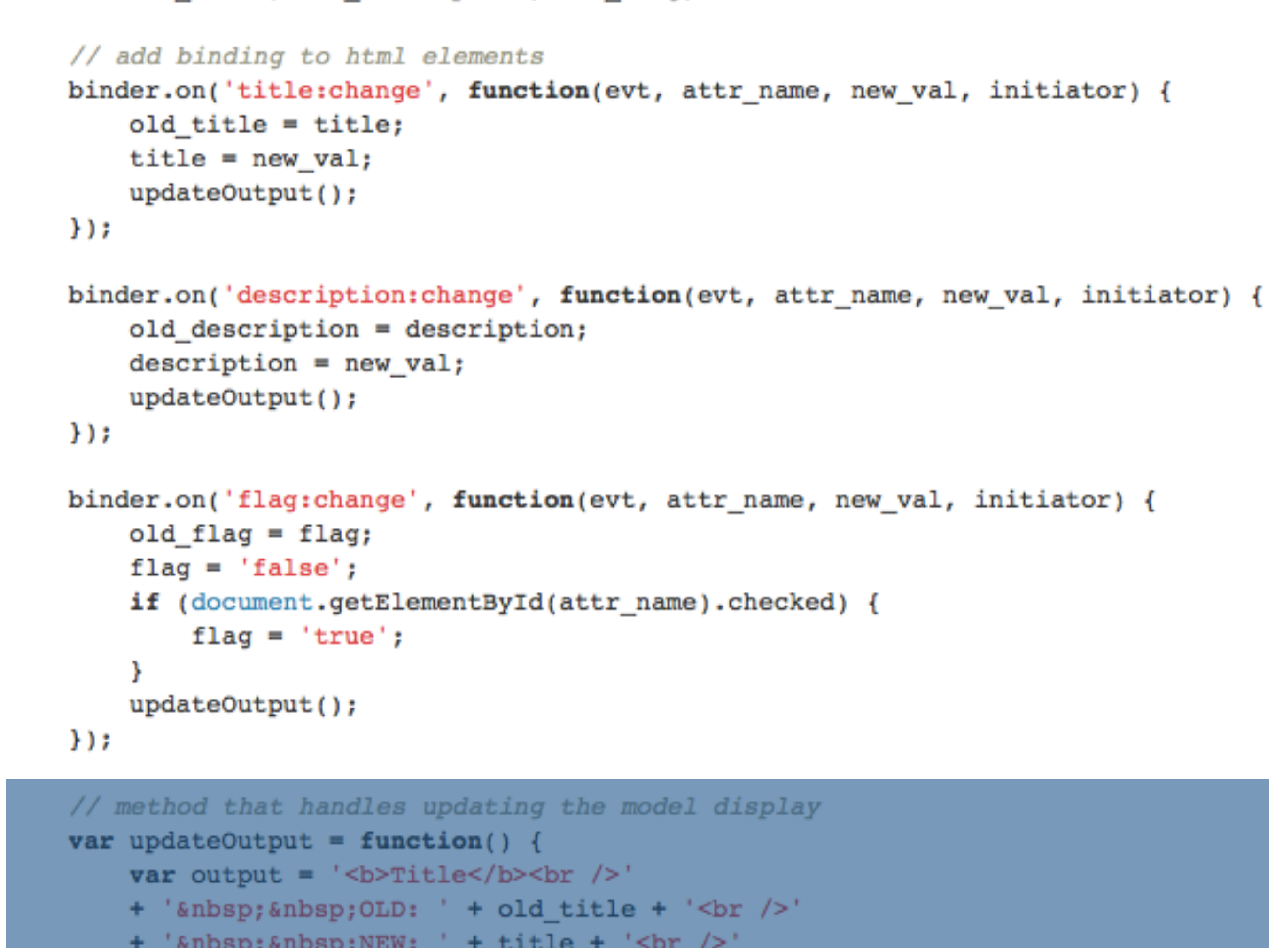
Our setup script is pretty simple. We'll use it to bind our html elements to binding data store and event handler. We'll also add some basic model functionality. Just enough to to how the binding works



```

function setup() {

    var binder = new dataBinder(); // Our binder object
    var title, description, flag; // store for new values
    var old_title, old_description, old_flag; // store old values

    // add binding to html elements
    binder.on('title:change', function(evt, attr_name, new_val, initiator) {
        old_title = title;
        title = new_val;
        updateOutput();
    });

    binder.on('description:change', function(evt, attr_name, new_val, initiat
        old_description = description;
        description = new_val;
        updateOutput();
    });

    binder.on('flag:change', function(evt, attr_name, new_val, initiator) {
        old_flag = flag;
        flag = 'false';
        if (document.getElementById(attr_name).checked) {
            flag = 'true';
        }
        updateOutput();
```

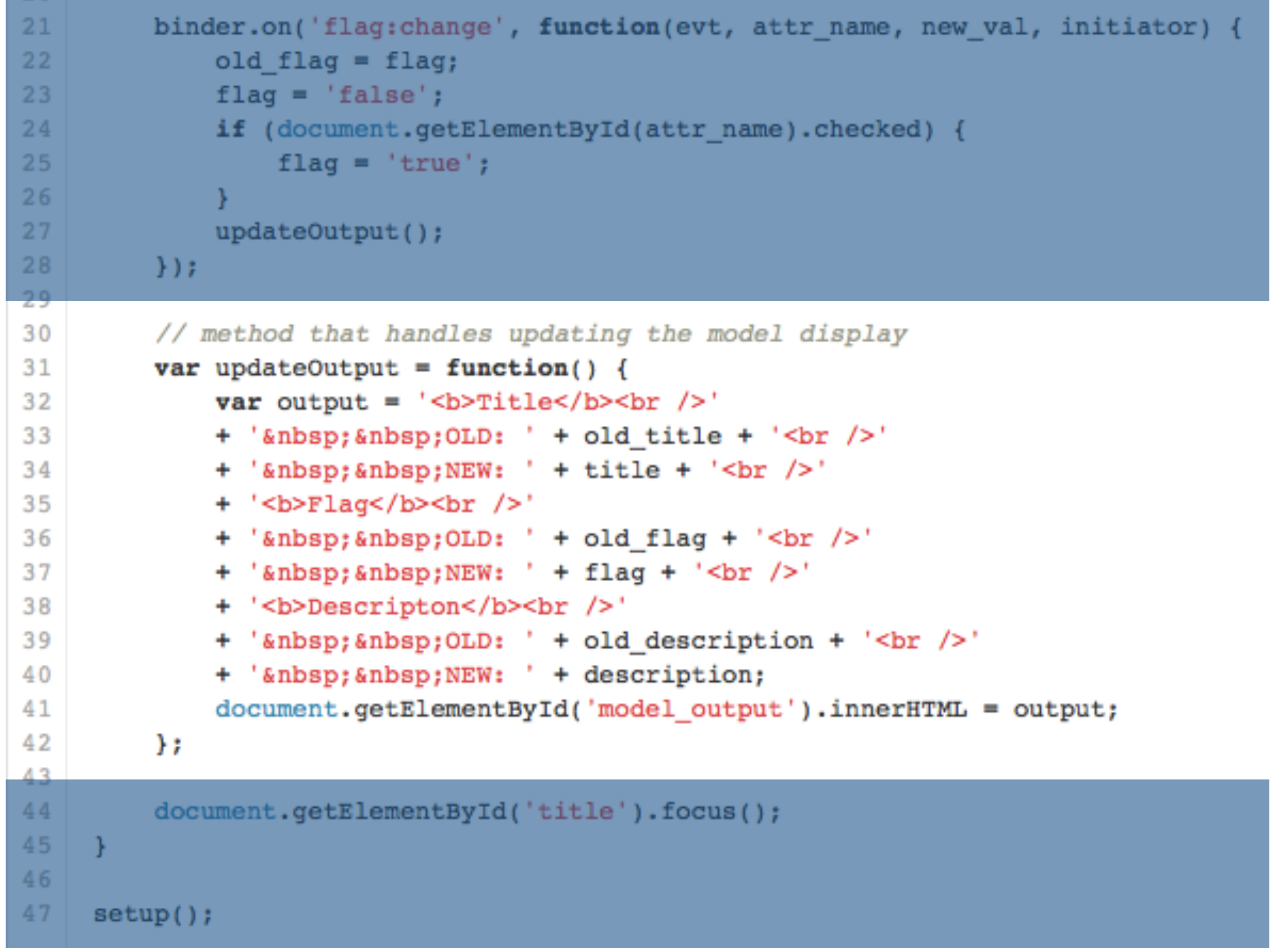
Going to put setup in a class to control when it’s called.

Instantiate the binder, define variables to mimic a data model.



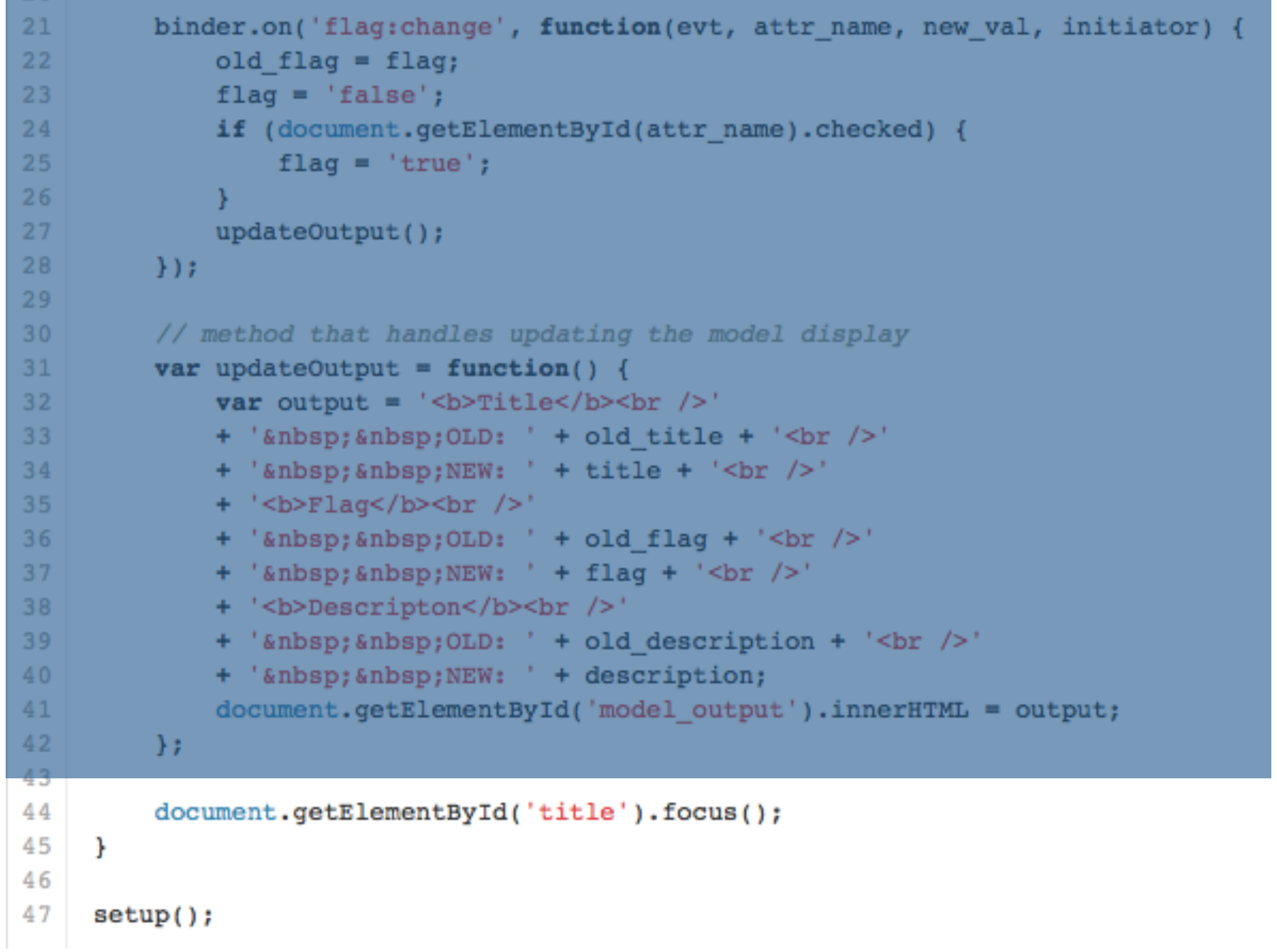
```
// add binding to html elements
binder.on('title:change', function(evt, attr_name, new_val, initiator) {
    old_title = title;
    title = new_val;
    updateOutput();
});

binder.on('description:change', function(evt, attr_name, new_val, initiator) {
    old_description = description;
    description = new_val;
    updateOutput();
});

binder.on('flag:change', function(evt, attr_name, new_val, initiator) {
    old_flag = flag;
    flag = 'false';
    if (document.getElementById(attr_name).checked) {
        flag = 'true';
    }
    updateOutput();
});

// method that handles updating the model display
var updateOutput = function() {
    var output = '<b>Title</b><br />'
    + '&nbsp;&nbsp;OLD: ' + old_title + '<br />'
    + '&nbsp;&nbsp;NEW: ' + title + '<br />'
```

Attach bindings to dom elements.



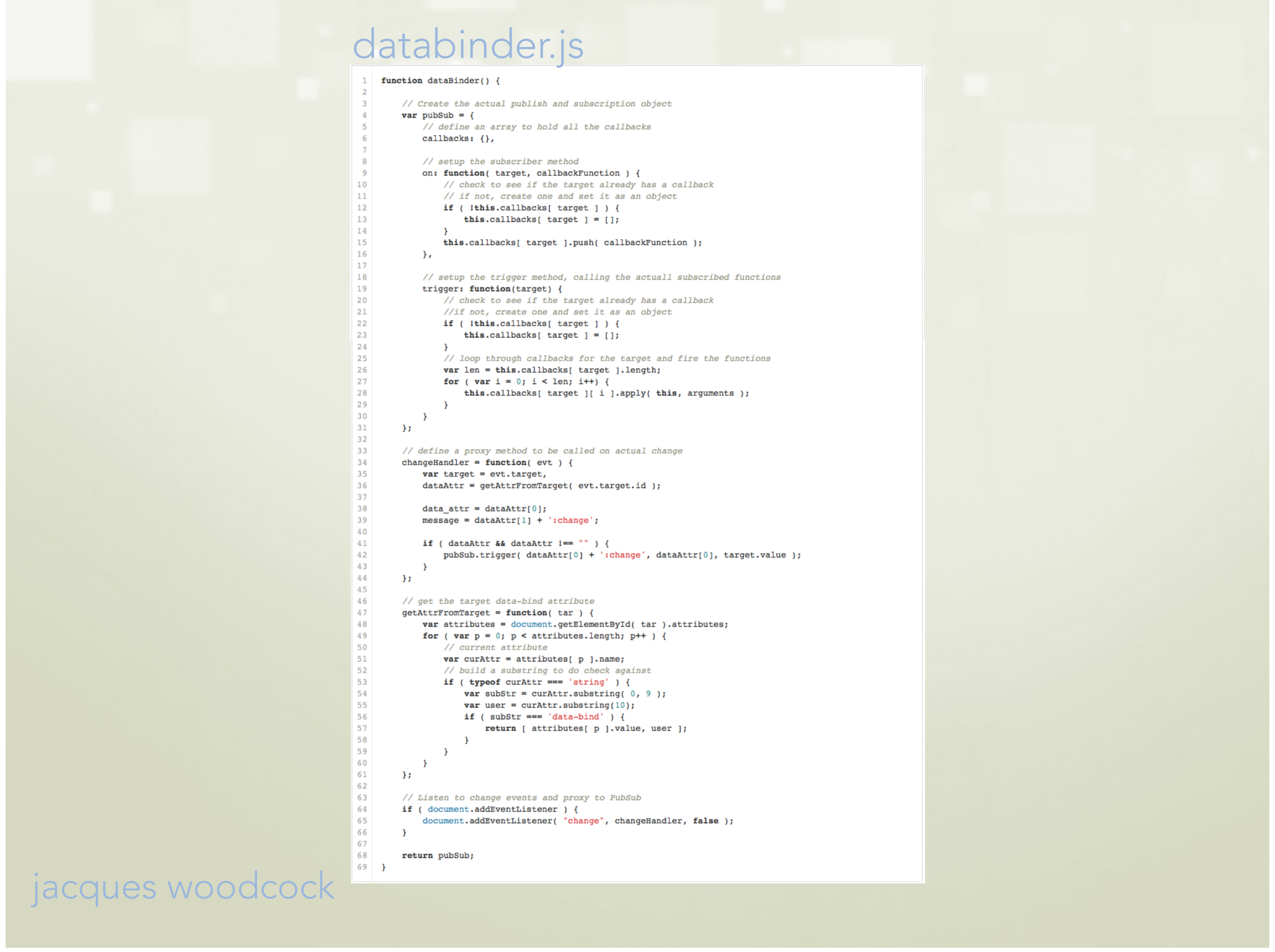
```
    binder.on('flag:change', function(evt, attr_name, new_val, initiator) {
        old_flag = flag;
        flag = 'false';
        if (document.getElementById(attr_name).checked) {
            flag = 'true';
        }
        updateOutput();
    });

    // method that handles updating the model display
    var updateOutput = function() {
        var output = '<b>Title</b><br />'
        + '&nbsp;&nbsp;OLD: ' + old_title + '<br />'
        + '&nbsp;&nbsp;NEW: ' + title + '<br />'
        + '<b>Flag</b><br />'
        + '&nbsp;&nbsp;OLD: ' + old_flag + '<br />'
        + '&nbsp;&nbsp;NEW: ' + flag + '<br />'
        + '<b>Descripton</b><br />'
        + '&nbsp;&nbsp;OLD: ' + old_description + '<br />'
        + '&nbsp;&nbsp;NEW: ' + description;
        document.getElementById('model_output').innerHTML = output;
    };

    document.getElementById('title').focus();
}

setup();
```

Method to update our display so we can see our small data model.



```
    binder.on('flag:change', function(evt, attr_name, new_val, initiator) {
        old_flag = flag;
        flag = 'false';
        if (document.getElementById(attr_name).checked) {
            flag = 'true';
        }
        updateOutput();
    });

    // method that handles updating the model display
    var updateOutput = function() {
        var output = '<b>Title</b><br />'
        + '&nbsp;&nbsp;OLD: ' + old_title + '<br />'
        + '&nbsp;&nbsp;NEW: ' + title + '<br />'
        + '<b>Flag</b><br />'
        + '&nbsp;&nbsp;OLD: ' + old_flag + '<br />'
        + '&nbsp;&nbsp;NEW: ' + flag + '<br />'
        + '<b>Descripton</b><br />'
        + '&nbsp;&nbsp;OLD: ' + old_description + '<br />'
        + '&nbsp;&nbsp;NEW: ' + description;
        document.getElementById('model_output').innerHTML = output;
    };

    document.getElementById('title').focus();
}

setup();
```

Set focus for usability and call the setup.

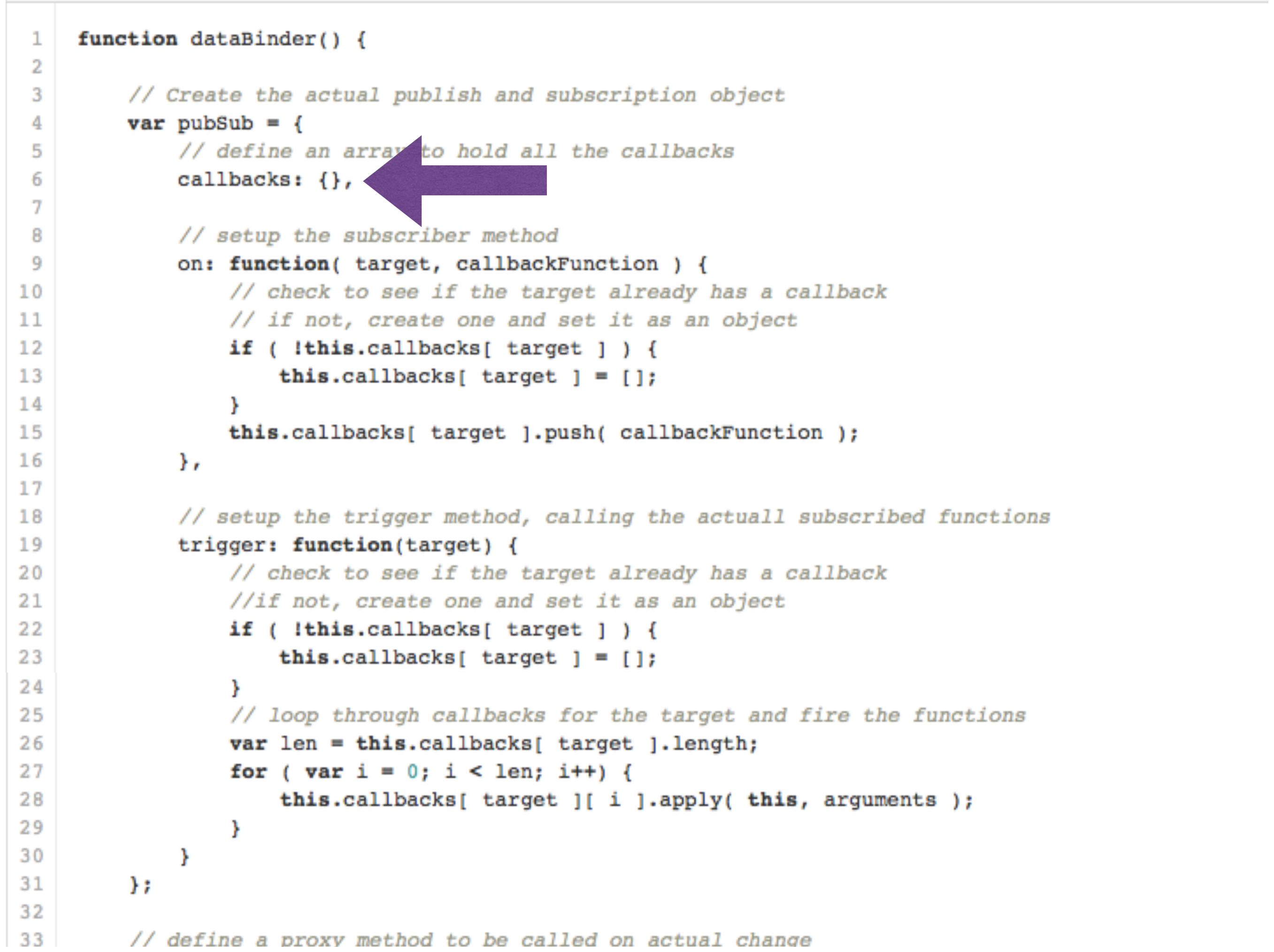


databinder.js

```javascript
function dataBinder() {

    // Create the actual publish and subscription object
    var pubSub = {
        // define an array to hold all the callbacks
        callbacks: {},

        // setup the subscriber method
        on: function( target, callbackFunction ) {
            // check to see if the target already has a callback
            // if not, create one and set it as an object
            if ( !this.callbacks[ target ] ) {
                this.callbacks[ target ] = [];
            }
            this.callbacks[ target ].push( callbackFunction );
        },

        // setup the trigger method, calling the actuall subscribed functions
        trigger: function(target) {
            // check to see if the target already has a callback
            //if not, create one and set it as an object
            if ( !this.callbacks[ target ] ) {
                this.callbacks[ target ] = [];
            }
            // loop through callbacks for the target and fire the functions
            var len = this.callbacks[ target ].length;
            for ( var i = 0; i < len; i++) {
                this.callbacks[ target ][ i ].apply( this, arguments );
            }
        }
    };

    // define a proxy method to be called on actual change
    changeHandler = function( evt ) {
        var target = evt.target,
        dataAttr = getAttrFromTarget( evt.target.id );

        data_attr = dataAttr[0];
        message = dataAttr[1] + ':change';

        if ( dataAttr && dataAttr !== "" ) {
            pubSub.trigger( dataAttr[0] + ':change', dataAttr[0], target.value );
        }
    };

    // get the target data-bind attribute
    getAttrFromTarget = function( tar ) {
        var attributes = document.getElementById( tar ).attributes;
        for ( var p = 0; p < attributes.length; p++ ) {
            // current attribute
            var curAttr = attributes[ p ].name;
            // build a substring to do check against
            if ( typeof curAttr === 'string' ) {
                var subStr = curAttr.substring( 0, 9 );
                var user = curAttr.substring(10);
                if ( subStr === 'data-bind' ) {
                    return [ attributes[ p ].value, user ];
                }
            }
        }
    };

    // Listen to change events and proxy to PubSub
    if ( document.addEventListener ) {
        document.addEventListener( "change", changeHandler, false );
    }

    return pubSub;
}
```

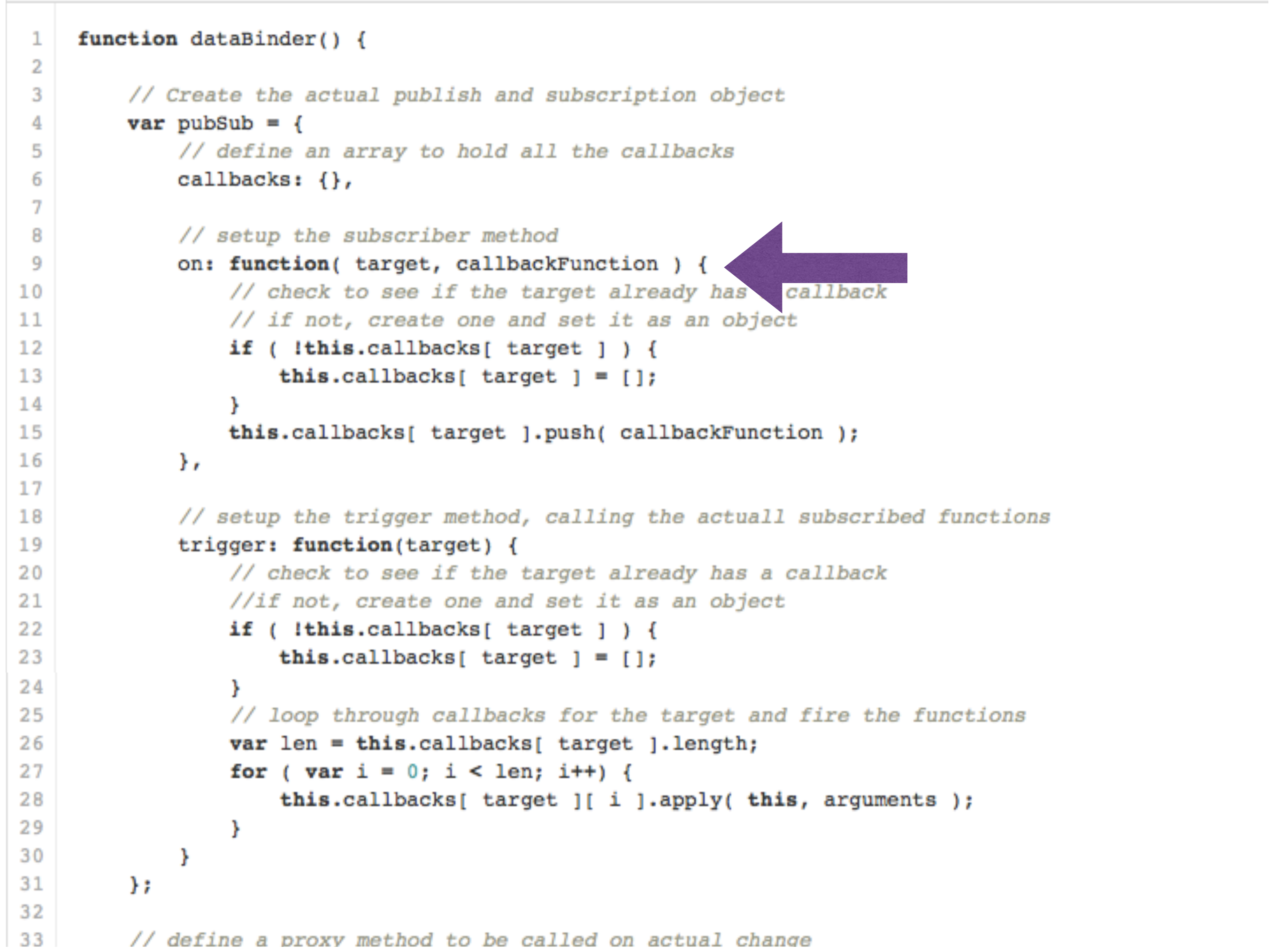
jacques woodcock

Our setup script is pretty simple. We'll use it to bind our html elements to binding data store and event handler. We'll also add some basic model functionality. Just enough to to how the binding works



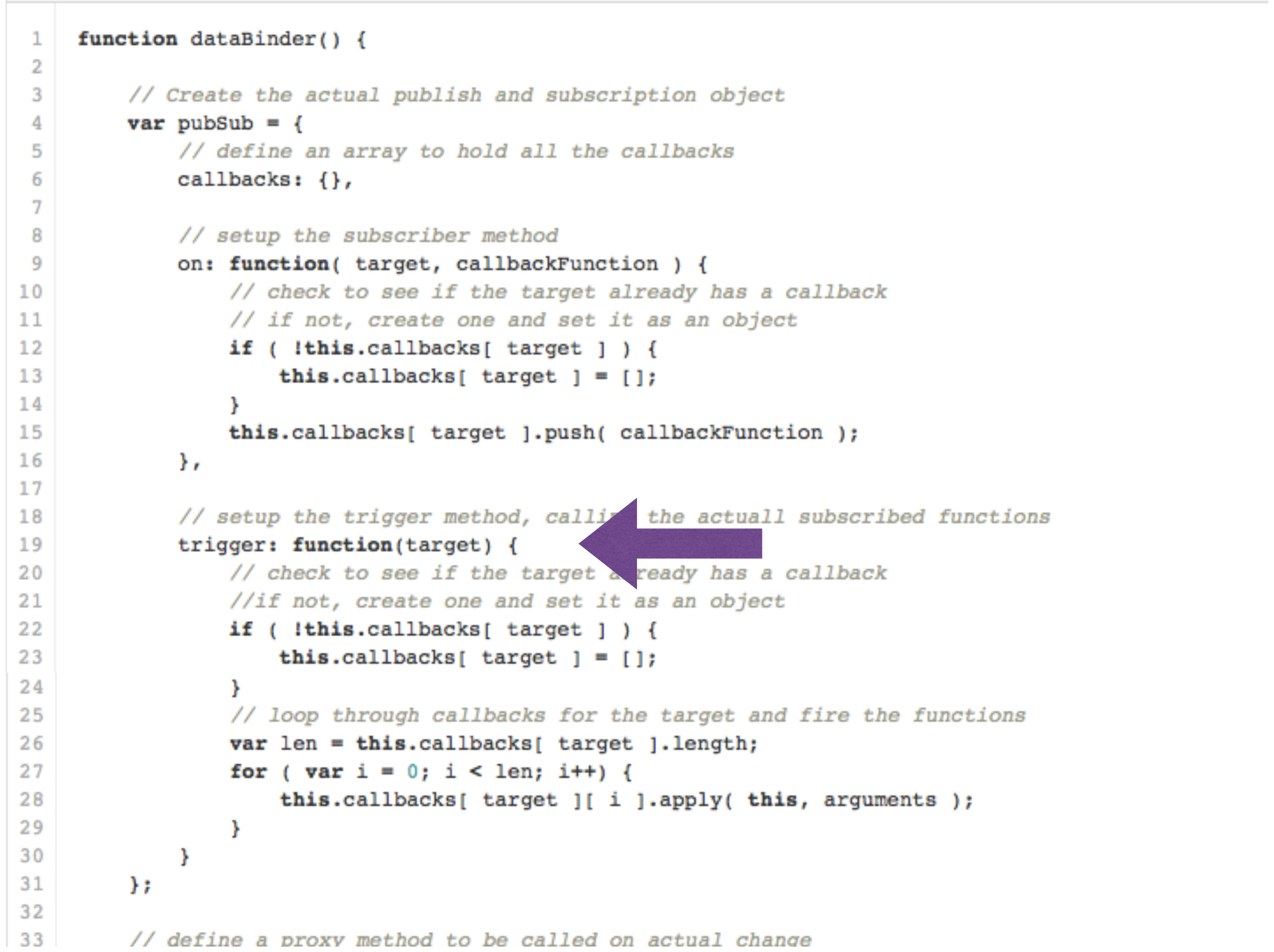
```
function dataBinder() {

    // Create the actual publish and subscription object
    var pubSub = {
        // define an array to hold all the callbacks
        callbacks: {},

        // setup the subscriber method
        on: function( target, callbackFunction ) {
            // check to see if the target already has a callback
            // if not, create one and set it as an object
            if ( !this.callbacks[ target ] ) {
                this.callbacks[ target ] = [];
            }
            this.callbacks[ target ].push( callbackFunction );
        },

        // setup the trigger method, calling the actuall subscribed functions
        trigger: function(target) {
            // check to see if the target already has a callback
            //if not, create one and set it as an object
            if ( !this.callbacks[ target ] ) {
                this.callbacks[ target ] = [];
            }
            // loop through callbacks for the target and fire the functions
            var len = this.callbacks[ target ].length;
            for ( var i = 0; i < len; i++) {
                this.callbacks[ target ][ i ].apply( this, arguments );
            }
        }
    };

    // define a proxy method to be called on actual change
```

The actual data store.



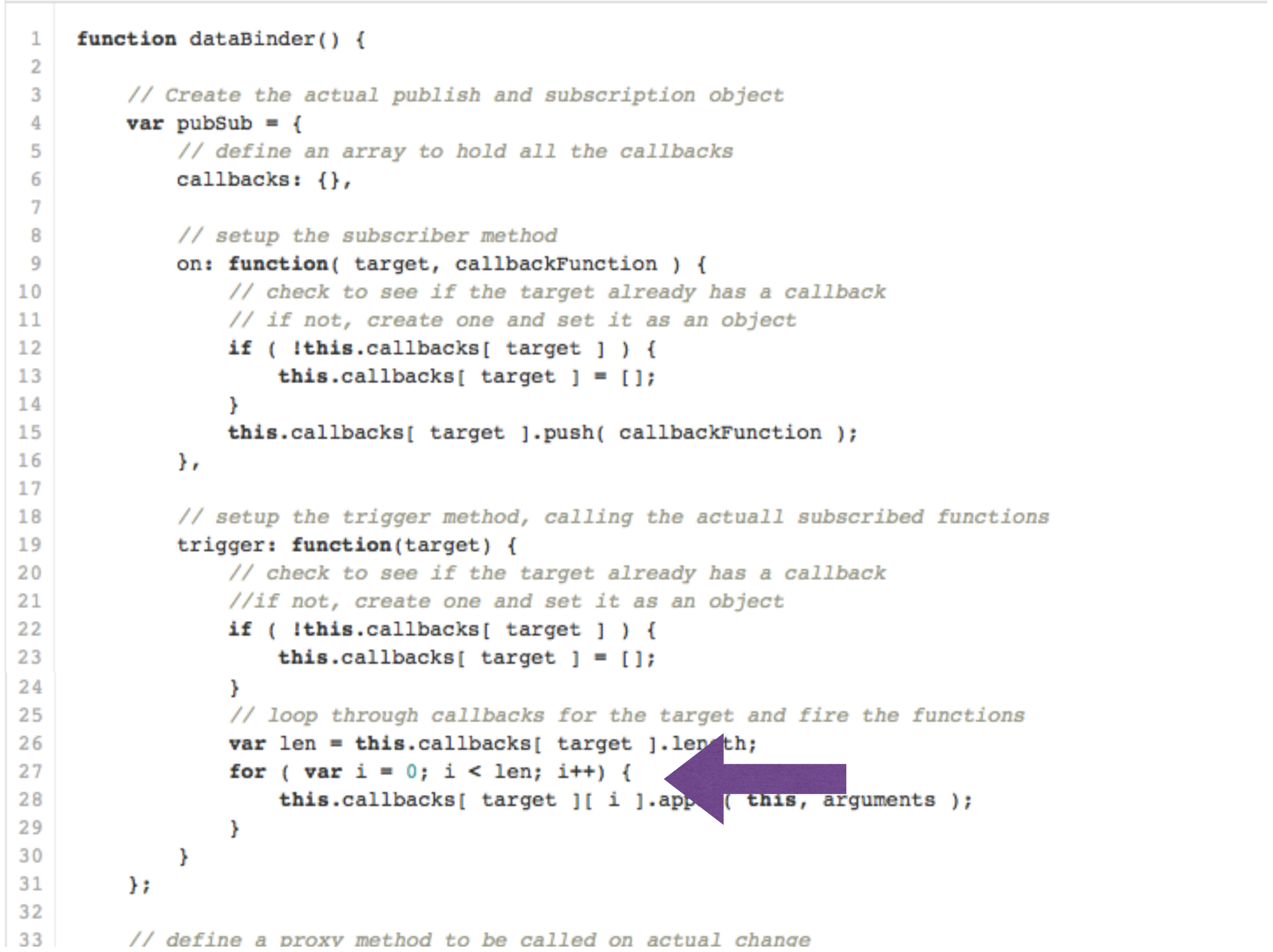
```
function dataBinder() {

    // Create the actual publish and subscription object
    var pubSub = {
        // define an array to hold all the callbacks
        callbacks: {},

        // setup the subscriber method
        on: function( target, callbackFunction ) {
            // check to see if the target already has a callback
            // if not, create one and set it as an object
            if ( !this.callbacks[ target ] ) {
                this.callbacks[ target ] = [];
            }
            this.callbacks[ target ].push( callbackFunction );
        },

        // setup the trigger method, calling the actuall subscribed functions
        trigger: function(target) {
            // check to see if the target already has a callback
            //if not, create one and set it as an object
            if ( !this.callbacks[ target ] ) {
                this.callbacks[ target ] = [];
            }
            // loop through callbacks for the target and fire the functions
            var len = this.callbacks[ target ].length;
            for ( var i = 0; i < len; i++) {
                this.callbacks[ target ][ i ].apply( this, arguments );
            }
        }
    };

    // define a proxy method to be called on actual change
```

Adding to data store.



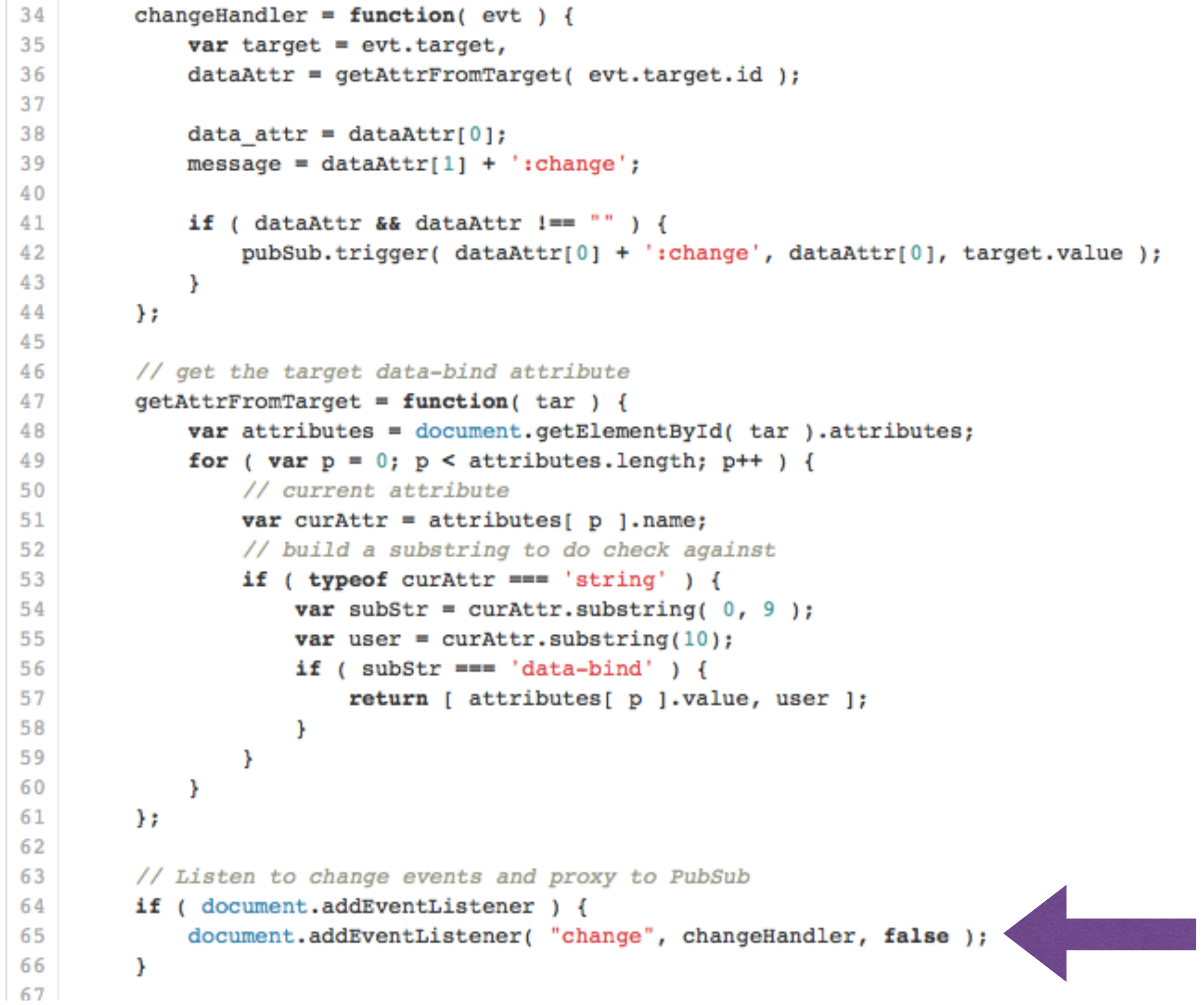
```
function dataBinder() {

    // Create the actual publish and subscription object
    var pubSub = {
        // define an array to hold all the callbacks
        callbacks: {},

        // setup the subscriber method
        on: function( target, callbackFunction ) {
            // check to see if the target already has a callback
            // if not, create one and set it as an object
            if ( !this.callbacks[ target ] ) {
                this.callbacks[ target ] = [];
            }
            this.callbacks[ target ].push( callbackFunction );
        },

        // setup the trigger method, calli  the actuall subscribed functions
        trigger: function(target) {
            // check to see if the target   ready has a callback
            //if not, create one and set it as an object
            if ( !this.callbacks[ target ] ) {
                this.callbacks[ target ] = [];
            }
            // loop through callbacks for the target and fire the functions
            var len = this.callbacks[ target ].length;
            for ( var i = 0; i < len; i++) {
                this.callbacks[ target ][ i ].apply( this, arguments );
            }
        }
    };

    // define a proxy method to be called on actual change
```

Triggering stored bindings.



```
function dataBinder() {

    // Create the actual publish and subscription object
    var pubSub = {
        // define an array to hold all the callbacks
        callbacks: {},

        // setup the subscriber method
        on: function( target, callbackFunction ) {
            // check to see if the target already has a callback
            // if not, create one and set it as an object
            if ( !this.callbacks[ target ] ) {
                this.callbacks[ target ] = [];
            }
            this.callbacks[ target ].push( callbackFunction );
        },

        // setup the trigger method, calling the actuall subscribed functions
        trigger: function(target) {
            // check to see if the target already has a callback
            //if not, create one and set it as an object
            if ( !this.callbacks[ target ] ) {
                this.callbacks[ target ] = [];
            }
            // loop through callbacks for the target and fire the functions
            var len = this.callbacks[ target ].len[illegible]th;
            for ( var i = 0; i < len; i++) {
                this.callbacks[ target ][ i ].app[illegible]( this, arguments );
            }
        }
    };

    // define a proxy method to be called on actual change
```

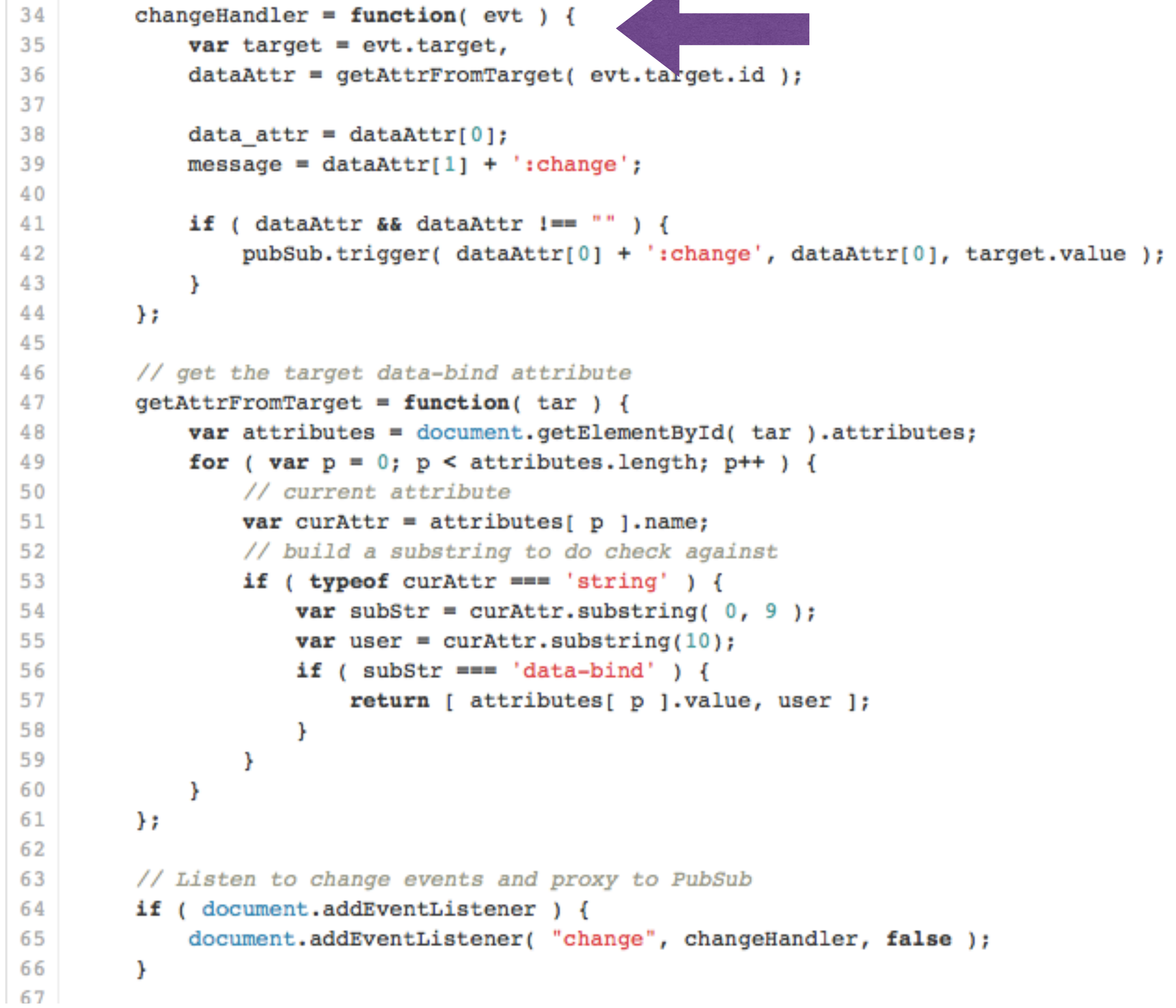
Notice the ability to have multiple stores per element.



```
    changeHandler = function( evt ) {
        var target = evt.target,
        dataAttr = getAttrFromTarget( evt.target.id );

        data_attr = dataAttr[0];
        message = dataAttr[1] + ':change';

        if ( dataAttr && dataAttr !== "" ) {
            pubSub.trigger( dataAttr[0] + ':change', dataAttr[0], target.value );
        }
    };

    // get the target data-bind attribute
    getAttrFromTarget = function( tar ) {
        var attributes = document.getElementById( tar ).attributes;
        for ( var p = 0; p < attributes.length; p++ ) {
            // current attribute
            var curAttr = attributes[ p ].name;
            // build a substring to do check against
            if ( typeof curAttr === 'string' ) {
                var subStr = curAttr.substring( 0, 9 );
                var user = curAttr.substring(10);
                if ( subStr === 'data-bind' ) {
                    return [ attributes[ p ].value, user ];
                }
            }
        }
    };

    // Listen to change events and proxy to PubSub
    if ( document.addEventListener ) {
        document.addEventListener( "change", changeHandler, false );
    }
```

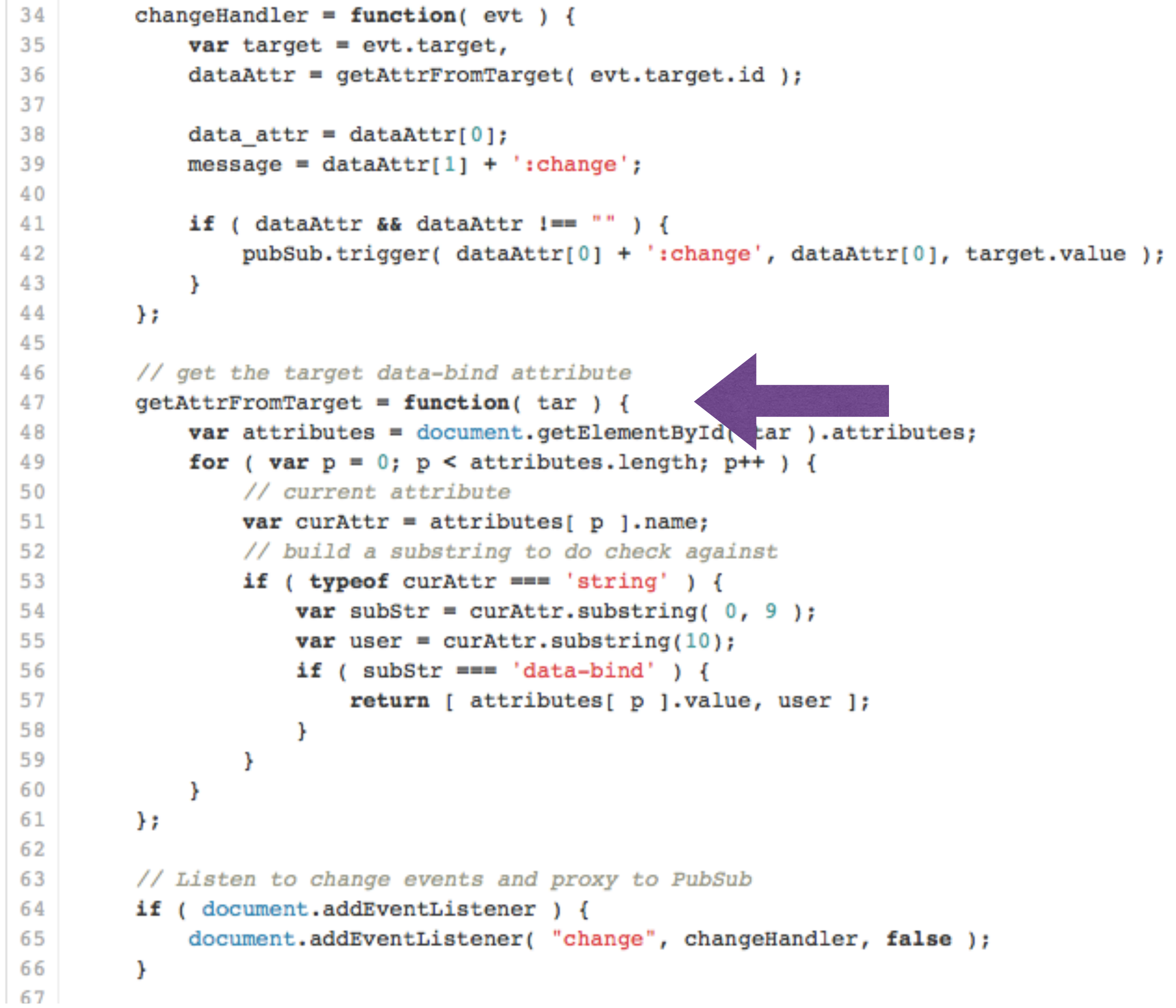
Also, to simply our scripts, we'll put our binding handler into the actual data store method.

Add a global listener for the document that defines what method processes the changes.



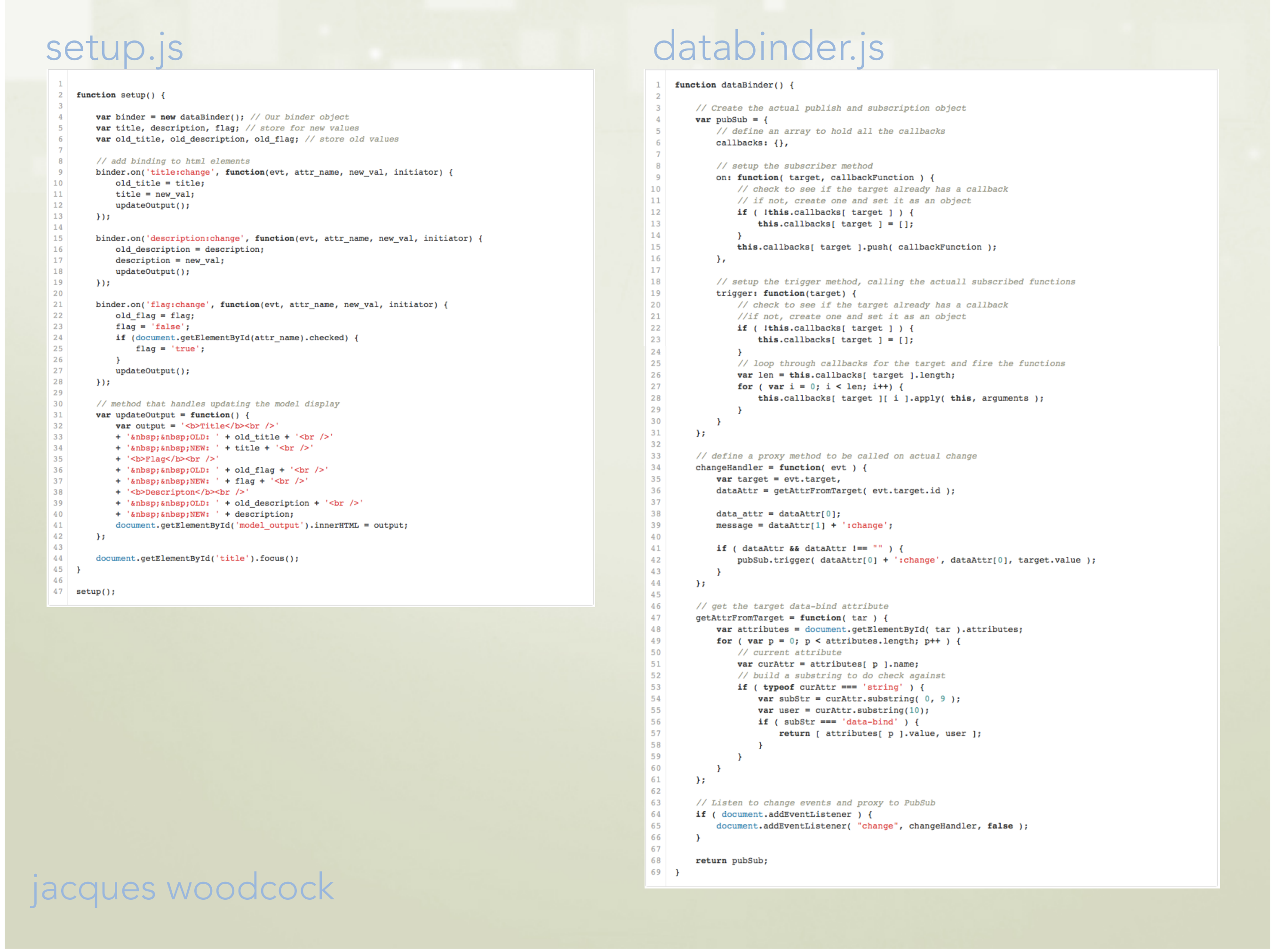
```
    changeHandler = function( evt ) {
        var target = evt.target,
        dataAttr = getAttrFromTarget( evt.target.id );

        data_attr = dataAttr[0];
        message = dataAttr[1] + ':change';

        if ( dataAttr && dataAttr !== "" ) {
            pubSub.trigger( dataAttr[0] + ':change', dataAttr[0], target.value );
        }
    };

    // get the target data-bind attribute
    getAttrFromTarget = function( tar ) {
        var attributes = document.getElementById( tar ).attributes;
        for ( var p = 0; p < attributes.length; p++ ) {
            // current attribute
            var curAttr = attributes[ p ].name;
            // build a substring to do check against
            if ( typeof curAttr === 'string' ) {
                var subStr = curAttr.substring( 0, 9 );
                var user = curAttr.substring(10);
                if ( subStr === 'data-bind' ) {
                    return [ attributes[ p ].value, user ];
                }
            }
        }
    };

    // Listen to change events and proxy to PubSub
    if ( document.addEventListener ) {
        document.addEventListener( "change", changeHandler, false );
    }
```

Method that handles the changes and calls the data store which looks to see if an event was stored for element.



```
    changeHandler = function( evt ) {
        var target = evt.target,
        dataAttr = getAttrFromTarget( evt.target.id );

        data_attr = dataAttr[0];
        message = dataAttr[1] + ':change';

        if ( dataAttr && dataAttr !== "" ) {
            pubSub.trigger( dataAttr[0] + ':change', dataAttr[0], target.value );
        }
    };

    // get the target data-bind attribute
    getAttrFromTarget = function( tar ) {
        var attributes = document.getElementById( tar ).attributes;
        for ( var p = 0; p < attributes.length; p++ ) {
            // current attribute
            var curAttr = attributes[ p ].name;
            // build a substring to do check against
            if ( typeof curAttr === 'string' ) {
                var subStr = curAttr.substring( 0, 9 );
                var user = curAttr.substring(10);
                if ( subStr === 'data-bind' ) {
                    return [ attributes[ p ].value, user ];
                }
            }
        }
    };

    // Listen to change events and proxy to PubSub
    if ( document.addEventListener ) {
        document.addEventListener( "change", changeHandler, false );
    }
```

Helper method to insure we’ve properly identified the element initiating the change.



setup.js

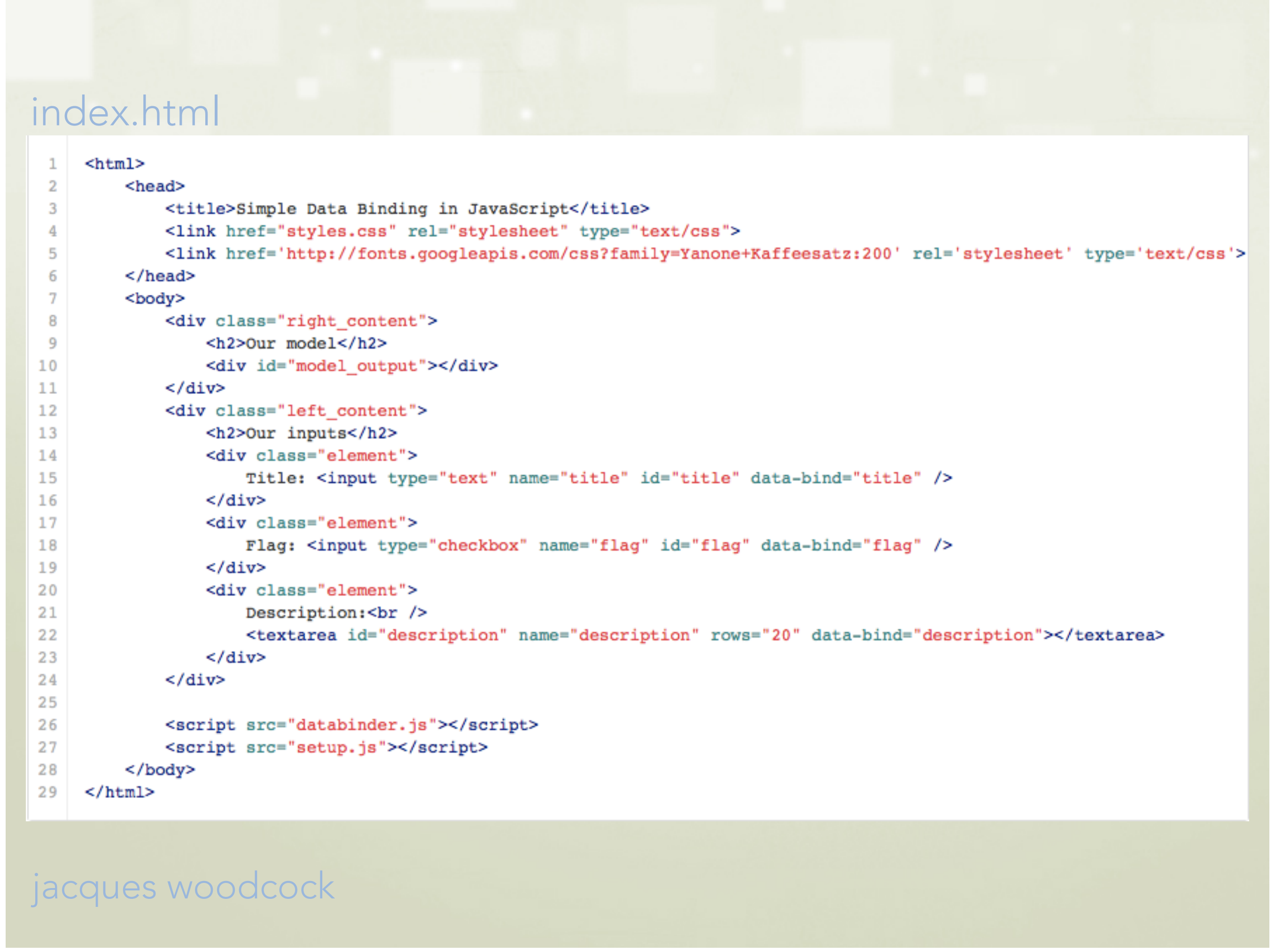
```
function setup() {

    var binder = new dataBinder(); // Our binder object
    var title, description, flag; // store for new values
    var old_title, old_description, old_flag; // store old values

    // add binding to html elements
    binder.on('title:change', function(evt, attr_name, new_val, initiator) {
        old_title = title;
        title = new_val;
        updateOutput();
    });

    binder.on('description:change', function(evt, attr_name, new_val, initiator) {
        old_description = description;
        description = new_val;
        updateOutput();
    });

    binder.on('flag:change', function(evt, attr_name, new_val, initiator) {
        old_flag = flag;
        flag = 'false';
        if (document.getElementById(attr_name).checked) {
            flag = 'true';
        }
        updateOutput();
    });

    // method that handles updating the model display
    var updateOutput = function() {
        var output = '<b>Title</b><br />'
        + '&nbsp;&nbsp;OLD: ' + old_title + '<br />'
        + '&nbsp;&nbsp;NEW: ' + title + '<br />'
        + '<b>Flag</b><br />'
        + '&nbsp;&nbsp;OLD: ' + old_flag + '<br />'
        + '&nbsp;&nbsp;NEW: ' + flag + '<br />'
        + '<b>Descripton</b><br />'
        + '&nbsp;&nbsp;OLD: ' + old_description + '<br />'
        + '&nbsp;&nbsp;NEW: ' + description;
        document.getElementById('model_output').innerHTML = output;
    };

    document.getElementById('title').focus();
}

setup();
```

databinder.js

```
function dataBinder() {

    // Create the actual publish and subscription object
    var pubSub = {
        // define an array to hold all the callbacks
        callbacks: {},

        // setup the subscriber method
        on: function( target, callbackFunction ) {
            // check to see if the target already has a callback
            // if not, create one and set it as an object
            if ( !this.callbacks[ target ] ) {
                this.callbacks[ target ] = [];
            }
            this.callbacks[ target ].push( callbackFunction );
        },

        // setup the trigger method, calling the actuall subscribed functions
        trigger: function(target) {
            // check to see if the target already has a callback
            //if not, create one and set it as an object
            if ( !this.callbacks[ target ] ) {
                this.callbacks[ target ] = [];
            }
            // loop through callbacks for the target and fire the functions
            var len = this.callbacks[ target ].length;
            for ( var i = 0; i < len; i++) {
                this.callbacks[ target ][ i ].apply( this, arguments );
            }
        }
    };

    // define a proxy method to be called on actual change
    changeHandler = function( evt ) {
        var target = evt.target,
        dataAttr = getAttrFromTarget( evt.target.id );

        data_attr = dataAttr[0];
        message = dataAttr[1] + ':change';

        if ( dataAttr && dataAttr !== "" ) {
            pubSub.trigger( dataAttr[0] + ':change', dataAttr[0], target.value );
        }
    };

    // get the target data-bind attribute
    getAttrFromTarget = function( tar ) {
        var attributes = document.getElementById( tar ).attributes;
        for ( var p = 0; p < attributes.length; p++ ) {
            // current attribute
            var curAttr = attributes[ p ].name;
            // build a substring to do check against
            if ( typeof curAttr === 'string' ) {
                var subStr = curAttr.substring( 0, 9 );
                var user = curAttr.substring(10);
                if ( subStr === 'data-bind' ) {
                    return [ attributes[ p ].value, user ];
                }
            }
        }
    };

    // Listen to change events and proxy to PubSub
    if ( document.addEventListener ) {
        document.addEventListener( "change", changeHandler, false );
    }

    return pubSub;
}
```

jacques woodcock

With these two scripts, you now have a simple data-bound html document.

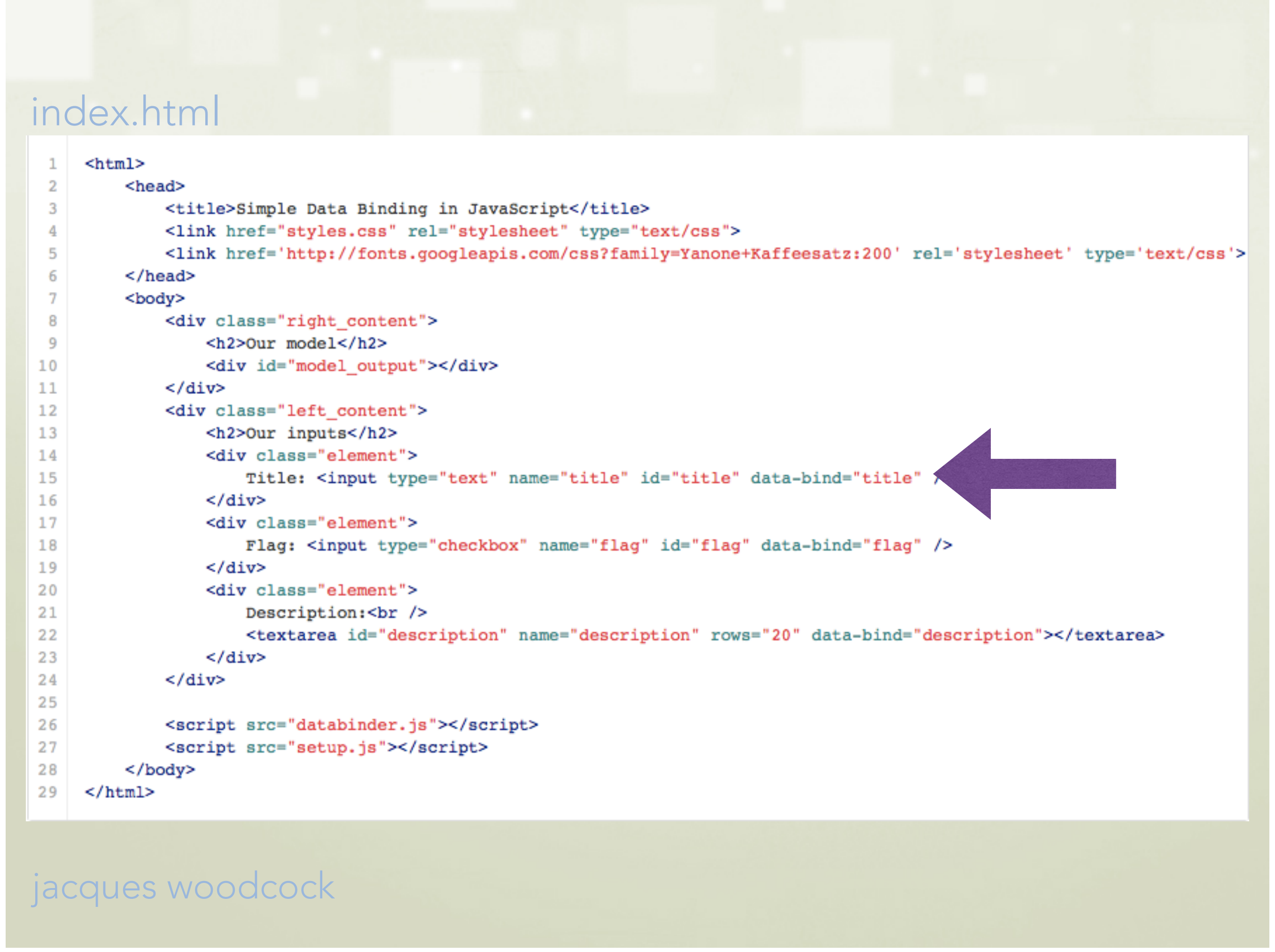


index.html

```html
<html>
    <head>
        <title>Simple Data Binding in JavaScript</title>
        <link href="styles.css" rel="stylesheet" type="text/css">
        <link href='http://fonts.googleapis.com/css?family=Yanone+Kaffeesatz:200' rel='stylesheet' type='text/css'>
    </head>
    <body>
        <div class="right_content">
            <h2>Our model</h2>
            <div id="model_output"></div>
        </div>
        <div class="left_content">
            <h2>Our inputs</h2>
            <div class="element">
                Title: <input type="text" name="title" id="title" data-bind="title" />
            </div>
            <div class="element">
                Flag: <input type="checkbox" name="flag" id="flag" data-bind="flag" />
            </div>
            <div class="element">
                Description:<br />
                <textarea id="description" name="description" rows="20" data-bind="description"></textarea>
            </div>
        </div>

        <script src="databinder.js"></script>
        <script src="setup.js"></script>
    </body>
</html>
```

jacques woodcock

Let’s take a look at the html, then we’ll see it in action.



index.html

```
<html>
    <head>
        <title>Simple Data Binding in JavaScript</title>
        <link href="styles.css" rel="stylesheet" type="text/css">
        <link href='http://fonts.googleapis.com/css?family=Yanone+Kaffeesatz:200' rel='stylesheet' type='text/css'>
    </head>
    <body>
        <div class="right_content">
            <h2>Our model</h2>
            <div id="model_output"></div>
        </div>
        <div class="left_content">
            <h2>Our inputs</h2>
            <div class="element">
                Title: <input type="text" name="title" id="title" data-bind="title"
            </div>
            <div class="element">
                Flag: <input type="checkbox" name="flag" id="flag" data-bind="flag" />
            </div>
            <div class="element">
                Description:<br />
                <textarea id="description" name="description" rows="20" data-bind="description"></textarea>
            </div>
        </div>

        <script src="databinder.js"></script>
        <script src="setup.js"></script>
    </body>
</html>
```

jacques woodcock

Here in the html is the most important tag. The data-bind. This tells the everything, what element this is. If there were two elements with the same tag, they both would be updated.

Let’s see it in action.

jacques woodcock

Here in the html is the most important tag. The data-bind. This tells the everything, what element this is. If there were two elements with the same tag, they both would be updated.

Simple Data Binding

https://github.com/jwoodcock/Simple-Data-Binding

jacques woodcock