The "with" binding

Purpose

The `with` binding creates a new binding context, so that descendant elements are bound in the context of a specified object.

Of course, you can arbitrarily nest `with` bindings along with the other control-flow bindings such as `if` and `foreach`.

Example 1

Here is a very basic example of switching the binding context to a child object. Notice that in the `data-bind` attributes, it is *not* necessary to prefix `latitude` or `longitude` with `coords.`, because the binding context is switched to `coords`.

```
<h1 data-bind="text: city"> </h1>
<p data-bind="with: coords">
    Latitude: <span data-bind="text: latitude"> </span>,
    Longitude: <span data-bind="text: longitude"> </span>
</p>

<script type="text/javascript">
    ko.applyBindings({
        city: "London",
        coords: {
            latitude:  51.5001524,
            longitude: -0.1262362
        }
    });
</script>
```

Example 2

This interactive example demonstrates that:

- The `with` binding will dynamically add or remove descendant elements depending on whether the associated value is `null`/`undefined` or not
- If you want to access data/functions from parent binding contexts, you can use special context properties such as `$parent` and `$root`.

Try it out:

Twitter account: @example Get tweets

Source code: View

```
<form data-bind="submit: getTweets">
    Twitter account:
    <input data-bind="value: twitterName" />
    <button type="submit">Get tweets</button>
</form>

<div data-bind="with: resultData">
    <h3>Recent tweets fetched at <span data-bind="text: retrievalDate"> </span>
    <ol data-bind="foreach: topTweets">
        <li data-bind="text: text"></li>
    </ol>

    <button data-bind="click: $parent.clearResults">Clear tweets</button>
</div>
```

Source code: View model

```
function AppViewModel() {
    var self = this;
    self.twitterName = ko.observable('@example');
```

```
    self.resultData = ko.observable(); // No initial value

    self.getTweets = function() {
        var name = self.twitterName(),
            simulatedResults = [
                { text: name + ' What a nice day.' },
                { text: name + ' Building some cool apps.' },
                { text: name + ' Just saw a famous celebrity eating lard. Yum.'
            ];

        self.resultData({ retrievalDate: new Date(), topTweets: simulatedResult
    }

    self.clearResults = function() {
        self.resultData(undefined);
    }
}

ko.applyBindings(new AppViewModel());
```

Parameters

- Main parameter

  The object that you want to use as the context for binding descendant elements.

  If the expression you supply evaluates to `null` or `undefined`, descendant elements will *not* be bound at all, but will instead be removed from the document.

  If the expression you supply involves any observable values, the expression will be re-evaluated whenever any of those observables change. Then, descendant elements will be cleared out, and **a new copy of the markup** will be added to your document and bound in the context of the new evaluation result.

- Additional parameters
  - None

Note 1: Using “with” without a container element

Just like other control flow elements such as `if` and `foreach`, you can use `with` without any container element to host it. This is useful if you need to use `with` in a place where it would not be legal to introduce a new container element just to hold the `with` binding. See the documentation for `if` or `foreach` for more details.

Example:

```
<ul>
    <li>Header element</li>
    <!-- ko with: outboundFlight -->
        ...
    <!-- /ko -->
    <!-- ko with: inboundFlight -->
        ...
    <!-- /ko -->
</ul>
```

The `<!-- ko -->` and `<!-- /ko -->` comments act as start/end markers, defining a “virtual element” that contains the markup inside. Knockout understands this virtual element syntax and binds as if you had a real container element.

Dependencies

None, other than the core Knockout library.

(c) knockoutjs.com