This may be out of date. Read what's new here!

THE TOP 7 TIPS FOR MASTERING ANGULAR KENDO UI

Telerik Kendo UI

You've got Angular. You need a robust UI library with feature rich widgets like a Grid or TreeView. Angular Kendo UI is here.

The Angular integrations for Telerik Kendo UI® bring the full power of the Kendo UI comprehensive collection of open source and premium enterprise grade widgets to Angular developers everywhere. The integrations cover the open source Kendo UI Core, as well as Kendo UI Professional, including the Data Visualization and charting components. From DropDowns to Schedulers to Charts, the tight Angular integration in Kendo UI enables you to drop a few script and style files into your page and get instant and complete access to Kendo UI via directives.

Once you've got Angular Kendo UI in your project, there are a few subtle tricks and best practices you should know to get the most out of Kendo UI, without sacrificing any of the wonderful framework features Angular has to offer.

1: Use ObservableArrays or DataSources Instead of Arrays

Angular raises developer expectations for the UI: anything we place on the scope, we now expect to be bound to the UI. Changing either should update the other. By and large, this applies to Angular on its own because Angular tracks all the scope items and DOM bindings. Accordingly, it's natural to assume if a Kendo UI widget is bound to an array on the scope, modifying that array will modify the data in the widget. But there is a good reason this is not the case.

When an array on the scope is set as the k-data-source for a Kendo UI widget, the directives pass this array to the Kendo UI widget as is—just an array. Kendo UI doesn't watch every collection object for changes. It only watches special objects, specifically Observables, ObservableArrays and DataSources (which wrap observables). That means when you deal with a Kendo UI widget from the scope, you will want to use either an ObservableArray or a Kendo UI DataSource instead of just a "Plain Jane" array.

Here is an example. This grid does not update when a new item is added to the collection because the collection is just a plain JavaScript array.

View Example

Now we swap out the plain array for a Kendo UI ObservableArray. This array contains most of the native array methods, and Kendo UI watches this object constantly.

View Example

2: Don't Forget About kendoEvent

Kendo UI widgets fire events. These events usually have an event parameter accompaning them and will contain important information you will need. In the Angular integrations for Kendo UI, you have to pass this parameter explicitly on the directive as kendoEvent.

```
<select kendo-drop-down-list k-on-change="change()"></select>
<script>
  function HomeCtrl($scope) {
    // logs 'undefined'
    $scope.change = function(e) {
      console.log(e);
    }
  }
</script>
```

Instead, Angular Kendo UI requires you to explicitly pass in the kendoEvent object with the event binding.

```
<select kendo-drop-down-list k-on-change="change(kendoEvent)"></select>
<script>
  function HomeCtrl($scope) {
    // logs the kendo event object
    $scope.change = function(e) {
      console.log(e);
    }
  }
</script>
```

Remembering this subtly will save valuable time otherwise spent wondering why the event bindings aren't receiving any arguments.

3: Sometimes You Gotta $Apply

Since some widgets get super verbose in their configuration via attributes (particularly true with charts), we made it possible to provide the options configuration object via the scope using `k-options`.

This works great and allows you to keep separation of UI and configuration concerns. However, any event bindings you provide via configuration objects on the scope are not watched by Angular. This means you need to call `$apply` when making changes to the scope inside one of these events.

Here is an example where selecting the row and updating the `$scope.firstName` and `$scope.lastName` values does not update in the UI.

View Example

Since that change event binding is specified on the scope object, it's necessary `$apply` changes so the scope recognizes something has changed.

View Example

Note this is only necessary when using the configuration object. If you set the event as an attribute on the HTML element, Kendo UI will automatically trigger a call apply in the directive—saving you time and effort.

4: Use ng-model Instead of Value for Bi-Directional Binding

The Kendo UI Directives are primarily concerned with the **change** `event` and `value` **method** of each widget. While it's possible to set the value of a widget on initialization, to change it later on, you must use the `value()`` method of the widget instance.

In Angular, what we really want is Bi-directional binding on widgets with the scope. To accomplish this, instead of the `k-value` attribute, use `ng-model`. This is tied to the widgets value under the covers and provides the desired two-way binding.

View Example

It's always an option to call the value method on any widget instance at any time in a controller. However, Angular Kendo UI is actually doing this exact thing under the covers when ng-model is used.

5: Remember That Strings Are 'Strings'

Angular's parsing engine requires “strings” attributes to be quoted as such, otherwise the value is looked for as a scope property. This can be confusing at first since Kendo UI does NOT require such quoting in its declarative initialization.

This simple oversight can be problematic when working with the AutoComplete, ComboBox, DropDownlist or any other widget that must determine which field in the DataSource object contains the key, and which contains text. Take for example a simple AutoComplete bound to a DataSource which has objects. Notice the subtle difference when using Kendo UI Declarative initialization vs. the Angular Kendo UI integration.

```
<script>
// assuming we have data that looks like this...
app.people = new kendo.data.DataSource({
  data: [ { text: "Alabama", value: "AL" },
          { text: "Alaska", value: "AK" },
          { text: "Arizona", value: "AZ" },
          { text: "Arkansas", value: "AR" } ]
});
</script>

<!-- the autocomplete declaration looks like this (kendo ui declarative) -->
<input data-role="autocomplete" data-source="app.people" data-text-field="text"
data-value-field="value" />


<!-- but the Angular integrations require quotes around the dataTextField and
dataValueField attributes -->
<input kendo-auto-complete k-data-source="people" k-data-text-field="'text'" k-data-
value-field="'value'" />
```

However, this is how Angular works, and we thought it counter intuitive to auto-quote strings for you. It is admittedly confusing if you are unfamiliar with Angular attribute parsing. This is why Angular Kendo UI will log out to the console anytime it gets an attribute it can't find on the scope. That means you will get an error message akin to the following if you use a value that cannot be found on the scope.

```
kendoAutoComplete's kDataTextField attribute resolved to undefined. Maybe you meant
to use a string literal like: 'text'?
```

6: Leverage Widget References

There will come some point in your application when you need to get a reference to a Kendo UI widget. If you weren't using Angular, you would just select the element with jQuery and pull the widget reference off its data attribute.

```
<script>
  // get the grid widget reference
  $('#grid').data('kendoGrid');
  // OR
  $('#grid').getKendoGrid();
</script>
```

Of course, selecting items from an Angular controller with jQuery is expressly frowned upon and could result in the universe collapsing in on itself. That being the case, we decided to give you an alternate way of getting the widget reference. All you need to do is pass in a scope variable to the directive.

```
<div kendo-grid="grid" ...></div>

<script>
function HomeCtrl($scope) {
  $scope.refresh = function() {
    // scope.grid is the widget reference
    $scope.grid.refresh();
  }
}
</script>
```

7: Respect the Scope Hierarchy

Often, you will find yourself embedding widgets inside other widgets—or rather directives inside directives. A common practice is to put certain Kendo UI controls inside a Kendo UI Window, or widgets inside a TabsTrip, Splitter and so on. When doing this, you will most likely run into scope hierarchy issues if you don't prefix your scope bindings with a .. Remember, the scope is not a model, it's just where your model lives. That means you can step all over yourself if you aren't careful. This will result in null widget references and all manner of weirdness you may attribute back to Kendo UI, when it is really a scope issue.

The best way to understand this is to watch John Lindquist's explanation in this short video on "The Dot". You can also read this comprehensive Gist on "Understanding Scopes."

Enjoy!

Angular Kendo UI is designed to bring the complete power of Kendo UI to Angular, without forcing you to compromise or adopt a pattern that might contradict your own preferences. It's designed to be intuitive for those who have done jQuery-based widget initialization and coding. Angular is a fantastic JavaScript framework for structuring applications, and it deserves a great UI that just works. *Note: For more information on Angular Kendo UI and how it to use it, make sure you visit the* Angular Kendo UI Demos.