Porting the ToDo Listing Screen

Derik Whittaker

Twitter: @derikwhittaker



- Create our first Knockout View Model
 - Knockout vs XAML ViewModel
- Create our first HTML View
 - Compare layout in HTML vs XAML
- Fetch data via ajax from our Web API endpoint
 - Ajax vs RestClient
- Apply styles to an HTML element via Knockout
 - Knockout css binding vs StyleConverters
- Formatting data via Computed Observables
 - Knockout computed vs Formatters

- Creating our View Model
- Creating our HTML View
- Fetching data via Ajax
- Applying styles to an HTML element via Knockout
- Formatting data via Computed Observables

Implementing our View Model

- No need to implement INotifyPropertyChanged
- No need to raise Property Changed events on each property
- Properties must be Knockout Observables to be found
- Knockout Observables are treated as methods not as 'properties'

Our First View Model

Silverlight View Model

Typescript View Model

```
module ToDo.ViewModels {
    export class MainViewModel {
    }
}
```

No need to implement INotifyPropertyChanged

Implements

Our First Observable Property

Silverlight Property

Typescript View Model w/ Knockout js

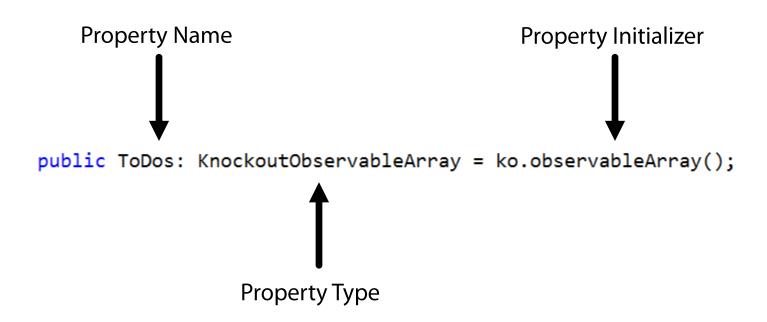
```
export class MainViewModel {
    public ToDos: KnockoutObservableArray = ko.observableArray();
}
```

Implements
Observable Array

Implements

No need to raise any Property Changed

Breaking down our Observable Property



Using an Observable Property

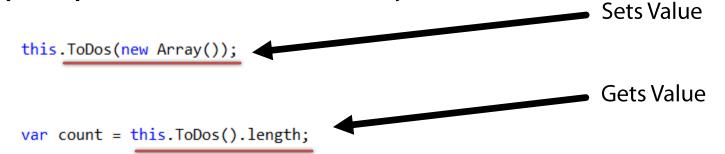
Silverlight Property

```
ToDoItems = new ObservableCollection<Models.ToDo>();

Gets Value

var count = ToDoItems.Count();
```

Typescript View Model w/ Knockout js



Notice we access these like methods NOT properties

- Creating our View Model
- Creating our HTML View
- Fetching data via Ajax
- Applying styles to an HTML element via Knockout
- Formatting data via Computed Observables

Implementing our View

Must declare EACH

XAML View

```
column/row up front
<Grid Height="20">
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="25" />
                                                                  Create UI Control for each
                                                                  data point.
        <ColumnDefinition Width="50" />
   </Grid.ColumnDefinitions>
                                                                  Assign row/column index
   <Ellipse Grid.Column="0" StrokeThickness="1" Height="10" Width="10"</pre>
        Style="{Binding Status.Description, Converter={StaticResource StateToEllipseCon
        HorizontalAlignment="Center" VerticalAlignment="Center" Margin="0,0,5,0" />
   <TextBlock Grid.Column="1" Text="{Binding Task}" Style="{StaticResource GridNormalTe</pre>
   <TextBlock Grid.Column="2" Text="{Binding DueDate, StringFormat='d'}" Style="{Static
    <TextBlock Grid.Column="3" Text="{Binding ReminderDate, StringFormat='d'}" Style="{S
   <TextBlock Grid.Column="4" Text="{Binding Category.Description}" Style="{StaticResou
   <TextBlock Grid.Column="5" Text="{Binding Priority.Description}" Style="{StaticResou
   <Button Grid.Column="6" Content="Edit" Command="{Binding ElementName=LayoutRoot, Pat</pre>
           CommandParameter="{Binding .}" Width="45" Ho.izontalAlignment="Right" />
   <Button Grid.Column="7" Content="Delete" Command="{Binding ElementName=LayoutRoot, P</pre>
            CommandParameter="{Binding .}" Width="45" HorizentalAlignment="Right" />
</Grid>
                                                                  Bind each element to
                                                                  the View Model
```

Implementing our View

Use a table and bind our

HTML View

```
collection to it via the
<div id="todoItems" class="container">
                                                 'foreach' control flow
  binding
      <div class="circle status-active-color" />
         <span data-bind="text: Task" ></span>
         <span data-bind="text: DisplayDueDate"></span>
         <span data-bind="text: DisplayReminderDate"></span>
         <span data-bind="text: Priority.Description"></span>
         <span data-bind="text: Category.Description"></span>
         <input type="button" class="btn" value="Edit" style="width: 65px;" />
         <input type="button" class="b { value="Delete" style="width: 65px;" />
      Bind to property in the
  view model
</div>
```

- Creating our View Model
- Creating our HTML View
- Fetching data via Ajax
- Applying styles to an HTML element via Knockout
- Formatting data via Computed Observables

Fetching Remote Data

Using RestClient in C# Web Api Route public void SchduledToDos(Action<IList<Models.ToDo>> callbackAction) var client = new RestClient("http://localhost:8888/ToDoServices/api/ToDo"); var request = new RestRequest(); Async request to get data client.ExecuteAsync(request, (response, handle) => (response.StatusCode == HttpStatusCode.OK) var results = JsonConvert .DeserializeObject<IList<Models.ToDo>>(response.Content); DispatcherHelper.CheckBeginInvokeOnUI(() => callbackAction.Invoke(results); Response callback }); Pushing to UI thread

Fetching Remote Data

Using jQuery

```
Web Api Route
var url = "http://localhost:8888/ToDoServices/api/ToDo/";
                                                               Ajax 'get' to the web api
$.get(url)
                                                               route
    .done((data) => {
        var temp = self.ToDos();
        _.each(data, (item) => {
           var toDoVM = ko.mapping.fromJS(item, {}, new ToDoListItemViewModel());
           temp.push(toDoVM);
        });
        self.ToDos.valueHasMutated();
                                                               Ajax call back to handle
    });
                                                               processing
```

- Creating our View Model
- Creating our HTML View
- Fetching data via Ajax
- Applying styles to an HTML element via Knockout
- Formatting data via Computed Observables

Changing Styles on the Fly

```
Custom Class implements
Silverlight Value Converter
                                                                      IValueConverter
   public class StateToEllipseConverter : IValueConverter
      public object Convert(object value, Type targetType, object parameter, CultureInfo culture)
          var status = value != null ? value.ToStr
                                                                       Value to 'test' against
          return GetStyle(string.Format("{0}StatusElipseStyle", statu
                                                                       provided
      }
      public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)
      public virtual object GetStyle(string styleToUse)...
                                                                      Bound property to 'test'
XAML Implementation
                                                                      against
   <Ellipse
        Style="{Binding Status.Description,
        Converter={StaticResource StateToEllipseConverter}}"
                                                                      Invoking the
         />
                                                                      IValueConverter
```

Changing Styles on the Fly

Knockout Computed Method

```
this.StatusStyle = ko.computed(() => {
    return "circle status-" + this.Status().toLowerCase() + "-color"
});
    Triggers off another
    Observable
```

HTML Implementation

<div data-bind="attr: {class: StatusStyle}" />

Uses the attr binding key

Bound to the Observable

Computed Observable

- Creating our View Model
- Creating our HTML View
- Fetching data via Ajax
- Applying styles to an HTML element via Knockout
- Formatting data via Computed Observables

Formatting Data

Silverlight StringFormat

Can use the built in StringFormat on bound field

Knockout Formatting

Can bind to an observable property

```
public DisplayDueDate: KnockoutComputed;
this.DisplayDueDate = ko.computed(() => {
   var displayDate = "";
   if (this.DueDate() != undefined && this.DueDate() != "") {
      displayDate = moment(this.DueDate()).format('L');
   }
   return displayDate;
});
```

Create a Computed Field to do the formatting

Could also use Custom Bindings in Knockout.

Summery

- Created our View Model
- Created our HTML View
- Learned how to use jQuery and Ajax to fetch remote data
- Learned how to apply styles via Knockout
- Learned how to apply formatting via Knockout