Porting the ToDo Listing Screen

Derik Whittaker
Twitter: @derikwhittaker

pluralsight
hardcore developer training

Agenda

- **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

Agenda

- 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

```
namespace ToDo.Xaml.ViewModels
{
    public class MainViewModel : GalaSoft.MvvmLight.ViewModelBase
    {

    }
}
```

Implements INotifyPropertyChanged

Typescript View Model

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

    }
}
```

No need to implement INotifyPropertyChanged

Our First Observable Property

Silverlight Property

```
public class MainViewModel : GalaSoft.MvvmLight.ViewModelBase
{
    private ObservableCollection<Models.ToDo>
                _toDoItems = new ObservableCollection<Models.ToDo>();

    public ObservableCollection<Models.ToDo> ToDoItems
    {
        get { return _toDoItems; }
        set { _toDoItems = value; RaisePropertyChanged(() => ToDoItems); }
    }
}
```

Implements Observable Collection

Must manually raise Property Changed

Typescript View Model w/ Knockout js

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

Implements Observable Array

No need to raise any Property Changed

Breaking down our Observable Property

Property Name

Property Initializer

```
public ToDos: KnockoutObservableArray = ko.observableArray();
```

Property Type

Using an Observable Property

Silverlight Property

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

Sets Value

Gets Value

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

Typescript View Model w/ Knockout js

Sets Value

```
this.ToDos(new Array());
```

Gets Value

```
var count = this.ToDos().length;
```

Notice we access these like methods NOT properties

Agenda

- 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

- **XAML View**

```
<Grid Height="20">
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="25" />
        ...
        <ColumnDefinition Width="50" />
    </Grid.ColumnDefinitions>

    <Ellipse Grid.Column="0" StrokeThickness="1" Height="10" Width="10"
         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
    <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
            CommandParameter="{Binding .}" Width="45" HorizontalAlignment="Right" />
    <Button Grid.Column="7" Content="Delete" Command="{Binding ElementName=LayoutRoot, P
            CommandParameter="{Binding .}" Width="45" HorizontalAlignment="Right" />
</Grid>
```

Must declare EACH column/row up front

Create UI Control for each data point.

Assign row/column index

Bind each element to the View Model

Implementing our View

- **HTML View**

```
<div id="todoItems" class="container">
    <table class="table table-hover">
        <tbody data-bind="foreach: ToDos">
            <tr>
                <td style="width: 15px; vertical-align: middle; padding-right: 0px;">
                    <div class="circle status-active-color" />
                </td>
                <td style="width: 300px;"><span data-bind="text: Task" ></span></td>
                <td style="width: 75px;"><span data-bind="text: DisplayDueDate"></span></td>
                <td style="width: 75px;"><span data-bind="text: DisplayReminderDate"></span></td>
                <td style="width: 75px;"><span data-bind="text: Priority.Description"></span></td>
                <td style="width: 75px;"><span data-bind="text: Category.Description"></span></td>
                <td style="width: 75px;"><input type="button" class="btn" value="Edit" style="width: 65px;" /></td>
                <td style="width: 75px;"><input type="button" class="b[illegible]" value="Delete" style="width: 65px;" /></td>
            </tr>
        </tbody>

    </table>
</div>
```

Use a table and bind our collection to it via the 'foreach' control flow binding

Bind to property in the view model

Agenda

- 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#**

```
public void SchduledToDos( Action<IList<Models.ToDo>> callbackAction )
{
    var client = new RestClient("http://localhost:8888/ToDoServices/api/ToDo");
    var request = new RestRequest();

    client.ExecuteAsync(request, (response, handle) =>
    {
        if (response.StatusCode == HttpStatusCode.OK)
        {
            var results = JsonConvert
                .DeserializeObject<IList<Models.ToDo>>(response.Content);

            DispatcherHelper.CheckBeginInvokeOnUI(() =>
            {
                callbackAction.Invoke(results);
            });
        }
    });
}
```

Web Api Route

Async request to get data

Response callback
Pushing to UI thread

Fetching Remote Data

- **Using jQuery**

```
var url = "http://localhost:8888/ToDoServices/api/ToDo/";

$.get(url)
    .done((data) => {
        var temp = self.ToDos();

        _.each(data, (item) => {
            var toDoVM = ko.mapping.fromJS(item, {}, new ToDoListItemViewModel());

            temp.push(toDoVM);
        });

        self.ToDos.valueHasMutated();
    });
```

Web Api Route

Ajax 'get' to the web api route

Ajax call back to handle processing

Agenda

- 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

Silverlight Value Converter

Custom Class implements IValueConverter

```
public class StateToEllipseConverter : IValueConverter
{
    public object Convert(object value, Type targetType, object parameter, CultureInfo culture)
    {
        var status = value != null ? value.ToString() : "";

        return GetStyle(string.Format("{0}StatusElipseStyle", status));
    }

    public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)

    public virtual object GetStyle(string styleToUse)...
}
```

Value to 'test' against provided

XAML Implementation

Bound property to 'test' against

```
<Ellipse
     Style="{Binding Status.Description,
     Converter={StaticResource StateToEllipseConverter}}"
      />
```

Invoking the IValueConverter

Changing Styles on the Fly

Knockout Computed Method

Computed Observable

```
this.StatusStyle = ko.computed(() => {

    return "circle status-" + this.Status().toLowerCase() + "-color"

});
```

Triggers off another Observable

HTML Implementation

Bound to the Observable

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

Uses the attr binding key

Agenda

- 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**

```
<TextBlock Grid.Column="2"
           Text="{Binding DueDate, StringFormat='d'}"
           />
```

Can use the built in StringFormat on bound field

- **Knockout Formatting**

```
<span data-bind="text: DisplayDueDate"></span>
```

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