MVVM

## Gitting the Source Code

If you have a git client installed, then clone the repository. Create a working branch. For example:

cd c:\projects

git clone https://github.com/latish/win8.mvvm.git

cd win8.mvvm

git checkout –b take1

After each successful burst, commit your changes. If you ever get lost, commit that branch, go back to master, and create a new one:

git add –A

git commit –m "I missed that."

git checkout master

git pull

git checkout –b take2

## Downloading the Source Code

If you don’t have git installed, then go to the following URL and click on the “ZIP” button:

https://github.com/latish/win8.mvvm

Unzip to your project folder. If you ever get lost, go back to the web page and download the zip again.

# MVVM

The MVVM pattern is a variation of another well-known separation pattern named Model-View-Controller, or MVC. The main advantage of a separation pattern is that it assigns clearly defined responsibilities to each of the layers. This enables different team members to work simultaneously on multiple layers and makes it easier to test. MVVM in particular takes advantage of the declarative data-binding available in XAML allowing a designer to work on the UI in parallel to the developer.

We’ll be using MvvmLight to reduce the amount of boilerplate code that you have to write. Install [MvvmLight Toolkit](https://mvvmlight.codeplex.com/releases) to get the Visual Studio Snippets. The sample code already has references to the needed libraries. In this lab, the view already has some default markup. We’ll start with the Model, and then work on the ViewModel.

The Model represents our domain data that we would retrieve from a data store. Open Model/ScheduleItem.cs and set base class as GalaSoft.MvvmLight.ObservableObject. Using snippet mvvminpcset, add the following observable properties. Set the initial values of the backing fields as "" for strings and DateTime.MinValue for the date.

Title (string), DateTime (DateTime), SpeakerFirstName (string), SpeakerLastName (string)

For example, Title will look like

public const string TitlePropertyName = "Title";

private string \_title = "";

public string Title { get {return \_title;} set {Set(TitlePropertyName, ref \_title, value);} }

We need a backing store to get the data. Open Services/DataService.cs and uncomment the hardcoded data in GetHardCodedData().

Now let’s move to the ViewModel, which can have more view specific properties in it (like a formatted full name or time). Open ViewModel/ScheduleItemViewModel.cs and derive class from ViewModelBase. Add the following code:

public ScheduleItem Item { get; private set; }

public ScheduleItemViewModel(ScheduleItem item)

{

Item = item;

Item.PropertyChanged += Item\_PropertyChanged;

}

void Item\_PropertyChanged(object sender, System.ComponentModel.PropertyChangedEventArgs e)

{

if (e.PropertyName == ScheduleItem.SpeakerFirstNamePropertyName ||

e.PropertyName == ScheduleItem.SpeakerLastNamePropertyName)

{

RaisePropertyChanged(()=>SpeakerName);

return;

}

if (e.PropertyName == ScheduleItem.DateTimePropertyName)

{

RaisePropertyChanged(()=>Time);

return;

}

}

public string SpeakerName

{

get { return string.Format("{0} {1}", Item.SpeakerFirstName, Item.SpeakerLastName); }

}

public string Time

{

get { return Item.DateTime.ToString("hh:mm"); }

}

Last step, update the MainViewModel class to tie everything together. Add an ObservableCollection of ScheduleItemViewModels to expose the list of items and initialize it in the constructor.

public ObservableCollection<ScheduleItemViewModel> Items { get; private set; }

//Put below statement in constructor

Items = new ObservableCollection<ScheduleItemViewModel>();

Add a RelayCommand so that the button on the View can refresh the list of schedule items.

private RelayCommand \_refreshCommand;

public RelayCommand RefreshCommand

{

get

{

return \_refreshCommand

?? (\_refreshCommand = new RelayCommand(ExecuteRefreshCommand));

}

}

private void ExecuteRefreshCommand()

{

var items = \_dataService.GetSchedule();

if (items != null)

{

Items.Clear();

foreach (var scheduleItem in items)

Items.Add(new ScheduleItemViewModel(scheduleItem));

}

}