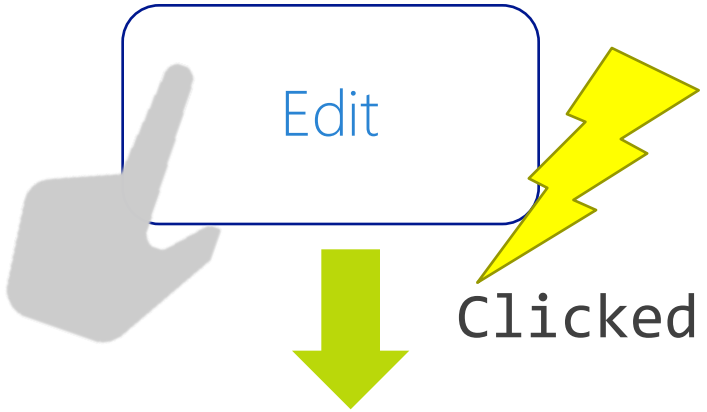
Xamarin.Forms: Commanding Events

James Montemagno
@JamesMontemagno

Event Handling

- UI raises events to notify code about user activity
  - `Clicked`
  - `ItemSelected`
  - …
- The downside is that these events must be handled in the code behind file



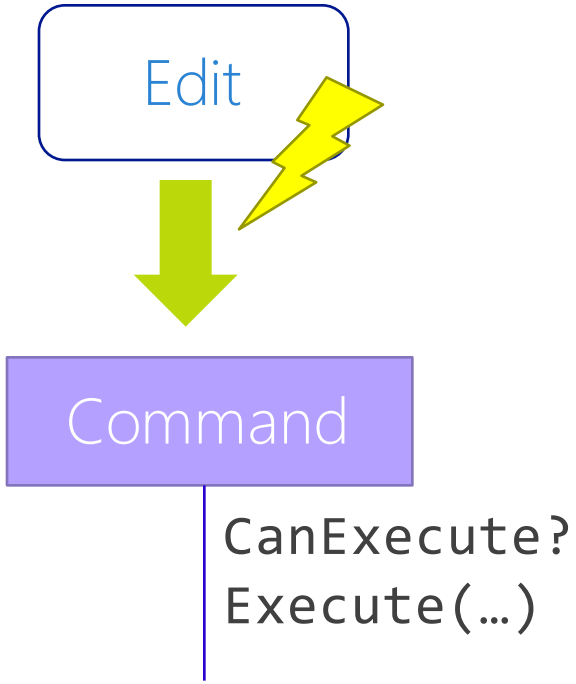

```
public MainPage()
{
    ...
    Button editButton = ...;
    editButton.Clicked += OnClick;
}

void OnClick (object sender, EventArgs e)
{
     ...
}
```

Commands

- Microsoft defined the **`ICommand`** interface to provide a commanding abstraction for their XAML frameworks

```
public interface ICommand
{
    event EventHandler CanExecuteChanged;
    bool CanExecute(object parameter);
    void Execute(object parameter);
}
```

Can provide an optional parameter (often `null`) for the command to work with for context



ICommand

- **`ICommand`** has three required members you must implement

`CanExecute` is called to determine whether the command is valid, this can enable / disable the control which is bound to the command

```
public interface ICommand
{
    bool CanExecute(object parameter);
    void Execute(object parameter);
    event EventHandler CanExecuteChanged;
}
```

ICommand

- **`ICommand`** has three required members you must implement

`Execute` is called to actually run the logic associated with the command when the control is activated – it will only be called if `CanExecute` returned `true`

```
public interface ICommand
{
    bool CanExecute(object parameter);
    void Execute(object parameter);
    event EventHandler CanExecuteChanged;
}
```

ICommand

- **`ICommand`** has three required members you must implement

`CanExecuteChanged` is an event which the binding will subscribe to, the ViewModel should raise this event when the validity of the command changes

```
public interface ICommand
{
    bool CanExecute(object parameter);
    void Execute(object parameter);
    event EventHandler CanExecuteChanged;
}
```

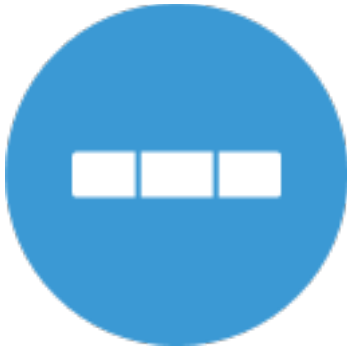
The binding will then call **`CanExecute`** and enable / disable the UI in response

Commands in Xamarin.Forms

- A few Xamarin.Forms controls expose a **`Command`** property for the main action of a control

`Menu`

`Button`



`ToolbarItem`

`TextCell`

Commands in Xamarin.Forms

- A few Xamarin.Forms controls expose a **Command** property for the main action of a control

```
public ICommand GiveBonus { get; }
```

```
<Button Text="Give Bonus"
        Command="{Binding GiveBonus}" />
```

Can data bind a property of type **ICommand** to the **Command** property

Gesture-based commands

- Xamarin.Forms also includes a **`TapGestureRecognizer`** which can provide a command interaction for other controls or visuals

```
<Image Source="IDareYouToTapMe.jpg">
    <Image.GestureRecognizers>
        <TapGestureRecognizer
            Command="{Binding BeBraveCommand}"
            CommandParameter="TheyTookTheDare!" />
    </Image.GestureRecognizers>
</Image>
```

`CommandParameter` property supplies the command's parameter – in this case as a **`string`**

Using delegate commands

- **Command<T>** and **Command** provides mechanism to centralize the logic for the commands into the VM

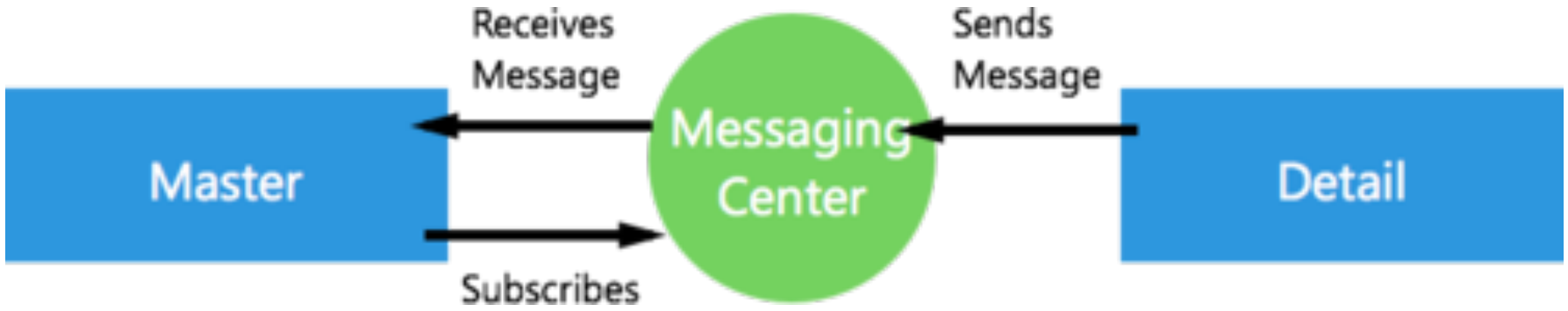
```
public class EmployeeViewModel : INotifyPropertyChanged
{
    public ICommand GiveBonus { get; private set; }
    public EmployeeViewModel(Employee model) {
        GiveBonus = new Command(OnGiveBonus, OnCanGiveBonus);
    }

    void OnGiveBonus() { ... }
    bool OnCanGiveBonus() { return ... }
}
```

Let’s do it!

Passing Messages to the UI

James Montemagno
@JamesMontemagno

Messaging Center



- MessagingCenter.Subscribe<T>(object subscriber, string message, Action<T> callback);
- MessagingCenter.Send(T item, string message);

Let’s handle exceptions

Messaging Center

- Master Page:

```
//Subscibe to insert expenses
MessagingCenter.Subscribe<TripExpense>(this, "AddNew", (expense) =>
{
  Expenses.Add(expense);
});
```

- Detail Page:

```
MessagingCenter.Send(expense, "AddNew");
```

Xamarin.Forms: Pull to Refresh

James Montemagno
@JamesMontemagno

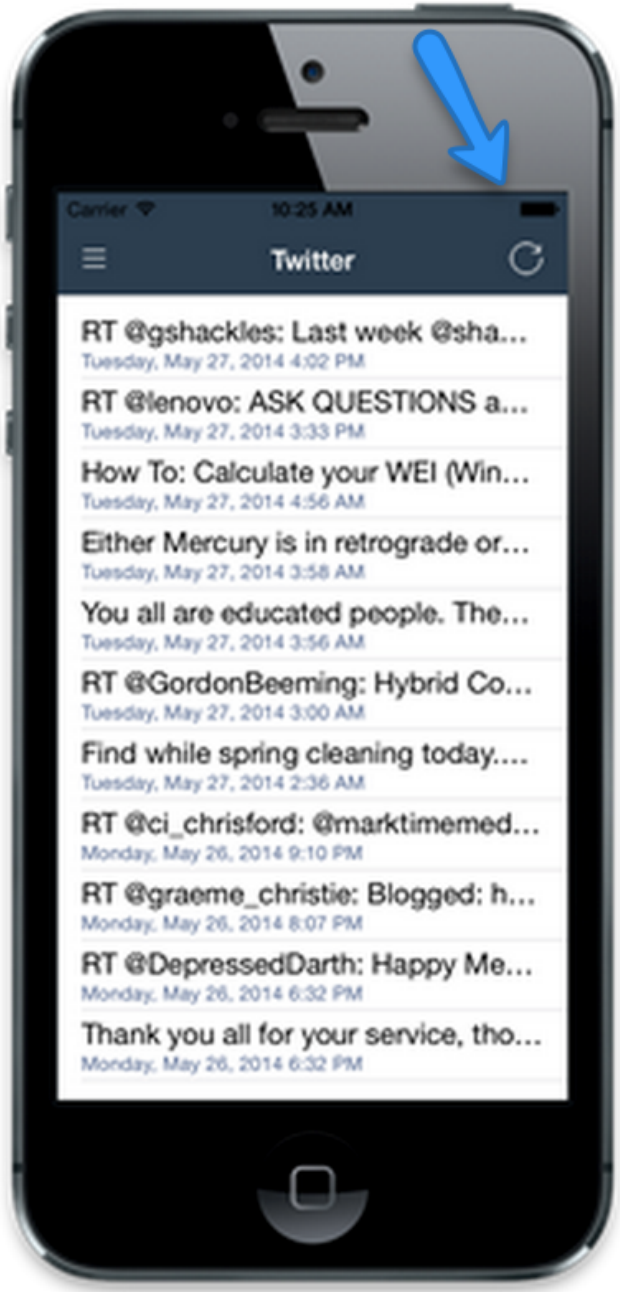
```
public class MyViewModel : INotifyPropertyChanged
{
    bool isBusy;
    public bool IsBusy
    {
        get { return isBusy; }
        set
        {
            if (isBusy == value)
                return;

            isBusy = value;
            OnPropertyChanged ("IsBusy");
        }
    }


    public event PropertyChangedEventHandler PropertyChanged;

    public void OnPropertyChanged(string propertyName)
    {
        if (PropertyChanged == null)
            return;

        PropertyChanged (this, new PropertyChangedEventArgs (propertyName));
    }
}
```

```
ICommand refreshCommand;
public ICommand RefreshCommand
{
    get
    { return refreshCommand ??
                  (refreshCommand =
                  new Command (async ()=> await ExecuteRefreshCommand()));
    }
}

async Task ExecuteRefreshCommand()
{
    if (IsBusy)
        return;

    IsBusy = true;
    //do stuff
    IsBusy = false;
}
```

A long time ago…

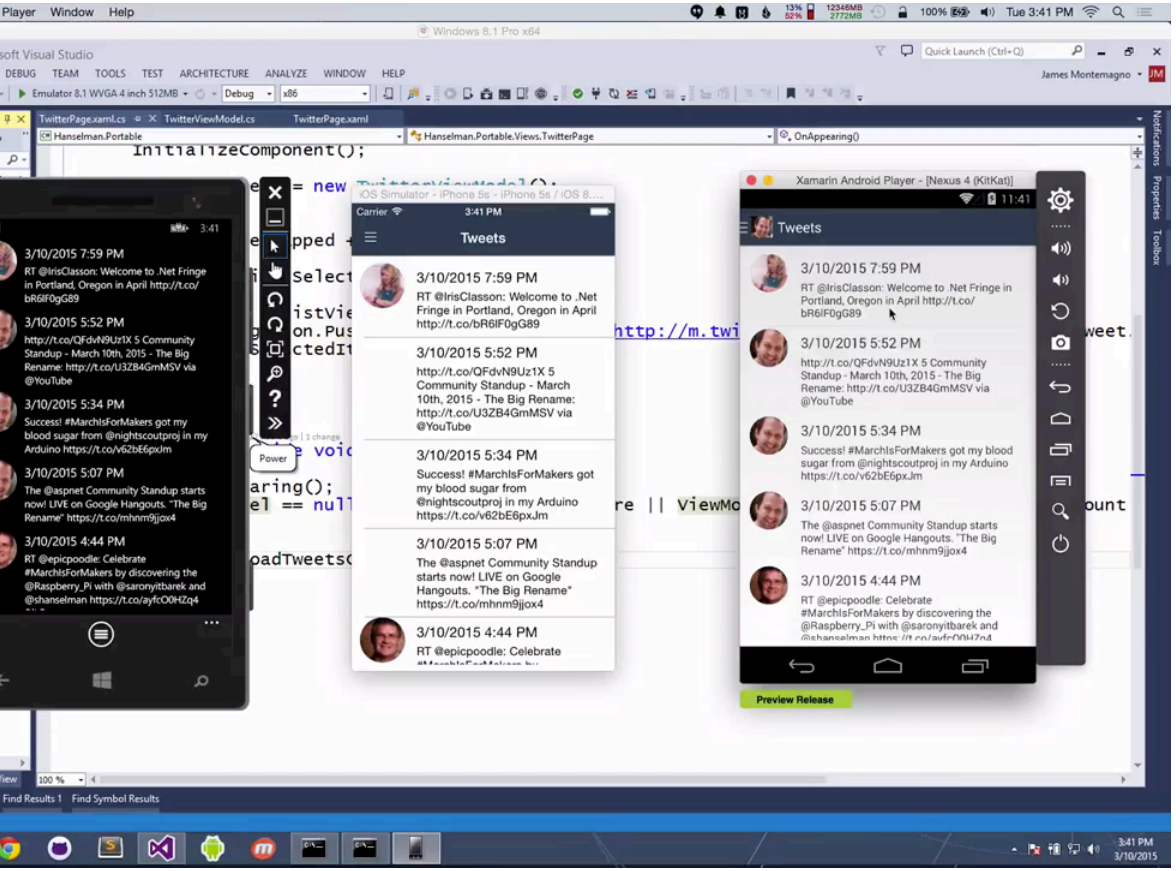
No pull to refresh ☹

```
var refresh = new ToolbarItem
                {
                    Icon = "refresh.png",
                    Text = "Refresh",
                    Command = viewModel.RefreshCommand
                };

ToolbarItems.Add(refresh);
```



Game Changer:
Pull to Refresh in Xamarin.Forms!

Pull to Refresh



- Built in ListView
- Uses Native Controls
- Supports iOS, Android, and Windows Phone!

The API

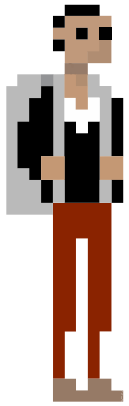
```
//Enable/Disable all pull to refresh
public bool IsPullToRefreshEnabled { get; set; } = false;

//Is the spinner currently shown
public bool IsRefreshing { get; set; } = false;

//The method/command to trigger when pulled
public ICommand RefreshCommand { get; set; } = null;

//Manual Events to trigger/subscribe
public void BeginRefresh ();
public void EndRefresh ();
public event EventHandler Refreshing;
```

The XAML

```
<ListView x:Name="listView"
          ItemsSource="{Binding Items}"
          HasUnevenRows="True"
          IsPullToRefreshEnabled="True"
          RefreshCommand="{Binding RefreshCommand}"
          IsRefreshing="{Binding IsBusy, Mode=OneWay}">
```

Let’s pull to refresh!

20 Minute Break



James
Montemagno
Developer Evangelist, Xamarin

james@xamarin.com motzcod.es @JamesMontemagno