The Extras

Laurent Bugnion
@LBugnion
http://www.galasoft.ch

Outline

- **Binding Views and ViewModels, strategies**
  - Using the ViewModelLocator
  - In code behind
- **Why do we have extras?**
- **Managing dependencies with SimpleIoc**
- **Handling events and commands with EventToCommand**
- **Summary**

Setting the DataContext

pluralsight
hardcore developer training

DataContext in XAML

- Instead of writing:

```
<TextBox
 Text="{Binding SearchQuery,
                Source={StaticResource Main},
                Mode=TwoWay}" />

<Button
 Content="{Binding SearchText,
                   Source={StaticResource Strings}}"
 Command="{Binding FindCommand,
                   Source={StaticResource Main}}" />

<ItemsControl
 ItemsSource="{Binding Results,
                       Source={StaticResource Main}}" />
```

DataContext in XAML

- We can write:

```
<Page.DataContext>
    <Binding Source="{StaticResource Main}" />
</Page.DataContext>

<TextBox
 Text="{Binding SearchQuery, Mode=TwoWay}" />

<Button
 Content="{Binding SearchText,
                   Source={StaticResource Strings}}"
 Command="{Binding FindCommand}" />

<ItemsControl
 ItemsSource="{Binding Results}" />
```

The DataContext

- This is a shortcut.
- In XAML, works great with designers (Blend, Visual Studio)
- In code, sometimes more convenient (for instance Detail view in Master-Detail apps)
- For instance in Windows 8:

```
protected override void OnNavigatedTo(NavigationEventArgs e)
{
    DataContext = e.Parameter as DetailsViewModel;
    base.OnNavigatedTo(e);
}
```

The Design Time DataContext

- When the DataContext is set in code behind, Blend does not run that code.

- Use d:DataContext in XAML

```
<Page x:Class="App1.MainPage“
      xmlns=http://schemas.microsoft.com/winfx/2006/xaml/presentation
      xmlns:x=http://schemas.microsoft.com/winfx/2006/xaml
      xmlns:d=http://schemas.microsoft.com/expression/blend/2008
      xmlns:mc=http://schemas.openxmlformats.org/markup-compatibility/2006
      xmlns:app1="using:App1"
      mc:Ignorable="d"
      d:DataContext="{d:DesignInstance,
                      Type=app1:DetailsViewModel,
                      IsDesignTimeCreatable=True}">
```

The ViewModelLocator

- This is just an additional level of control
- Defined as a global resource → known to Blend
- Exposes the MainViewModel as public property

```
public class ViewModelLocator
{
    public MainViewModel Main
    {
        get;
        private set;
    }

    // ...
}
```

The ViewModelLocator

- Syntax in App.xaml:

```
<Application.Resources>
    <vm:ViewModelLocator x:Key="Locator" />
</Application.Resources>
```

- Syntax in the view:

```
DataContext="{Binding Main,
                      Source={StaticResource Locator}}">
```

Why Do We Have Extras?

- **Extras require an external reference.**
  - SimpleIoc and Microsoft.Practices.ServiceLocation
  - EventToCommand and System.Windows.Interactivity

- **Some users may have issues with it.**
  - Licenses
  - Procurement

Managing Dependencies with Simpleloc

pluralsight
hardcore developer training

What is Dependency Injection?

- Instead of doing:

```
public class MainViewModel
{
    private IDataService _dataService;
    public MainViewModel()
    {
        _dataService = new DataService();
    }
}
```

- We do:

```
public class MainViewModel
{
    private IDataService _dataService;
    public MainViewModel(IDataService dataService)
    {
        _dataService = dataService;
    }
}
```

What is Dependency Injection?

- Delegates the creation of services to another class.
- Injects the dependency inside the consumer.
- The consumer doesn't have to decide which implementation to use.

```
public class Startup
{
    // ...
    public void Start()
    {
        IDataService service;
        if (condition)
        {
            service = new DataService();
        }
        else
        {
            service = new AnotherService();
        }
        var viewModel = new MainViewModel(service);
    }
}
```

Using an IOC Container

- **IOC == Inversion Of Control**
- **An IOC container is:**
  - Responsible to create services when needed
  - Responsible for injecting them
  - Responsible for caching the objects
  - and providing access to them
- **Multiple IOC containers on the market**
  - Unity (Microsoft)
  - Ninject
  - StructureMap
  - CastleWindsor
  - And more…

Using the Simpleloc

- **Why yet another IOC container?**
- **Very well suited to MVVM apps**
  - Works in Blend!
- **Many developers have never used an IOC container**
  - And many have a favorite one
  - Which one to select?
- **Why not provide a very simple one?**
  - Not many features
  - But a good “gateway drug” to IOC and DI
- **Hence Simpleloc was born**

Registering to SimpleIoc

- Default registration:

```
SimpleIoc.Default.Register<MainViewModel>();
```

- Registration with interface:

```
SimpleIoc.Default.Register<IDataService, DataService>();
```

- Conditional registration:

```
if (condition)
{
    SimpleIoc.Default.Register<IDataService, DataService>();
}
else
{
    SimpleIoc.Default.Register<IDataService, AnotherService>();
}
```

Registering to Simpleloc

- Creation is on demand!
- Objects are cached
- Must explicitly unregister to remove an object from cache

Registering to SimpleIoc (Factory)

- Using pre-created objects:

```
var myService = new DataService();
SimpleIoc.Default.Register<IDataService>(() => myService);
```

- Passing parameters to a constructor

```
var param1 = true;
var param2 = "Hello world";
SimpleIoc.Default.Register<MainViewModel>(
    () => new MainViewModel(param1, param2));
```

- Factory execution once, on demand!
- Result of factory execution is cached

Registering to SimpleIoc (Options)

- Create the instance at registration

- Register with a key (for multiple instances)

```
SimpleIoc.Default.Register<MainViewModel>(
    () => new MainViewModel(), "MyUniqueKey");

SimpleIoc.Default.Register<IDataService>(
    () => new DataService(), "AnotherUniqueKey");
```

Getting an Instance

- Getting the default instance:

```
var instance
    = SimpleIoc.Default.GetInstance<MainViewModel>();
var service
    = SimpleIoc.Default.GetInstance<IDataService>();
```

- Getting a keyed instance

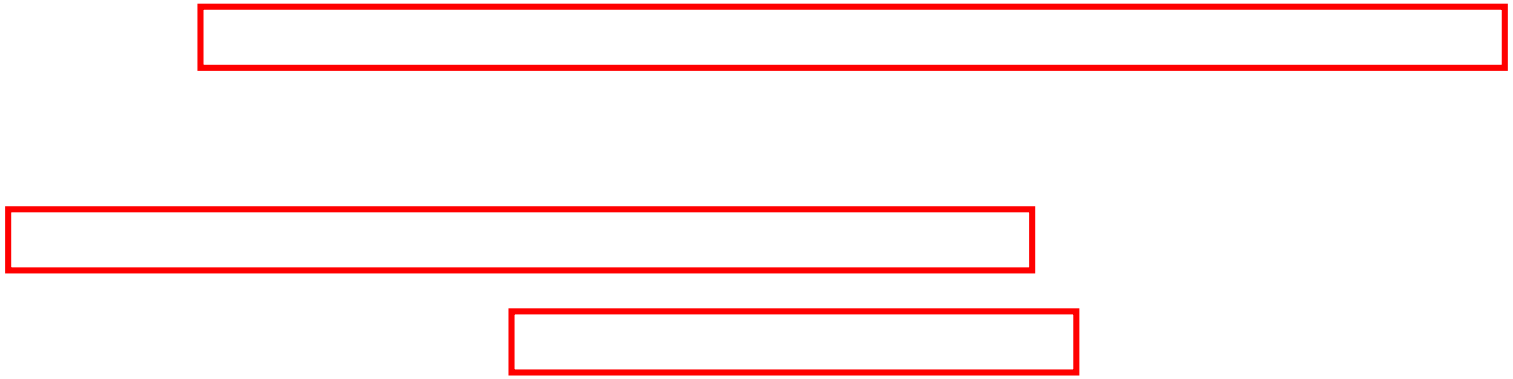
Composing Dependencies

- SimpleIoc can handle dependencies between objects

Constructor Injection Vs Property Injection

```
public class MainViewModel : ViewModelBase
{

}
```



Unregistering an Instance

- Unregistering removes instances from the cache.
- This call unregisters the class/interface completely:

```
SimpleIoc.Default.Unregister<MainViewModel>();
SimpleIoc.Default.Unregister<IDialogService>();
```

- This call unregisters instances but NOT the interface:

```
SimpleIoc.Default.Unregister<IDialogService>(this);
SimpleIoc.Default.Unregister<IDataService>("key1");
```

A Typical Scenario

- **When navigating to a page**
  - Registering the page to display user dialogs
- **When navigating away**
  - Unregistering the page so that others can register

```
public sealed partial class MainPage : IDialogService
{
    // ...

    protected override void OnNavigatedTo(NavigationEventArgs e)
    {
        SimpleIoc.Default.Register<IDialogService>(() => this);
        base.OnNavigatedTo(e);
    }

    protected override void OnNavigatedFrom(NavigationEventArgs e)
    {
        SimpleIoc.Default.Unregister(this);
        base.OnNavigatedFrom(e);
    }
}
```

Utility Methods:
IsRegistered Vs ContainsCreated

- Checking if a class or interface has been registered.
- Checking if a class has at least one instance created.

Utility Methods: GetAllCreatedInstances

- **Getting all the instances that have been already created.**
  - This does NOT force the creation of instances!
  - Only already existing instances are returned.

Utility Methods: GetAllInstances

- **Getting all the instances.**
  - This DOES force the creation of one default instance per registered class/interface!

```
SimpleIoc.Default.Register<IDataService, DataService>();
var allInstances
    = SimpleIoc.Default.GetAllInstances<IDataService>();
    // One instance
```

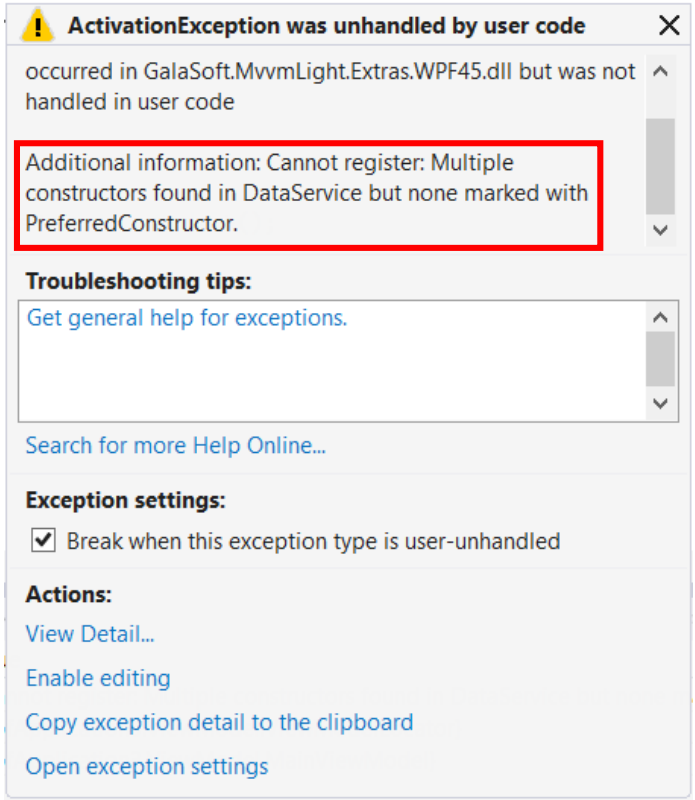
Utility Attribute: PreferredConstructor

- **Which constructor will SimpleIoc use?**
  - By default, uses the default constructor (doh…)
  - If multiple constructors are found, ActivationException is thrown

- **In order to be more precise, use the PreferredConstructor attribute**

```
public class DataService
{
    public DataService()
    {
    }

    [PreferredConstructor]
    public DataService(
      IAnotherService another)
    {
    }
}
```



What is Microsoft.Practices.ServiceLocation?

- Contains the ServiceLocator class.
- An agreement between most IOC containers.
- Allows easy swapping from one IOC container to another.

- Defining the ServiceLocator:

```
ServiceLocator.SetLocatorProvider(() => SimpleIoc.Default);
```

- Using the ServiceLocator:

```
ServiceLocator.Current.GetInstance<IDataService>();
```

- Exact equivalent to:

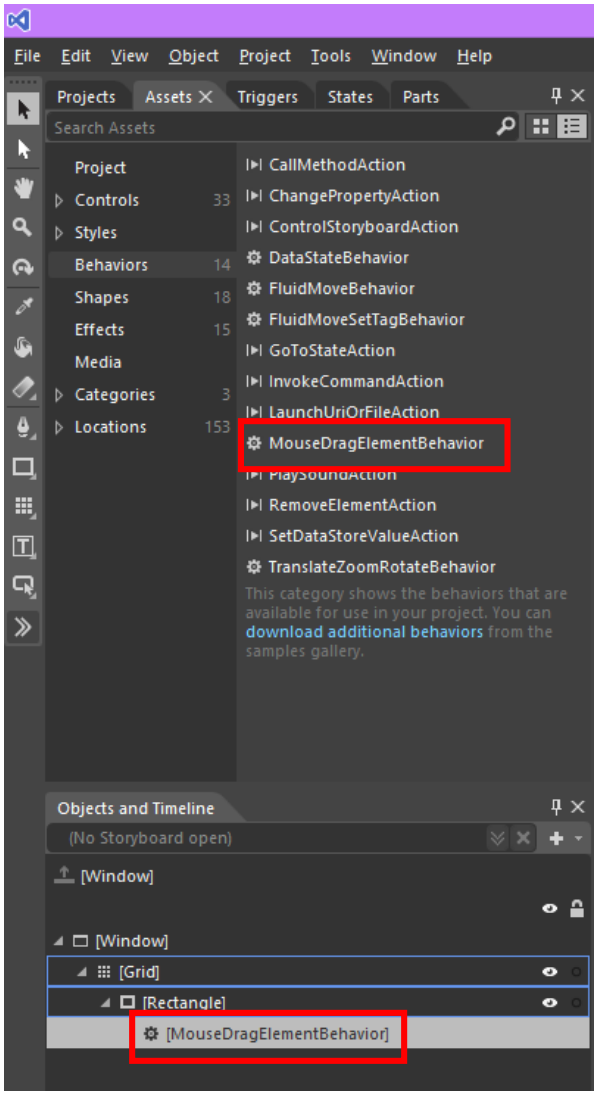
```
SimpleIoc.Default.GetInstance<IDataService>();
```

Handling Events and Commands with EventToCommand

pluralsight
hardcore developer training

What are Behaviors?

- **Based on Attached Behaviors**
  - developed by the WPF community
  - http://galasoft.ch/s/attbehaviors
- **Initially developed by the Blend team**
  - But no dependency on Blend
- **Small pieces of "packed" code behind**
- **Easily redistributable**
- **Attachable to a UI element**
- **Very optimized for Blend**



How to Add a Behavior?

- In Blend, simply drag from the Assets library
- Configure in Properties panel

- In XAML:

```
<Rectangle Fill="Blue“
           Height="100“
           Width="150">
    <i:Interaction.Behaviors>
        <ei:MouseDragElementBehavior />
    </i:Interaction.Behaviors>
</Rectangle>
```

- With:

```
xmlns:i="http://schemas.microsoft.com/expression/2010/interactivity"
xmlns:ei="http://schemas.microsoft.com/expression/2010/interactions"
```

Differences Between Behaviors and Actions

- **A behavior is standalone**
  - For instance MouseDragElementBehavior
- **An action is always attached to a Trigger**
  - Think of an Action as actuator
  - And Trigger as sensor
- **When triggered, the action can execute its task**
- **Triggers can be of multiple kinds**
  - Event, Data, etc

The EventToCommand Action

- EventToCommand is not a very good name
- In fact it is an action
  - So it can be attached to *any* trigger

- When the trigger is actuated, executes a Command

- Syntax in XAML:

```
<Rectangle Fill="Blue“
           Height="115“
           VerticalAlignment="Top">
    <i:Interaction.Triggers>
        <i:EventTrigger EventName="MouseLeftButtonDown">
            <Custom:EventToCommand Command="{Binding MyCommand}“
                                   CommandParameter="a parameter" />
        </i:EventTrigger>
    </i:Interaction.Triggers>
</Rectangle>
```

The EventToCommand, Scenarios

- **Useful whenever an element doesn't have a Command property**
  - Anything else than ButtonBase (Button, ToggleButton, RadioButton, etc)
- **Useful when another event than "Click" must be used**
  - Also for Button
- **Useful with other trigger types**
  - For instance DataTrigger
- **Useful whenever event handlers are not applicable**
  - For example DataTemplate, etc

EventToCommand Properties

- **Command**
  - Can be databound.
  - Any ICommand target
- **CommandParameter**
  - Can be databound
  - Any object target

EventToCommand Properties

- **PassEventArgsToCommand**
  - If True, and the trigger is EventTrigger
    → EventArgs will be passed to the ICommand
- **EventArgsConverter**
  - A resource implementing IEventArgsConverter
- **EventArgsConverterParameter**
  - Used to pass additional information to the EventArgsConverter

EventToCommand Vs InvokeCommandAction

- **In Blend SDK: InvokeCommandAction**
  - Same intent
  - No PassEventArgsToCommand
- **In Windows 8.1:**
  - InvokeCommandAction + InputConverter + InputConverterParameter
  - Note: InputConverter is an IValueConverter
- **Windows 8.0:**
  - No behaviors
  - See www.galasoft.ch/s/msdncommand

In Windows 8.1

- In XAML:

```
<TextBox>
  <interactivity:Interaction.Behaviors>
    <core:EventTriggerBehavior EventName="KeyDown">
      <core:InvokeCommandAction
        Command="{Binding MyCommand, Mode=OneWay}"
        InputConverter="{StaticResource KeyEventArgsToStringConverter}"
        InputConverterParameter="This is a {0} test" />
    </core:EventTriggerBehavior>
  </interactivity:Interaction.Behaviors>
</TextBox>
```

- Other XAML frameworks:
  - No InputConverter!

Summary

- **Binding Views and ViewModels, strategies**
  - Using the ViewModelLocator
  - In code behind
- **Why do we have extras?**
- **Managing dependencies with SimpleIoc**
- **Handling events and commands with EventToCommand**