Contents

Punch integrates MEF with Caliburn.Micro. This allows for wiring the ViewModels and its dependencies through dependency-injection for loose-coupling of all the parts that make up the application, ease of extending an application, and ease of maintenance.

At this point we assume that you are somewhat familiar with MVVM and Caliburn.Micro. For more information on Caliburn.Micro, please visit the Caliburn.Micro Documentation.

Basic ViewModel

Caliburn.Micro offers a number of base classes to build ViewModels with certain attributes and behavior. In its most basic form, though, a ViewModel is simply a POCO.

```
using System.ComponentModel.Composition;
using Caliburn.Micro;
[Export]
public class BasicViewModel : Screen
{
  // Add Data Properties and Actions
}
```

In the above example we are using the Caliburn.Micro Screen class as the base. The only other thing required is the [Export] attribute that is used to register the ViewModel with MEF, and allow MEF to create instances as required by the application as well as injecting the necessary dependencies into it. The above example doesn't have any dependencies.

ViewModel with Dependencies

The most common dependency of a ViewModel is the UnitOfWork. We use it to retrieve and save data and perform business logic on the data. Just like the ViewModels, the UnitOfWork is registered with MEF. Most often though, the UnitOfWork is registered not by the concrete type, but by an interface, so that it can be substituted for unit testing for example.

```
using System.ComponentModel.Composition;
using Caliburn.Micro;
[Export]
public class BasicViewModel : Screen
{
  private readonly IUnitOfWork<Customer> _uow;
   [ImportingConstructor]
  public BasicViewModel(IUnitOfWork<Customer> uow)
   {
      _uow = uow;
   }
  // Add Data Properties and Actions
}
```

ViewModel Composition

More complex ViewModels are often composed together of multiple smaller ViewModels. This can be achieved through the same dependency injection mechanisms. Communication between the composed ViewModels is often done through messaging with the help of an event aggregator. Caliburn.Micro offers such an event aggregator and Punch registers it with MEF.

```
using System.ComponentModel.Composition;
using Caliburn.Micro;
using Cocktail;
public class DoSomethingMessage
{
}
[Export]
public class MenuViewModel : Screen
{
  public void MenuItemClicked()
   {
```

```
        EventFns.Publish(new DoSomethingMessage());
    }
}
[Export]
public class WorkspaceViewModel : Screen, IHandle<DoSomethingMessage>
{
   public WorkspaceViewModel()
    {
        EventFns.Subscribe(this);
    }
   public void Handle(DoSomethingMessage message)
    {
      // Do something
   }
}
[Export]
public class Shell : Screen
{
    [ImportingConstructor]
   public Shell(MenuViewModel menuViewModel, WorkspaceViewModel workspaceViewModel)
    {
        MenuViewModel = menuViewModel;
        WorkspaceViewModel = workspaceViewModel;
    }
   public IScreen MenuViewModel { get; private set; }
   public IScreen WorkspaceViewModel { get; private set; }
   // Add Data Properties and Actions
}
```

Windows Store apps

In Windows Store apps, MEF attributes are not required. Punch automatically registers every class ending in ViewModel. To instantiate the ViewModel, MEF selects the constructor with the most parameters. If a different constructor is supposed to be used, mark the constructor with the ImportingConstructorAttribute.