MVVM

Model, View, ViewModel

- Why separation?
- Explain the three components, focus on the ViewModel

Sample Application

- Any idea for a customer specific sample app?
  - Fallback: Cookbook
- Create Models, ViewModels, Views
- Possible topics on the way
  - INotifyPropertyChanged, ObservableCollection
  - Be aware of the "hidden" WPF feature: change notification works, as long as all changes are done through databinding (not in code) and the DataContext classes are **not** implementing INotifyPropertyChanged. As soon as INPC is implemented, every magic stops and you are responsible for notifying.
  - SelectedItem-Binding for ItemsSource
  - Commands (see DelegateCommand below)

Creating a RelayCommand

```
public class RelayCommand : ICommand
{
    private Action<object> execute;
    private Predicate<object> canExecute;

    public RelayCommand(Action<object> execute, Predicate<object> canExecute = null)
    {
        this.execute = execute;
        this.canExecute = canExecute;
    }

    public bool CanExecute(object parameter)
    {
        return canExecute == null || canExecute(parameter);
    }

    public event EventHandler CanExecuteChanged;

    public void Execute(object parameter)
    {
        execute(parameter);
    }
}
```

Use a MVVM library

- Install MvvmLight (Libs doesn't add any scaffolding, just the libraries), see mvvmlight.net
- ViewModelBase
  - Set-method
  - RaisePropertyChanged
- ViewModelLocator
  - `DataContext="{Binding Main, Source={StaticResource Locator}}"`
- RelayCommand

Use an IoC framework (SimpleIoC comes with MvvmLight)

- used for ViewModels, Services, etc.

```
public ViewModelLocator()
{
    ServiceLocator.SetLocatorProvider(() => SimpleIoc.Default);

    if (ViewModelBase.IsInDesignModeStatic)
    {
        SimpleIoc.Default.Register<IInstrumentManager, DesignModeInstrumentManager>();
    }
    else
    {
        SimpleIoc.Default.Register<IInstrumentManager, InstrumentManager>();
    }

    // InstrumentManager gets injected by SimpleIoc
    SimpleIoc.Default.Register<InstrumentViewModel>();
}
```