Quick Start

Create interface of your viewmodel and derive it from **MVVMDynamic.IViewModel.**

```
internal interface ICarViewModel : IViewModel
{
        bool SirenIsOn { get; set; }
        bool HeadlightsAreOn { get; set; }
        void Beep();
        float TurnSpeed { get; set; }
        void ShowText(string text);
}
```

Now you can create instances of generated viewmodel by calling **MVVMDynamic.TypeEmitter.Instance.CreateViewModel<>**

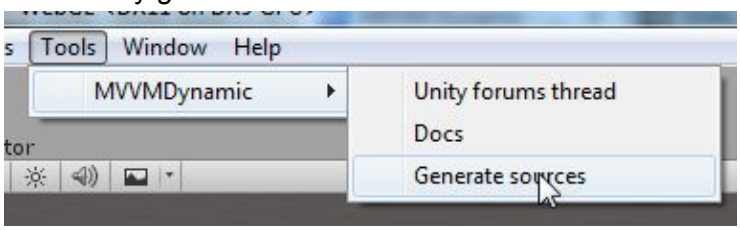
```
public void Awake()
{
        _carViewModel = TypeEmitter.Instance.CreateViewModel<ICarViewModel>();
}
```

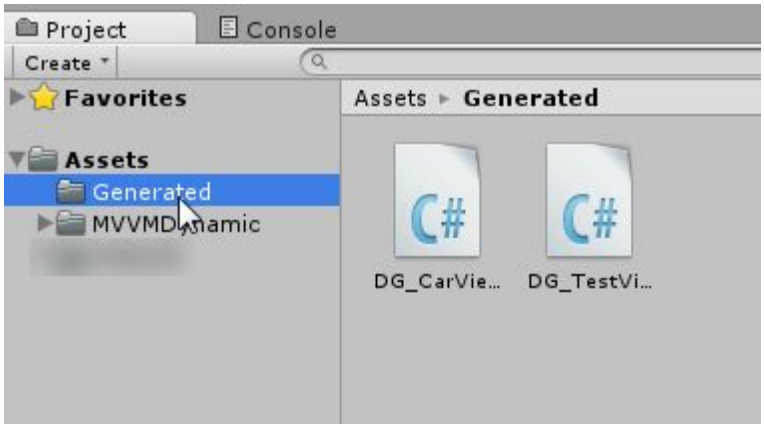
Note that **IViewModel** interface is derived from **INotifyPropertyChanged.**

```
public interface INotifyPropertyChanged
{
        void NotifyPropertyChanged(string propertyName, object oldValue, object newValue);
        event PropertyChangedEventHandler PropertyChanged;
}
```

Every time any property of your viewmodel changes **PropertyChanged** event will be invoked. Arguments of this event will contain old/previous value and new/just set value. **PropertyChanged** event also will be invoked on any **action** (void method) of viewmodel been called, but in this case arguments will contain first and second arguments of the action.

Well, that's it. Your viewmodel is ready. As you can see, there is not much magic happens inside generated viewmodel class, but if you want to see what exactly is inside this class, you can forcibly generate viewmodel sources.





But of course using **MVVMDynamic.Binder** is much more convenient.
It allows you not only bind to particular property/action, but also easily unbind them in a bunch.

Binder

Binder subscribes to **NotifyPropertyChanged** event of the passed viewModel and calls passed action when such event occur.

- Creating binder:

```
private Binder _binder = new Binder();
```

- Binding property:

```
_binder.BindProperty(_carViewModel, vm => vm.SirenIsOn,
              vm => _sirenAudioSource.enabled = vm.SirenIsOn, true);
```

first argument - **viewmodel**
second argument - **lambda** with property member
third argument - **action** that should be called on property changed

forth argument, optional - should call action immediately after binding? Useful for **initial update** of the view.

- Binding action:

```
_binder.BindAction(_carViewModel, vm => vm.Beep(), vm => _beepAudioSource.Play());
```

Not much difference. In this case lambda points on method instead of property.

- Unbinding:

Now if we want to swap viewmodels, we can easily drop the previous one.

```
private void Unbind()
{
        _binder.Reset(_carViewModel);
        //OR
        _binder.Reset();
}
```

Binder.Reset(viewModel) - drops all subscriptions to **viewModel**
Binder.Reset() - drops all subscriptions made with this binder

Pretty convenient to use it inside **OnDestroy** event to avoid ***The object of type <> has been destroyed but you are still trying to access it.*** exceptions.

```
public void OnDestroy()
{
        _binder.Reset();
}
```

Delaying Events

Sometimes state of view depends on multiple viewmodel properties simultaneously. In that case you'd desire to fully update viewmodel first and invoke any events later, that way on the moment when event received, every property will contain relevant data and their previous values will be still available and passed with events args.

Every generated viewmodel is derived from **ViewModelBase.**
ViewModelBase contains two methods:

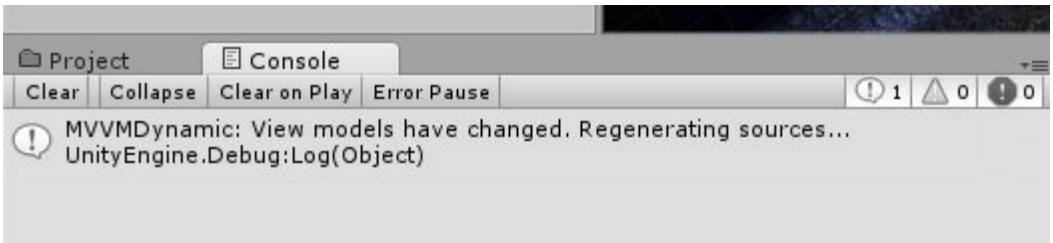
HaltEvents()
and
DropEvents()

But they are not available in **IViewModel** interface.
Instead there are two extensions for **IViewModel** in **MVVMDynamic.ViewModelExtensions** class that you can use like this:

```
        viewModel.Update(
                () =>
                {
                    viewModel.Test1 = 15;
                    viewModel.Test2 = "Test text";
                    viewModel.Test3 = 1234567890;
                });
or
       viewModel.Update(
                vm =>
                {
                    vm.Test1 = 15;
                    vm.Test2 = "Test text";
                    vm.Test3 = 1234567890;
                });
```

AOT Platforms

When target platform set to **iOS** or **WebGL** MVVM Dynamic generates viewmodel sources every time when any your viewmodel interface was changed.



Note that inside the editor MVVM Dynamic will still use IL generation and sources will be used only for release build.
You can find and inspect generated sources in **Assets/Generated** folder

Using own / non-primitive types

Implement **==** and **!=** operators and it’s ready.