



This repository Search

Explore Gist Blog Help



NVentimiglia



NVentimiglia / Unity3d-Databinding-Mvvm-Mvc

Unwatch

3

Unstar

13

Fork

3

branch: master

Unity3d-Databinding-Mvvm-Mvc / README.md



NVentimiglia 6 days ago Update README.md

1 contributor

141 lines (91 sloc) 7.534 kb

Raw

Blame

History



Foundation Databinding (v4.0) 3/8/2015

Nicholas Ventimiglia | AvariceOnline.com

Databinding is a mechanism to 'connect' your UI widgets (buttons, input, text, lists) to 'view scripts'. This strategy is necessary for the Model-View-ViewModel architecture that is very popular in the C# world. In MVVM Your views's properties, fields and methods are 'observed' by the UI elements and any changes to your view's are communicated to the UI elements so they may update worry free.

- Supports inheritance, interfaces, structs and using DLLs
- Bind to monobehaviours and plain CLR objects (wont have change notification)
- Foundation.Databinding.Model is a dll, so you can place your game model in a class library
- Bind to methods, fields, properties or coroutines
- Uses IObservableModel interface instead of IPropertyNotifyChange. IObservableModel includes the changed value along side the changed property name as to prevent an additional reflection call.

Dependencies

- [FullSerializer](#) Json Library
- [Foundation.Tasks](#) Async Library
- [Foundation.Localization](#) Translation Library
- Localization may be omitted by using the NoLocalization compilation directive
- [Foundation.Injector](#) Is mentioned in the comments. It is not needed but I personally use it so that I dont need to reference my components in the editor.

Model Logic

It is recommended that you inherit your monobehaviours from ObservableBehaviour and your clr objects from ObservableObject. These classes implement the IObservableModel interface for you and include a number of helper methods for quickly publishing changes to your views.

Example Model

Here is an example model with a single observable property. when the property changes the view (or any other listener) will receive the change event and have an opportunity to update itself.

```
public class MyModel : ObservableObject {
```

```

private int _myProperty;
public int MyProperty
{
    get { return _myProperty; }
    set
    {
        // Prevent Stack Overflow in two way binding scenarios
        if (_myProperty == value)
            return;

        _myProperty = value;

        // Notify Listeners
        NotifyProperty("MyProperty", value);
    }
}
}

```

Models vs ViewModels

Extending from ObservableBehaviour requires that you make an instance of your view in the scene (since it is an MonoBehaviour). This has the advantage of letting you set properties in your scene view.

Extending from ObservableScriptableObject requires that you make an instance in your editor (since it is an ScriptableObject). This has the advantage of letting you set properties in your editor.

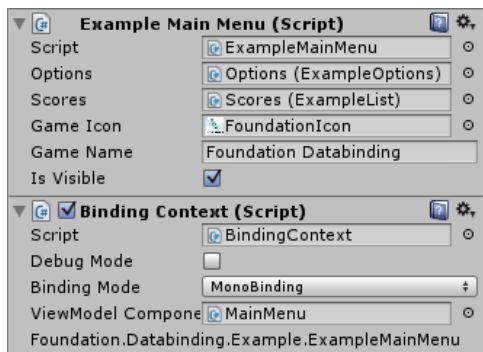
Extending from ObservableObject is my personal choice for most data objects (high scores, profiles, ect). It is a light weight CLR object which does not depend on the UnityEngine.

View Logic

Once you have your model defined and a mockup of your view using Unity's uGUI UI framework it is time to connect the two.

Binding Context

The binding context is responsible for gluing your model to your view elements. Once set child binders will update with a listing of properties to bind to.



This script sits at the root of your UI and operates in one of three modes.

- MonoBinding

Viewmodel is a monobehaviour (or observable behavior). Drag and drop the behavior into the editor field to bind to it.

- MockBinding

Viewmodel is a late bound. Use this option when you would like to set the datatype for child binders while not actually setting the model instance. Example uses

include list controls where you may know the type of the list item but the actual item instance is set at run time.

- Property Binding (Hierarchy Binding)

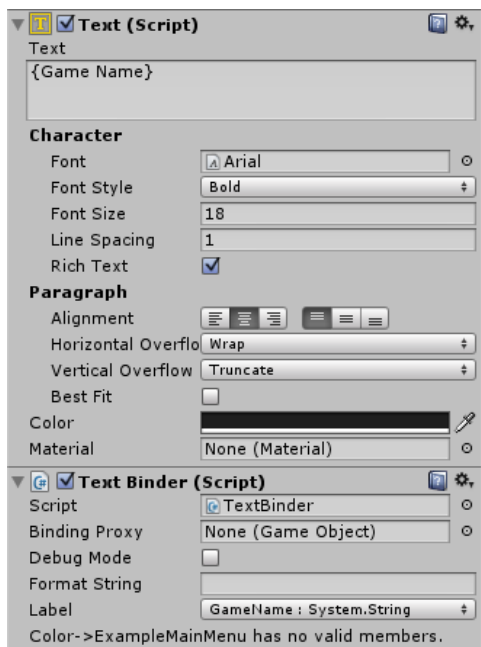
This allows you to index into a model within another model. For instance `MyModel.MyUser.MyName` where `MyModel` is the parent viewmodel, `MyUser` is a child model and `MyName` is a property of the child most model.

Binders

Binders sit underneath a binding context and next to their associated ui element. When the binding context is set the binders will update with a listing of valid properties and methods from the model.

There are a number of binders including `InputFieldBinder`, `ButtonBinder`, `ListBinder` and `ImageBinders`. I believe I have most major controls. If you need a custom binder inherit from `BinderBase`. If you feel that I have neglected a non specialized binder, let me know and I will make it.

Using a binder is straight forward. Just drag and drop the script onto the control that it will bind to. For instance if you want databinding on an Input Field include the Input Field binder as well.



AudioManager

I have included a `AudioManager` in this library. The `AudioManager` extends Unity's audio system with audio layers. This will allow the players to fine tune audio volume by layer (music, sound, ect).

- **Audio2DListener** : Attach to the main camera(s). Used to spawn audio sources for UI elements.
- **Audio2DSource** : for global (ui) sounds. Use just like the `AudioSource`.
- **AudioRegulator** : Attach to `AudioSources` to regulate audio volume by layers (music, ui, ect)
- **AudioManager** : A C# manager for mediating changes to layer volumes.

More

Part of the Unity3d Foundation toolkit. A collection of utilities for making high quality data driven games.

<http://unity3dFoundation.com>

- **Tasks** : An async task library for doing background work or extending coroutines with return results.
- **Messenger** : Listener pattern. A message broker for relaying events in a loosely coupled way. Supports auto

subscription via the [Subscribe] annotation.

- **Terminal**: A in game terminal for debugging !
- **Injector**: Service Injector for resolving services and other components. Supports auto injection using the [Inject] annotation
- **DataBinding** : For MVVM / MVC style databinding. Supports the new uGUI ui library.
- **Localization** : Supports in editor translation, multiple files and automatic translation of scripts using the [Localized] annotation.
- **Cloud** : Parse-like storage and account services using a ASP.NET MVC back end. Need to authenticate your users? Reset passwords with branded emails? Save high scores or character data in a database? Maybe write your own authoritative back end? This is it.
- **Lobby** : The ultimate example scene. Everything you need to deploy for a game, minus the actual game play.

License

[MIT License](#)

Donations

[I accept donations via papal.](#) Your money is an objective measure of my self esteem.

