IUnified
InterfacesForUnity

Quickstart

Simply define a new class that derives from ***IUnifiedContainer<T>*** (where ***T*** is your interface) and decorate it with the ***[System.Serializable]*** attribute.

```
using System;

public interface IEngine { /*...*/ }

public interface IAlarm { /*...*/ }

[Serializable]
public class IEngineContainer : IUnifiedContainer<IEngine> { }

[Serializable]
public class IAlarmContainer : IUnifiedContainer<IAlarm> { }
```

Any exposed field in your ***MonoBehaviour*** script that's of your derived type will automatically render in the editor using its custom property drawer.

```
using UnityEngine;

public class MyScript : MonoBehaviour
{
    public IEngineContainer Engine;

    public IAlarmContainer Alarm;
}
```

And that's it!

Abstracting Container

You can reference the interface from within code by accessing the container's ***Result*** property, or create a wrapping property around that and make the container ***private*** to pretty much forget about it altogether.

```
using UnityEngine;

public class MyScript : MonoBehaviour
{
    public IMyInterface Interface
    {
        get { return _interface.Result; }
        set { _interface.Result = value; }
    }

    [SerializeField]
    private IMyInterfaceContainer _interface;
}
```

Now the rest of your code doesn't need to know about the container type, it just deals with your interface directly:

```
using UnityEngine;

public class MyOtherScript : MonoBehaviour
{
    public MyScript MyScript;

    public void Example()
    {
        IMyInterface item = MyScript.Interface;
        item.InterfaceMethod();
        MyScript.Interface = new MyImplementation();
    }
}
```

If you go this route, make *sure* you decorate the private field with the ***[SerializeField]*** attribute or else it will not be exposed to the editor and Unity will not remember it when serializing/deserializing.

Abstracting Container Collection

You can similarly abstract a List of container derived types behind an ***IList<TInterface>*** by using the included ***IUnifiedContainers*** object, which is constructed given a delegate that returns the backing ***List<TContainer>*** field of the class. To implement a setter, use the included ***ToContainerList*** extension method as shown.

```
using UnityEngine;
using System.Collections.Generic;
using Assets.IUnified;

public class MyScript : MonoBehaviour
{
    public IList<IMyInterface> Interfaces
    {
        get
        {
            if(_interfacesDelegate == null)
            {
                _interfacesDelegate = new IUnifiedContainers<IMyInterfaceContainer, IMyInterface>
                    (() => _interfaces);
            }
            return _interfacesDelegate;
        }
        set
        {
            _interfaces = value.ToContainerList<IMyInterfaceContainer, IMyInterface>();
        }
    }
    private IList<IMyInterface> _interfacesDelegate;

    [SerializeField]
    private List<IMyInterfaceContainer> _interfaces;
}
```

This will allow you to reference your interfaces directly without having to access the Result property like so:

```
using UnityEngine;

public class MyOtherScript : MonoBehaviour
{
    public MyScript MyScript;

    public void Example()
    {
        foreach(var item in MyScript.Interfaces)
        {
            item.InterfaceMethod();
        }

        IMyInterface indexedItem = MyScript.Interfaces[0];
        MyScript.Interfaces[1] = new MyImplementation();
        MyScript.Interfaces = new[]
            {
                new MyImplementation(),
                new MyImplementation()
            };
    }
}
```

UI

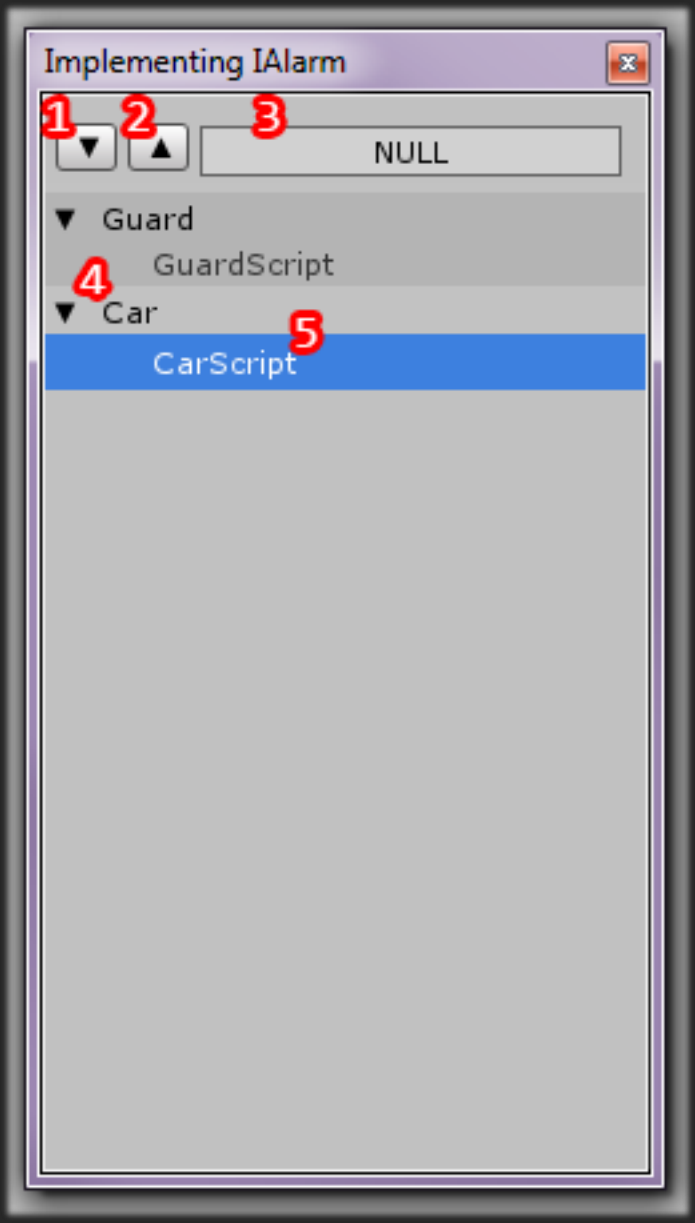
Property Drawer

1. Field Name
2. Value – Displays the reference that is currently implementing the interface in ***GameObject (Component)*** format if it is being implemented by a component; otherwise will display the type of the object that is currently implementing the interface or ***null*** if nothing is.

   You can drag and drop ***Components*** or ***GameObjects*** here to set it to a value. If it is currently being implemented by a ***Component*** then you can click on it to ping the parent ***GameObject*** in the editor.

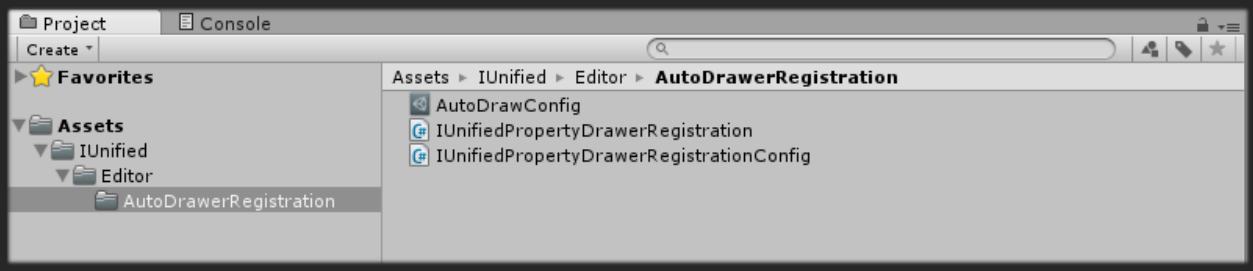
3. Click to set the value to ***null***.
4. Click to open a list of all ***GameObjects*** and ***Components*** that implement the interface.

Selection List



1. Expand all hierarchies below.
2. Collapse all hierarchies below.
3. Set value to null.
4. The ***GameObject*** with ***Components*** or children with ***Components*** that implement the interface. Click the foldout icon the expand/collapse the hierarchy and the name of the object to ping it in the editor.
5. The ***Component*** that implements the interface, click to select.

Automatic Property Drawer



Included is a special component that will automatically register the ***IUnifiedContainerPropertyDrawer*** to draw all properties that derive from ***IUnifiedContainer<T>*** without the necessity of a unique property drawer or editor each time you define a new type. This is made possible by using ***Reflection*** to register the property drawer for each derived type every time Unity serializes/deserializes - as such this approach is vulnerable to changes in Unity's internal workings in the future.

If this mechanic ever stops functioning then you must decorate each field that's of a type that derives from ***IUnifiedContainer <T>*** with the ***[IUnifiedContainer]*** attribute in order for it to be integrated with the editor.

You can also switch this mechanic on or off by selecting the option from the ***Edit > IUnified > Property Drawer Auto-Registration*** menu item, or permanently disable the mechanic by simply deleting the **AutoDrawerRegistration** folder altogether.

Well, that's about it! If you have any questions you can reach me at ***woundedwolfgames@gmail.com*** and I'll get back to you as soon as I can.

Thank you for your support!
Roman Habib Issa