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*** or ***ScriptableObject*** 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 Containers

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 in the inspector and Unity will not remember it when serializing/deserializing.

Also, please note that although you can set the ***Result*** property in code to *anything* that implements your interface, only ***UnityEngine.Object*** derived objects will be serialized by Unity – other non-serializable type references will be lost and are therefore only allowed to be set while your application is actually running.

Abstracting Container Collections

You can similarly abstract a List of container derived types behind an ***IList<TInterface>*** property 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.

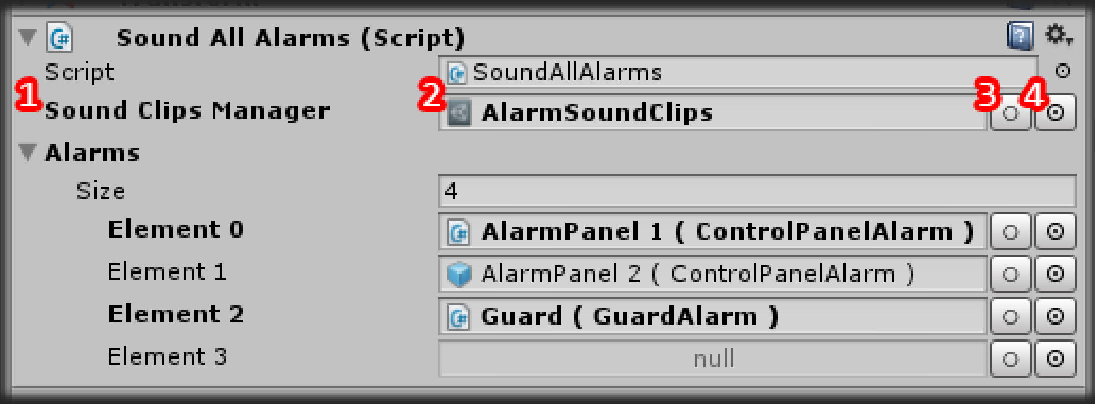
```
using UnityEngine;
using System.Collections.Generic;
using Assets.IUnified;

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



Sound All Alarms (Script)

Script	SoundAllAlarms
Sound Clips Manager	**AlarmSoundClips**
Alarms	
Size	4
Element 0	**AlarmPanel 1 (ControlPanelAlarm)**
Element 1	AlarmPanel 2 (ControlPanelAlarm)
Element 2	**Guard (GuardAlarm)**
Element 3	null

1. Field Name

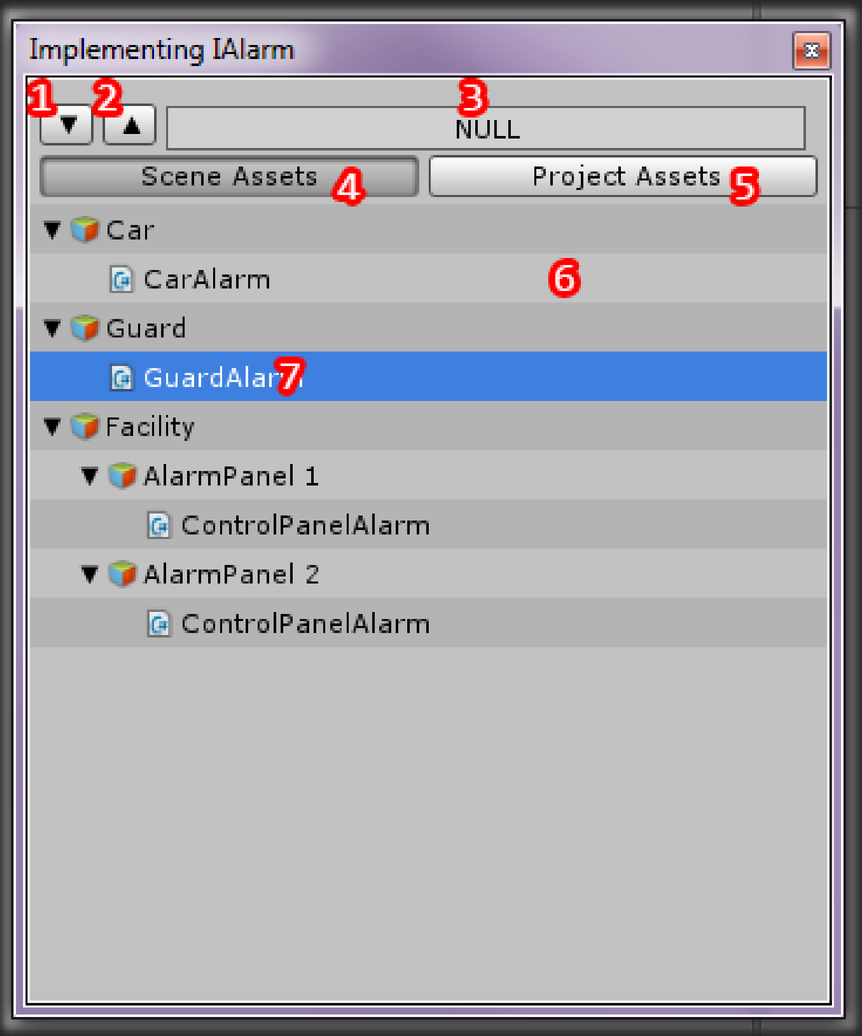
2. Displays the reference that is currently implementing the interface in ***GameObject (Component)*** format if it is being implemented by a ***Component*** (such as a ***MonoBehaviour***). If the implementation is of a serializable asset (like a saved ***ScriptableObject***), it will display the name of the asset. Otherwise, it will display the type name of the object that is currently implementing the interface or ***null*** if nothing is.

   You can drag and drop ***Components*** or ***UnityEngine.Object*** derived objects that implement the interface here to set the reference. If the implementation is pingable then you can click here to do so and have Unity highlight the location of the object in the project.

3. Click to set the value to ***null***.

4. Click to open a list of all serialized objects currently loaded that implement the interface.

Selection List



1. Expand all hierarchies.

2. Collapse all hierarchies.

3. Set value to null.

4. If available, click to display all currently loaded scene objects (such as component ***MonoBehaviours***) that implement the interface.

5. If available, click to display all currently loaded project assets (such as prefabs and saved ***ScriptableObjects***) that implement the interface.

6. If an object is pingable, click on or next to the name when the cursor turns into a magnifying glass to have Unity highlight it in the editor.

7. If an object is selectable, click on its name when the cursor turns into a hand icon to select it.

Well, that’s about it! If you have any questions, comments, or support requests you can reach me on the Unity forums at:

http://forum.unity3d.com/threads/206988-RELEASED-IUnified-C-Interfaces-for-Unity!

Or e-mail me directly:

woundedwolfgames@gmail.com

And I’ll get back to you as soon as I can.

Thank you for your support and good coding to you!

Roman Habib Issa