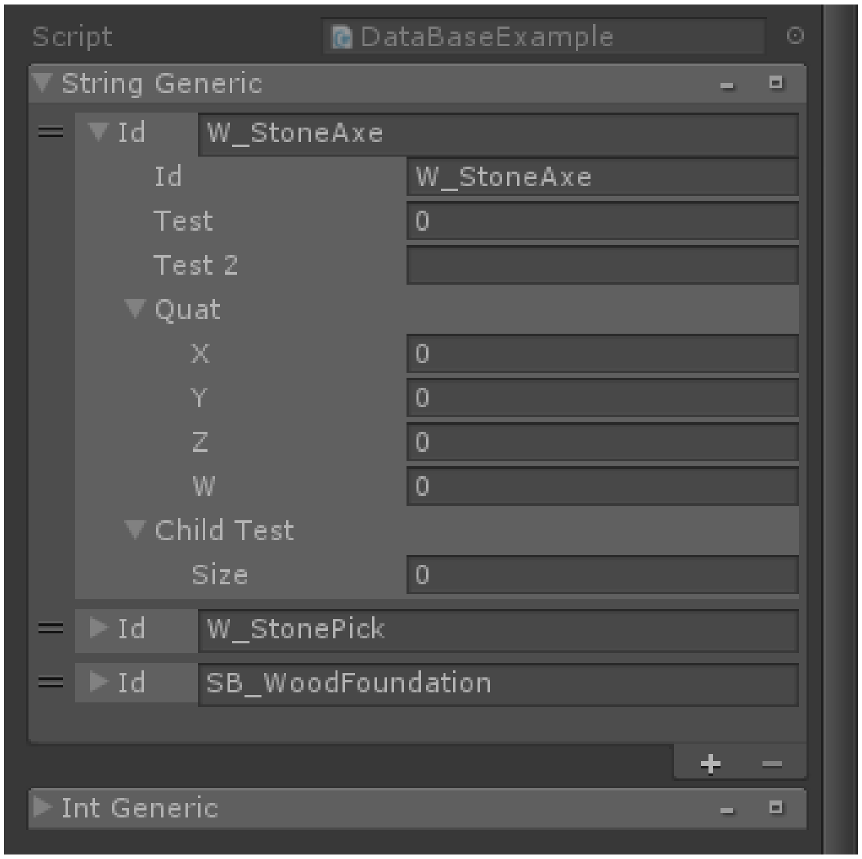
Serializable Dictionary

Rotary Heart

Documentation Version: 1.4

Email: ma.rotaryheart@gmail.com



```
Script                DataBaseExample
String Generic
  Id   W_StoneAxe
     Id          W_StoneAxe
     Test        0
     Test 2
     Quat
        X        0
        Y        0
        Z        0
        W        0
     Child Test
        Size     0
  Id   W_StonePick
  Id   SB_WoodFoundation
Int Generic
```

Description:

This package contains a class, SerializableDictionaryBase.cs, that can be inherited to be able to have a serializable dictionary. There is no limitation for what to use as Key or as Value, other than it being a serializable type. It also contains a class, DataBaseExample.cs, that has a couple of examples of how to setup the dictionary.

Setup:

To use simply add the SerializableDictionary folder to your project (which is on the package). It doesn’t need to be on root of your project, so feel free to move it wherever you want.

How to use:

This section is divided into 3 different cases for easier explanation, but for any type of implementation you will need to create a custom class, can be a nested class, that inherits from SerializableDictionaryBase and setup the respective types for key and value that are going to be used by the dictionary.

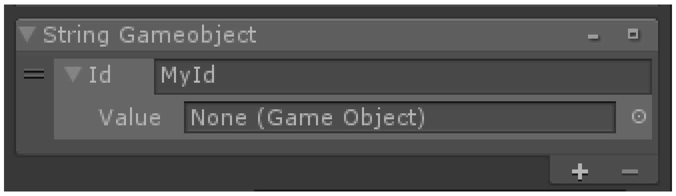
Note that for Unity to be able to serialize this class you need to add the attribute [System.Serializable] to your class since that is the way Unity serialization works.

Contents

Simple Dictionary

```
[System.Serializable]
public class MyDictionary : SerializableDictionaryBase<string, GameObject> {
}
```

On this example you can see that the dictionary that we will be using has a string for key and a GameObject for value. With this now you can have a field of type MyDictionary and you will be able to modify the data from the editor.



Preview of example provided

UnityEngine.Object as a key

If you want to use a UnityEngine.Object (GameObject, Texture, Sprite, etc) type for key, there's an extra step required for the system to handle it. First create your dictionary class.

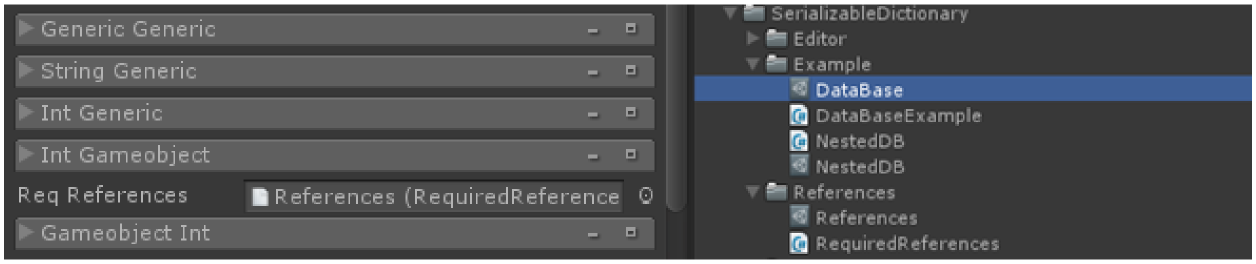
```
[System.Serializable]
public class MyDictionary : SerializableDictionaryBase<GameObject, string> {
}
```

Now you will need to go to the script RequiredReferences.cs and add the reference (if it's not included) needed here (in this case GameObject).

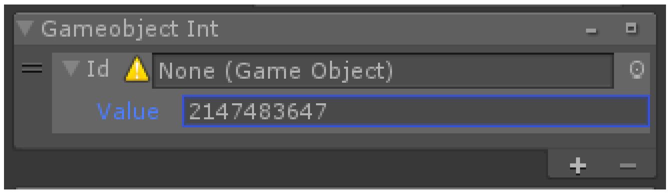
```
[UnityEngine.SerializeField]
private GameObject _gameObject;
```

This is needed because the Dictionary can't handle null for a key so it requires a default value and the editor already gives that value.

The last thing you will need is to setup the reference for the RequiredReferences object on your editor. Drag and drop the References scriptable object to the editor Req Reference field on top of the dictionary.



Now that everything is setup correctly you can start adding elements to your dictionary.



Advanced Dictionary

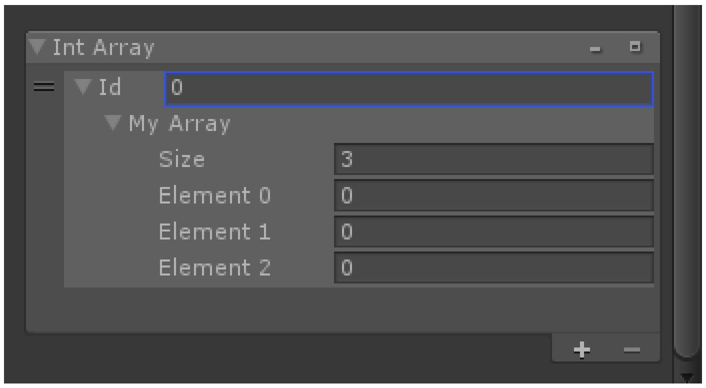
If the value that you want to use is either an Array or a List, you cannot use it directly into the dictionary value type because Unity doesn't serialize Array of Array or List of List. Instead you need to wrap your Array or List with a custom class. See the following example.

```
[System.Serializable]
public class Int_IntArray : SerializableDictionaryBase<int, MyClass> { }

[System.Serializable]
public class MyClass
{
        public int[] myArray;
}

[SerializeField]
private Int_IntArray _intArray;
```

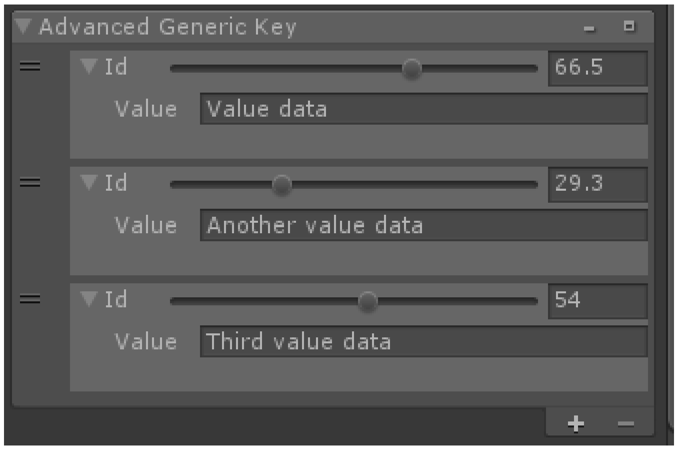
This way unity can serialize your array and the system will work without any problems.



```
Int Array
  Id          0
    My Array
      Size        3
      Element 0   0
      Element 1   0
      Element 2   0
```

Advanced Key

The system comes with an advanced attribute that can be used to allow keys to use any custom drawer, but to be able to use this we need to let the compiler know how to compare this advanced key. We will be using the example provided on the package to explain how to make it work.



For this example, we are using the attribute Range to allow the key to be drawn with the built-in range field. For this we need to setup the code as following:

Include the attribute to the class

```
[System.Serializable]
public class AdvancedGenericClass
{
   [Range(0, 100)]
   public float value;
```

This Equals function is added for convenience, it is not required.

```
   public bool Equals(AdvancedGenericClass other)
   {
      if (ReferenceEquals(null, other)) return false;
      if (ReferenceEquals(this, other)) return true;
      return other.value == value;
   }
```

Override the Equals function to include any logic that we need to make sure the keys are the same

```
    public override bool Equals(object obj)
    {
        if (ReferenceEquals(null, obj)) return false;
        if (ReferenceEquals(this, obj)) return true;
        if (obj.GetType() != typeof(AdvancedGenericClass)) return false;
        return Equals((AdvancedGenericClass)obj);
    }
```

Finally override the GetHashCode and include any kind of HashCode

```
    public override int GetHashCode()
    {
        unchecked
        {
            return value.GetHashCode();
        }
    }
}
```

```
[System.Serializable]
public class AdvanGeneric_String :
SerializableDictionaryBase<AdvancedGenericClass, string> { };
```

```
[SerializeField, DrawKeyAsProperty]
private AdvanGeneric_String _advancedGenericKey;
```

As you can see to make the system ignore the Generic type and draw it as a normal property we need to use the attribute DrawKeyAsProperty on the dictionary field.

Then we need any field that can be serialized by Unity inside the key class. The system will automatically detect the first serializable field and use it, if nothing is found it will fallback on drawing it as a regular generic key.

Now for the dictionary to work correctly comparing the keys we need to override the functions Equals and GetHashCode. Inside this function we will implement any logic that we want to be used to identify that they keys are the same.

Note that this step is required otherwise the keys will be compared by reference and not by the value set on the inspector.