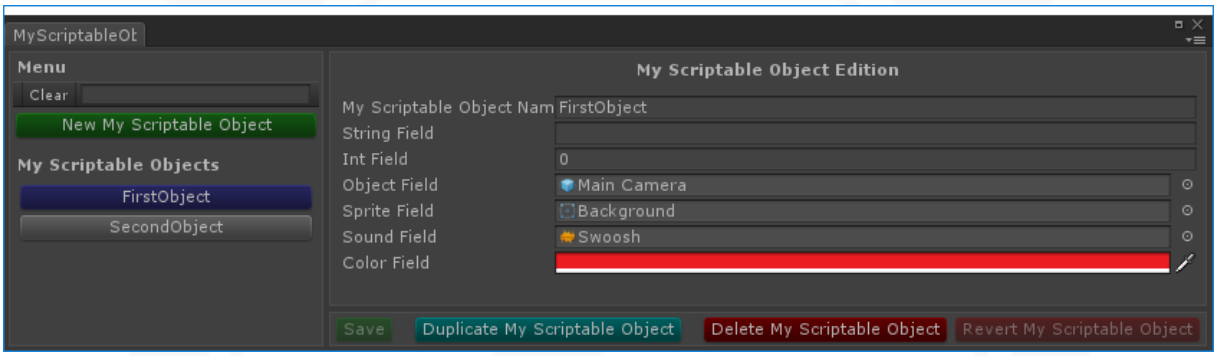
Scriptable Object Suite Documentation

1. What is Scriptable Object Suite?

Scriptable Objects are powerful Unity classes that help you serialize data that you would normally store in a JSON or XML file. Managing them can be painful without a proper toolchain.

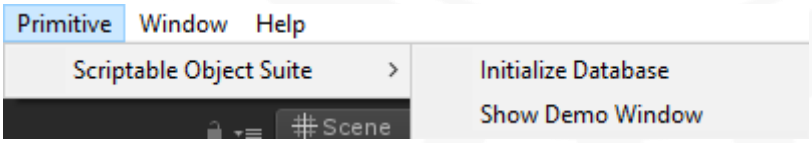
Scriptable Object Suite is a tool that helps you manage, create, edit, delete, and duplicate your Scriptable Objects.



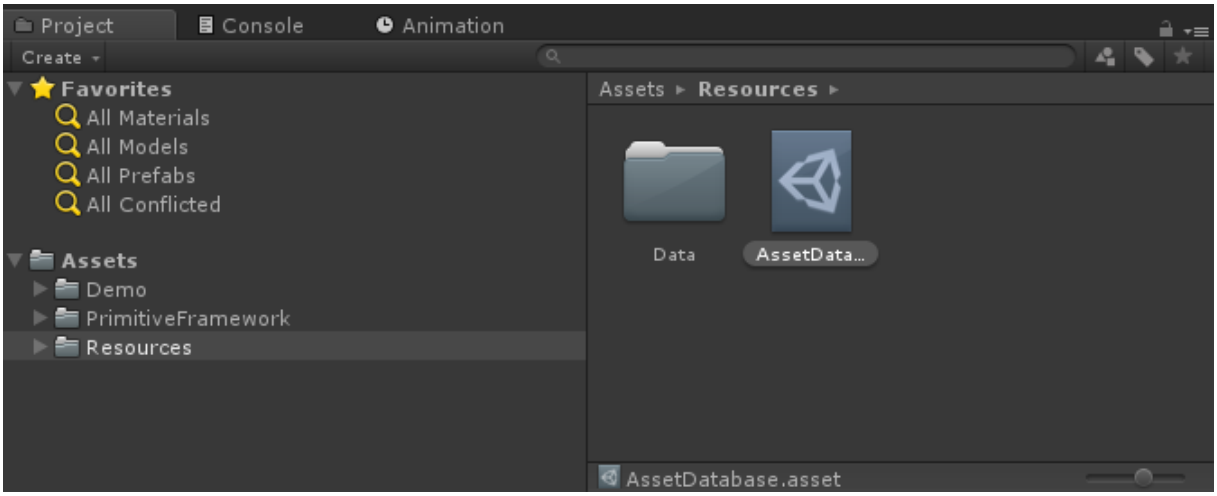
2. How to use Scriptable Object Suite?

Creating the Object Database

Before doing anything, once per project, a Scriptable Object Database has to be initialized. Scriptable Object Suite comes with an option in the Primitive menu of Unity to do so:



If successful, a ScriptableObjectDatabase object should appear in your Assets/Resources folder



Declaring your data

For each type of data, two files have to be defined:

- The Scriptable Object itself. It has to inherit from << *ScriptableObjectExtended* >>
- The Editor Window. It has to inherit from << *ScriptableObjectEditorWindow* >>

In the Scriptable Object script, define all the data fields. They can be of any type.

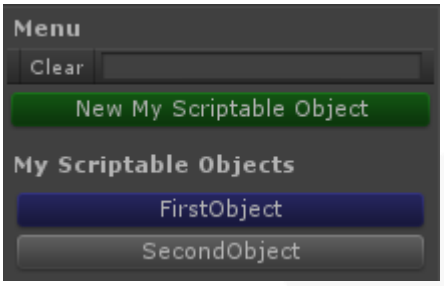
```
public class MyScriptableObject : ScriptableObjectExtended
{
    public string StringField;
    public int IntField;
    public GameObject ObjectField;
    public Sprite SpriteField;
    public AudioClip SoundField;
    public Color ColorField;
}
```

This is an example of a bare bone editor window:

```
public class MyScriptableObjectWindow :
ScriptableObjectEditorWindow<MyScriptableObject>
{
    //---- These are the 3 must-have overrides ----//
    // Name of the object (Display purposes)
    protected override string c_ObjectName { get { return "My Scriptable Object"; } }
    // Relative path from Resource Folder
    protected override string c_ObjectResourcePath { get { return
"Data/MyScriptableObjects/"; } }
    // Relative path from Project Root
    protected override string c_ObjectFullPath { get { return
string.Concat("Assets/Resources/", c_ObjectResourcePath); } }

    [MenuItem("Primitive/Scriptable Object Suite/Show Demo Window")]
    public static void ShowWindow()
    {
        MyScriptableObjectWindow window =
GetWindow<MyScriptableObjectWindow>("MyScriptableObject Editor");
        window.Show();
    }
}
```

Using the Editor Window



Use the left menu to create a new My Scriptable Object and list objects of this type.

A search bar filter is usable to search through all of your objects of this type and find a particular object.

In the main editor window you can edit each object field values.

Remember to use the Save button when you're done. A warning will pop if you change object before saving.

The Duplicate button will create a copy of the current object.
The Delete button will delete the current object.
The Revert button will revert all fields of the current object to their last save values.

3. Advanced use

Improving the Editor Window

Callbacks are defined if you want to perform specific actions

```
protected virtual void OnNewObject(ObjectType obj)

protected virtual void OnDuplicatedObject(ObjectType obj)

protected virtual void OnSavedObject(ObjectType obj)

protected virtual void OnDeletedObject(ObjectType obj)

protected virtual void OnCurrentObjectChanged(ObjectType oldObj, ObjectType newObj)
```

You can also override the Main Window GUI drawing by extending the DrawEditor method

```
protected virtual void DrawEditor(ObjectType target)
```

Using custom property drawers

Scriptable Object Suite is designed to use Serialized Properties to draw each field editor.
If you want to improve upon a certain field drawing, you can use the [CustomPropertyDrawer(typeof(MyType)] attribute to specify a custom property drawer for your type.
The editor window will call this accordingly to draw your property

Loading on demand

Scriptable Objects have the particularity that each and every one of its references are immediately loaded when the ScriptableObject itself is being loaded. That can cause an issue when objects have a long chain of references and you might want to load only part of your object.

The `[LoadOnDemand]` attribute can help you achieve that:

```
[System.NonSerialized]
public GameObject CharacterPrefab;
[SerializeField]
[LoadOnDemand("CharacterPrefab")]
private string m_CharacterPrefabPath;
```

The Suite internally uses unique GUIDs and paths to reference other objects. In this example, `m_CharacterPrefabPath` (string value) is the serialized data, not `CharacterPrefab` itself. To load the prefab, you have to call the `Load()` method.

```
Load(); // Load each and every one of the LoadOnDemand fields
Load("CharacterPrefab"); // Load a specific field
Load("CharacterPrefab, true"); // Load a specific field asynchronously
```

Corresponding `Unload ()` methods:

```
Unload();
Unload("CharacterPrefab");
```

Even though the serialized data is a string, a GameObject field (or whatever field type you defined) will appear in the window.

The system will automatically internally update the corresponding path or GUID

/!\ In order to function properly, the LoadOnDemand asset will need to be in a Resource folder as it's the only way for Unity to actually load assets on demand.

4. Support

The Scriptable Object suite has been developed by Primitive Factory.

For any support, contact assetsupport@primitive-factory.com