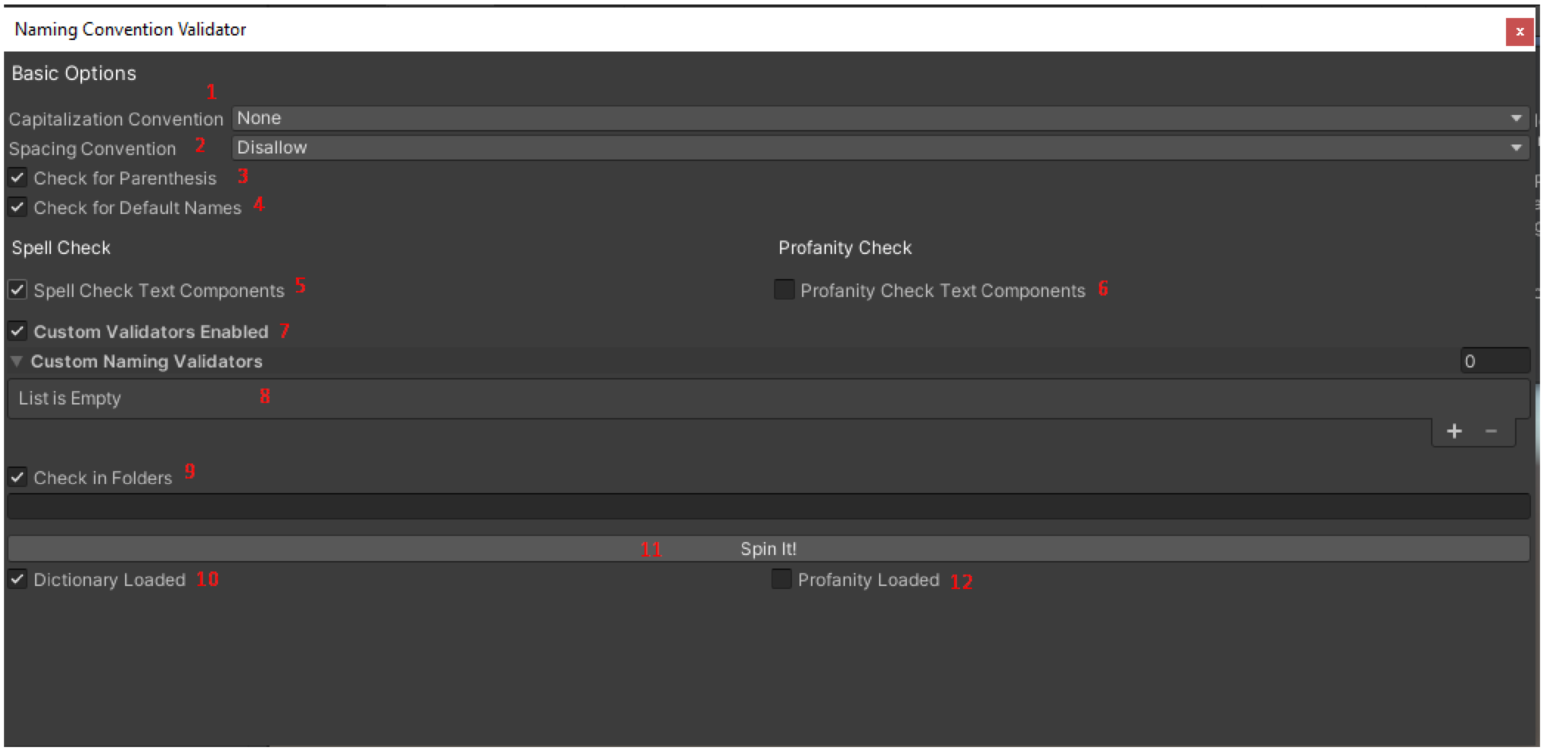
Naming Convention Validator and Spell Checker

Documentation

1. Usage


Naming Convention Validator

Basic Options

Capitalization Convention 1 None

Spacing Convention 2 Disallow

✓ Check for Parenthesis 3

✓ Check for Default Names 4

Spell Check

✓ Spell Check Text Components 5

Profanity Check

☐ Profanity Check Text Components 6

✓ Custom Validators Enabled 7

Custom Naming Validators 0

List is Empty 8

✓ Check in Folders 9

11 Spin It!

✓ Dictionary Loaded 10

☐ Profanity Loaded 12

Basic Options

1. **Capitalization Convention.**
    a. None: capitalization is not checked.
    b. Camel Case: capitalization should follow camel case convention.
    c. Pascal Case: capitalization should follow pascal case convention.
2. **Spacing Convention.**
    a. Allow: spacing is allowed, no checking is done.
    b. Disallow: spacing is not allowed.
    c. Allow only on top objects: only top objects in the hierarchy are allowed to have spaces.
3. **Check for parenthesis**
    Check this if you want to check for parenthesis left over after duplicating or creating objects from the GameObject menu. For example: GameObject (11).
4. **Check for Default Names**

Check this if you want to check for default names that are assigned to objects after creation. For example: (GameObject, Canvas, New Animation).

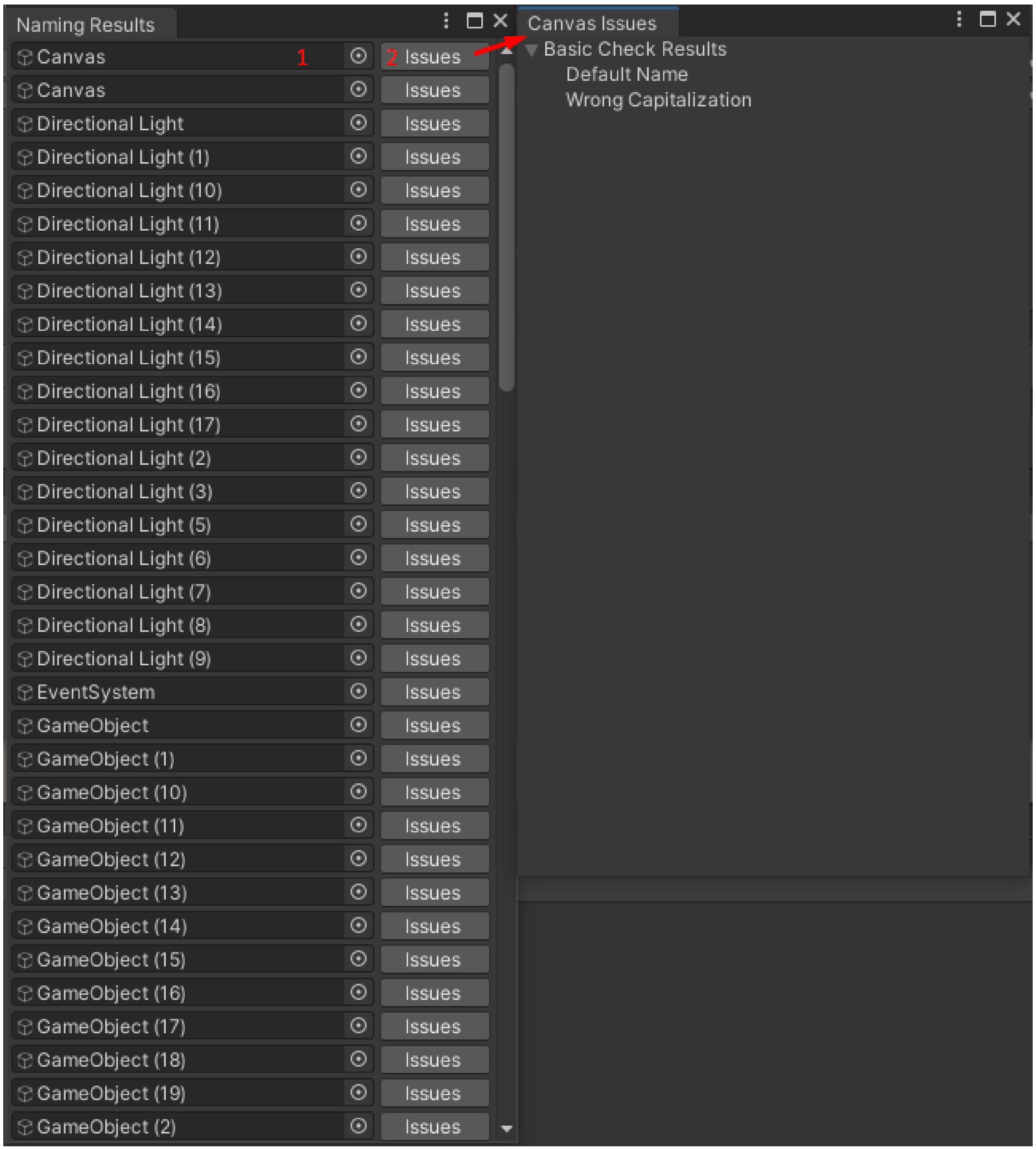
Spell Check

5. **Spell Check Text Components**
   Check this if you want the text components (Text and TMP) in the current scene to be spell checked.
6. **Profanity Check Text Components**
   Check this if you want the text components (Text and TMP) in the current scene to be checked for any profanities.

Custom Check

7. **Custom Validators Enabled**
   Check this if you want to use custom validators (See section 2, Extending the tool).
8. **Custom Validators List**
   Put your custom validators here. (Press the “+” button on the right to add extra validators)
9. **Check in Folders.**
   Check this if you want to check asset names in specific folders. Drag and drop folders into the text fields to input their paths.
10. **Dictionary Loaded (Readonly)**
    This checkmark shows whether or not the dictionary is loaded for spell checking.
11. **Spin it!**
    Click this button to start the checking process.
12. **Profanity Loaded (Readonly)**
    This checkmark shows whether or not the profanity list is loaded for spell checking.

After the checking process finishes, you will be greeted with the **results window**:



Naming Results

Canvas	Issues
Canvas	Issues
Directional Light	Issues
Directional Light (1)	Issues
Directional Light (10)	Issues
Directional Light (11)	Issues
Directional Light (12)	Issues
Directional Light (13)	Issues
Directional Light (14)	Issues
Directional Light (15)	Issues
Directional Light (16)	Issues
Directional Light (17)	Issues
Directional Light (2)	Issues
Directional Light (3)	Issues
Directional Light (5)	Issues
Directional Light (6)	Issues
Directional Light (7)	Issues
Directional Light (8)	Issues
Directional Light (9)	Issues
EventSystem	Issues
GameObject	Issues
GameObject (1)	Issues
GameObject (10)	Issues
GameObject (11)	Issues
GameObject (12)	Issues
GameObject (13)	Issues
GameObject (14)	Issues
GameObject (15)	Issues
GameObject (16)	Issues
GameObject (17)	Issues
GameObject (18)	Issues
GameObject (19)	Issues
GameObject (2)	Issues

Canvas Issues

- Basic Check Results
  - Default Name
  - Wrong Capitalization

1. **Object reference.**

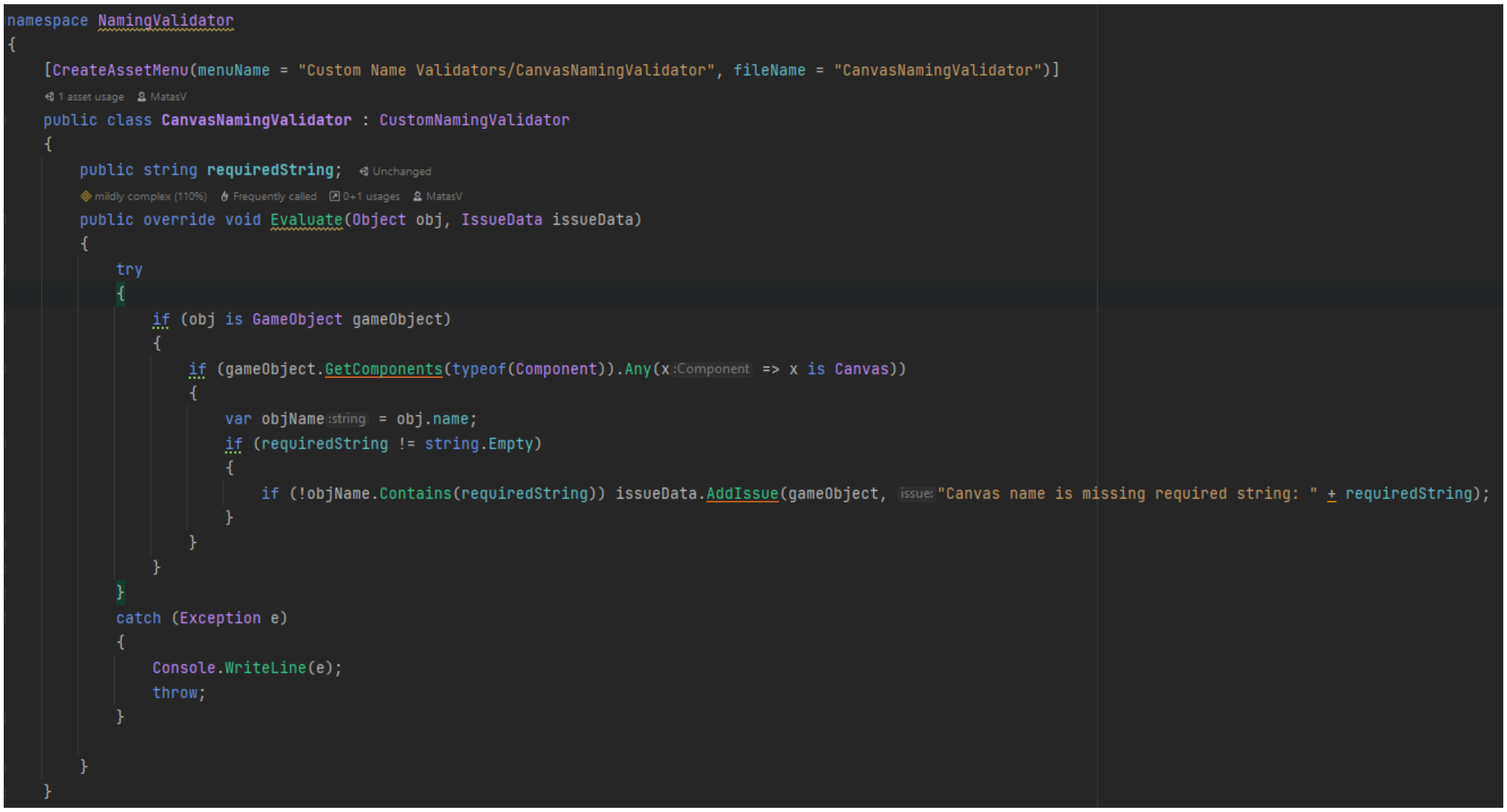
   This is a reference to the object in question. Pressing on the name of the object will highlight it in the Hierarchy (or Project window, if it's an asset) so you can modify its name).
2. **Issue window button**

Press this button to open a new window containing all the issues related to the object.

2. Extending the Tool

The tool is lacking the checks that you require? Write your own! Simply extend the CustomNamingValidator class and override the Evaluate() method.
Packed with the package come two custom evaluators:

CanvasNamingValidators.cs checks whether or not an object containing a Canvas component contains the word “canvas” in its name. (The required string can be changed in the Scriptable Object)



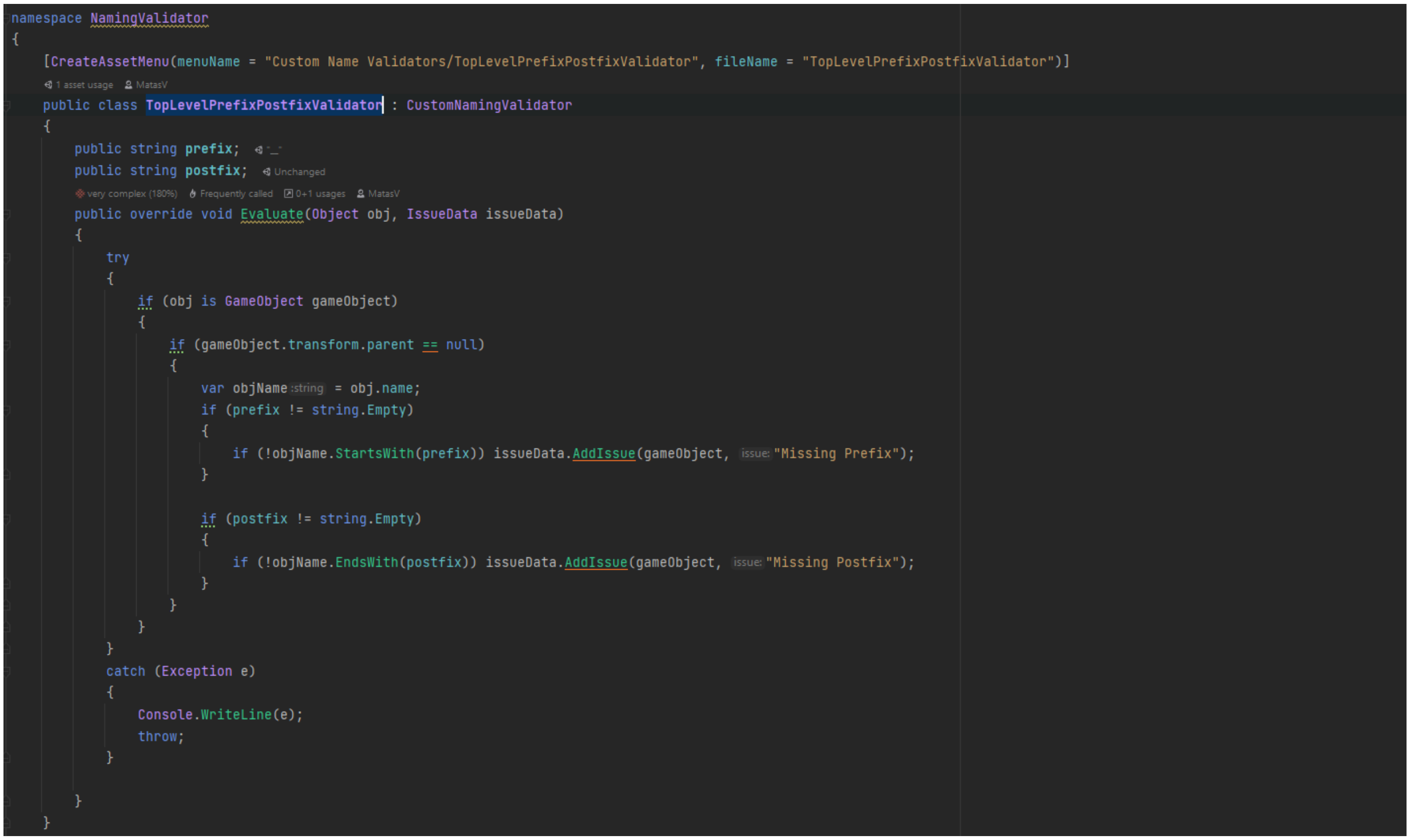
```csharp
namespace NamingValidator
{
    [CreateAssetMenu(menuName = "Custom Name Validators/CanvasNamingValidator", fileName = "CanvasNamingValidator")]
    public class CanvasNamingValidator : CustomNamingValidator
    {
        public string requiredString;
        public override void Evaluate(Object obj, IssueData issueData)
        {
            try
            {
                if (obj is GameObject gameObject)
                {
                    if (gameObject.GetComponents(typeof(Component)).Any(x => x is Canvas))
                    {
                        var objName = obj.name;
                        if (requiredString != string.Empty)
                        {
                            if (!objName.Contains(requiredString)) issueData.AddIssue(gameObject, "Canvas name is missing required string: " + requiredString);
                        }
                    }
                }
            }
            catch (Exception e)
            {
                Console.WriteLine(e);
                throw;
            }

        }
    }
```

Parameters:

- Object obj
  - The object in question.
- IssueData issueData

- A structure containing information about all the objects and their issues. Use .AddIssue(Gameobject object, string issue) to add an issue to the list.

TopLevelPrefixPostfixValidator.cs checks whether or not an object residing in the top level in hierarchy follows the required prefix and postfix rules. (The required strings can be changed in the Scriptable Object)



```csharp
namespace NamingValidator
{
    [CreateAssetMenu(menuName = "Custom Name Validators/TopLevelPrefixPostfixValidator", fileName = "TopLevelPrefixPostfixValidator")]
    public class TopLevelPrefixPostfixValidator : CustomNamingValidator
    {
        public string prefix;
        public string postfix;
        public override void Evaluate(Object obj, IssueData issueData)
        {
            try
            {
                if (obj is GameObject gameObject)
                {
                    if (gameObject.transform.parent == null)
                    {
                        var objName = obj.name;
                        if (prefix != string.Empty)
                        {
                            if (!objName.StartsWith(prefix)) issueData.AddIssue(gameObject, "Missing Prefix");
                        }

                        if (postfix != string.Empty)
                        {
                            if (!objName.EndsWith(postfix)) issueData.AddIssue(gameObject, "Missing Postfix");
                        }
                    }
                }
            }
            catch (Exception e)
            {
                Console.WriteLine(e);
                throw;
            }

        }
    }
```