Type Selector for Unity

Overview

The **Type Selector** package provides a custom **Property Attribute** with a custom **Property Drawer** that allows users to select and instantiate subclasses of a given abstract or base class within the **Unity Inspector**. This is particularly useful for working with **SerializedReference** fields and enables a more user-friendly workflow when dealing with polymorphic objects.

Features

- Custom **PropertyDrawer** for selecting types.
- Supports Unity's **SerializedReference** system.
- Three draw modes:
    - **Default**: Standard foldout-based UI.
    - **NoFoldout**: Displays all properties inline without a foldout.
    - **Inline**: Draws properties in a minimalistic layout.
- Type filtering to only allow valid subclasses.
- Optional custom naming for type selection.
- Supports missing reference cleanup for serialization safety.

Usage

1. Applying the TypeSelector Attribute

To use the Type Selector, apply the `[TypeSelector]` attribute to a `SerializedReference` field:

```
using TypeSelector;
using UnityEngine;

public class ExampleBehaviour : MonoBehaviour
{
    [SerializeReference, TypeSelector, Tooltip("Select a subclass of
ExampleAbstractClass")]
    public ExampleAbstractClass abstractField;
}
```

2. Creating an Abstract Base Class

```
using UnityEngine;

public abstract class ExampleAbstractClass
{
    public abstract void DoSomething();
}
```

3. Creating Subclasses

```
public class ExampleConcreteClassA : ExampleAbstractClass
{
    public override void DoSomething()
    {
        Debug.Log("ExampleConcreteClassA is doing something!");
    }
}

public class ExampleConcreteClassB : ExampleAbstractClass
{
    public override void DoSomething()
    {
        Debug.Log("ExampleConcreteClassB is doing something else!");
    }
}
```

4. Using Custom Display Names

You can specify a custom display name for your classes using `[TypeSelectorName]`:

```
using TypeSelector;

[TypeSelectorName("Custom Class A")]
public class CustomNameExampleClass : ExampleAbstractClass
{
    public override void DoSomething() => Debug.Log("Custom Class A
Selected");
}
```

5. Changing the Draw Mode

```
[SerializeReference, TypeSelector(DrawMode.NoFoldout)]
public ExampleAbstractClass noFoldoutField;
```

Available **DrawMode** options:

- **Default**: Standard foldout-based display.
- **NoFoldout**: Shows properties directly, without a foldout.
- **Inline**: Displays properties with a minimal layout.

Notes

- Ensure that the classes of fields using `[TypeSelector]` are marked with `[SerializeReference]` to work correctly.
- The package automatically filters out invalid types (e.g., abstract or Unity object types).
- If a selected class has missing serialization references, they will be cleared automatically.

Support

For issues, feature requests, or contributions, feel free to contact felipeishiminestore@gmail.com or visit felipeishimine.github.io