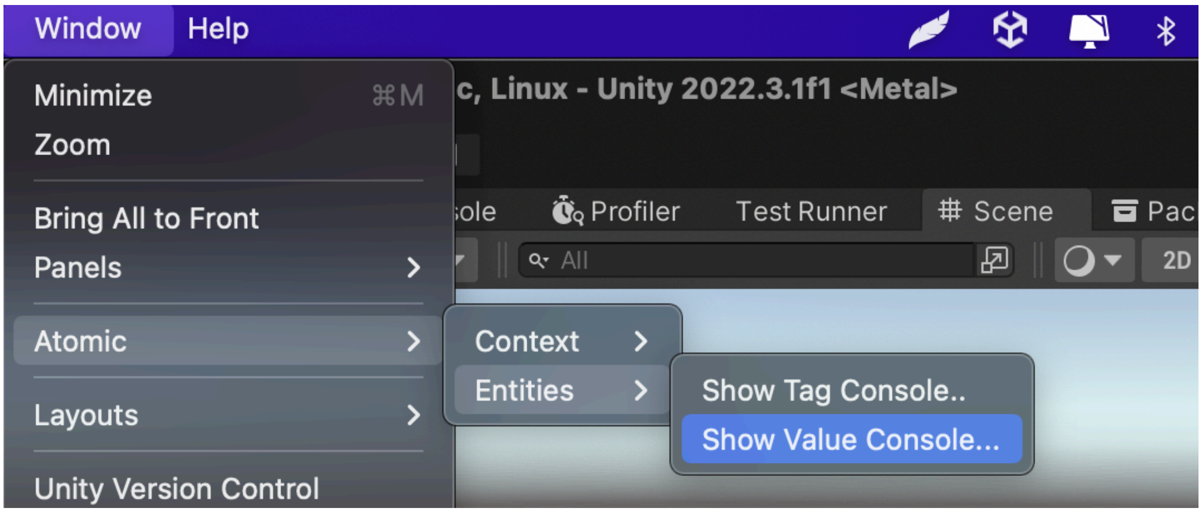
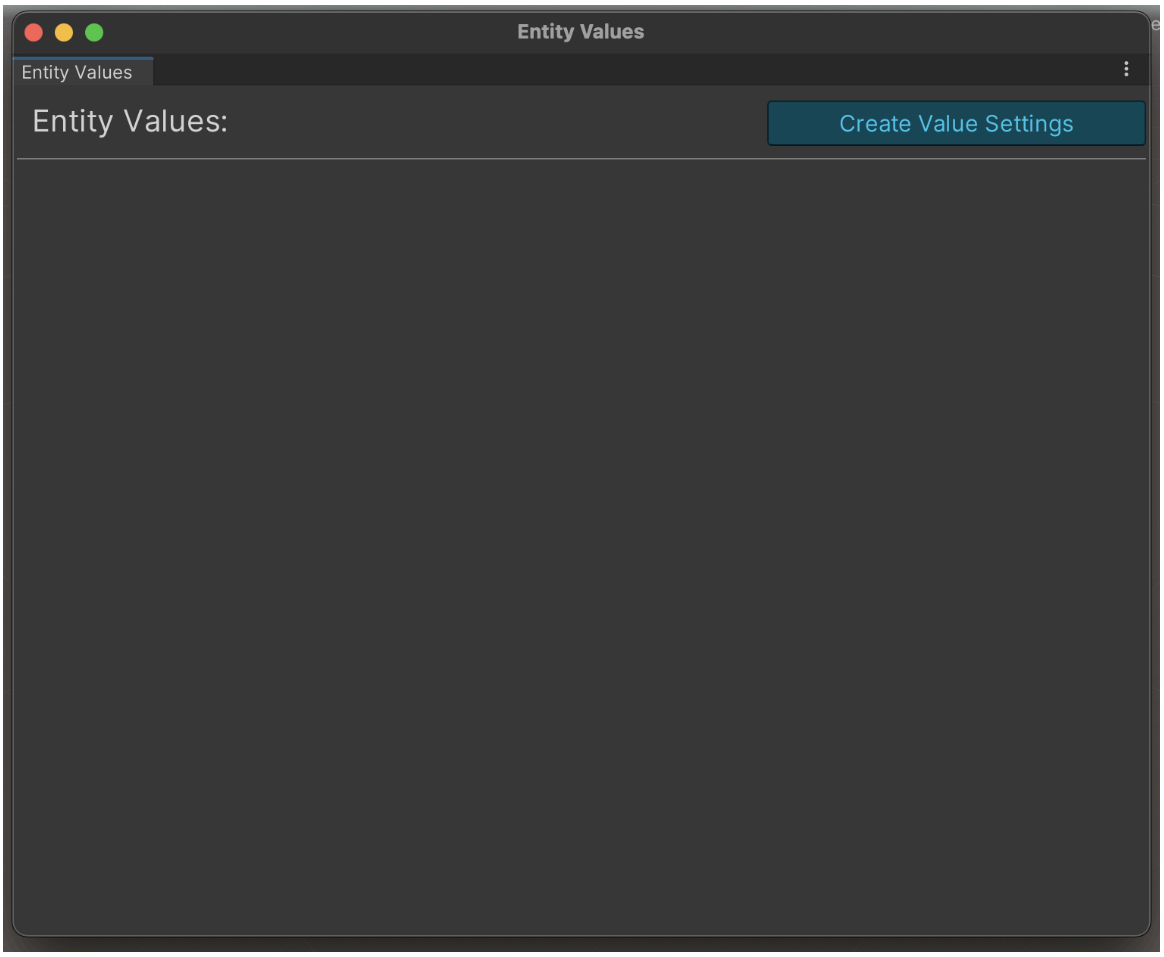
Before you start full-fledged character development, you need to get acquainted with consoles in the Atomic Framework.

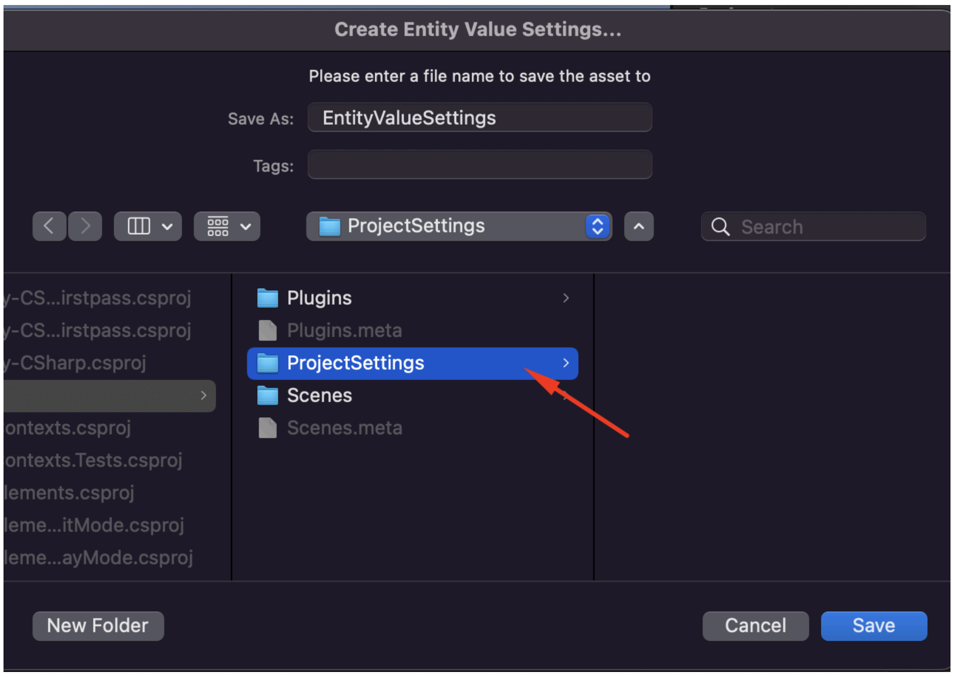
First thing, click on Window→Atomic→Entities→Show Value Console…



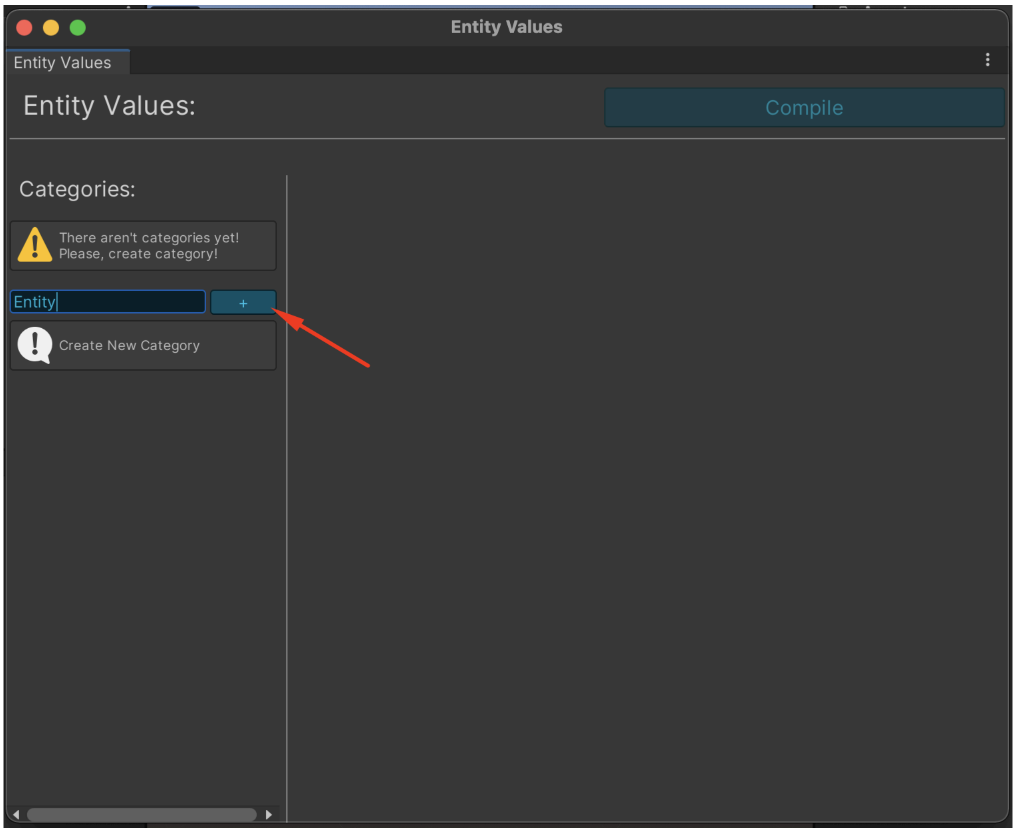
Next, click on "Create Value Settings", to create an asset that stores various identifiers for entity variables



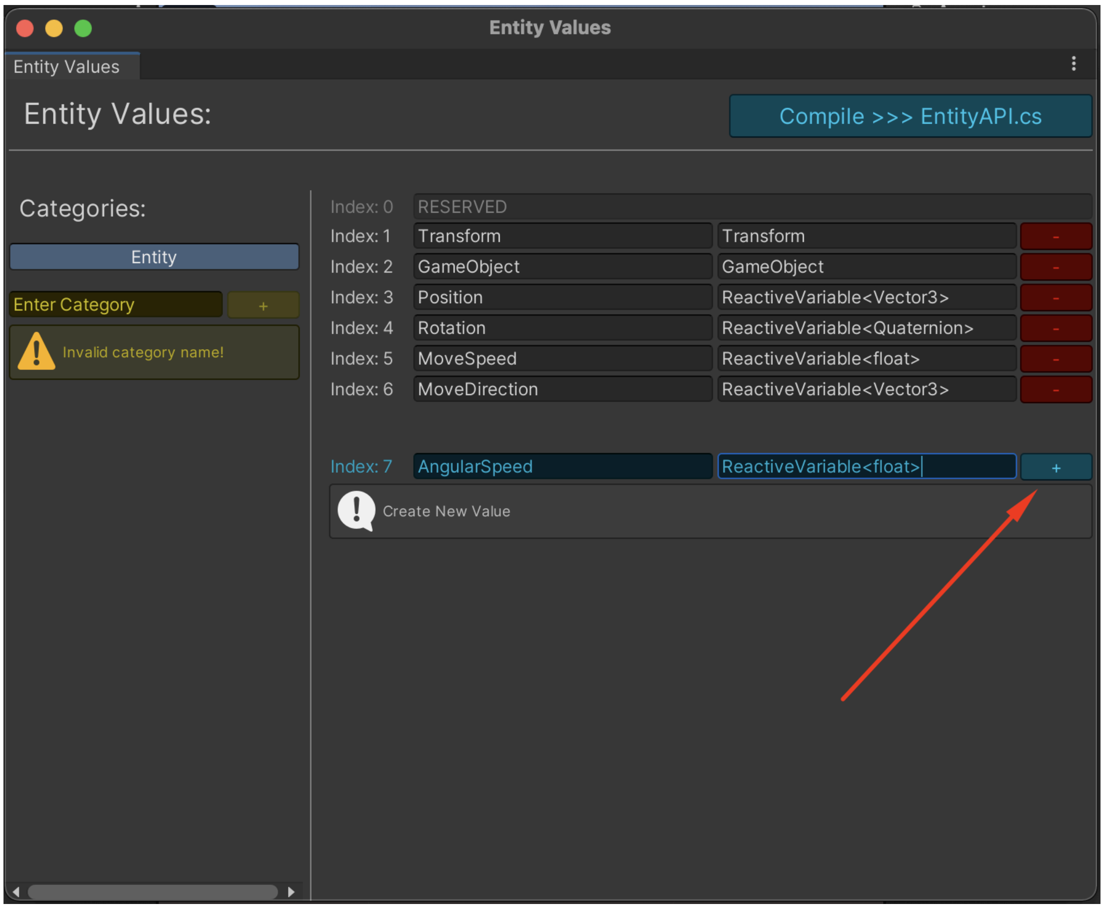
Next, create a “ProjectSettings” folder where the configuration with the IDs will be stored. Then click “Save”



After creating the asset in the “Entity Value” window, you will see the ability to add categories for your data. Categories can be distributed depending on your project . To begin with, we recommend creating a basic “Entities” category in which data identifiers will be stored as part of the demo game.

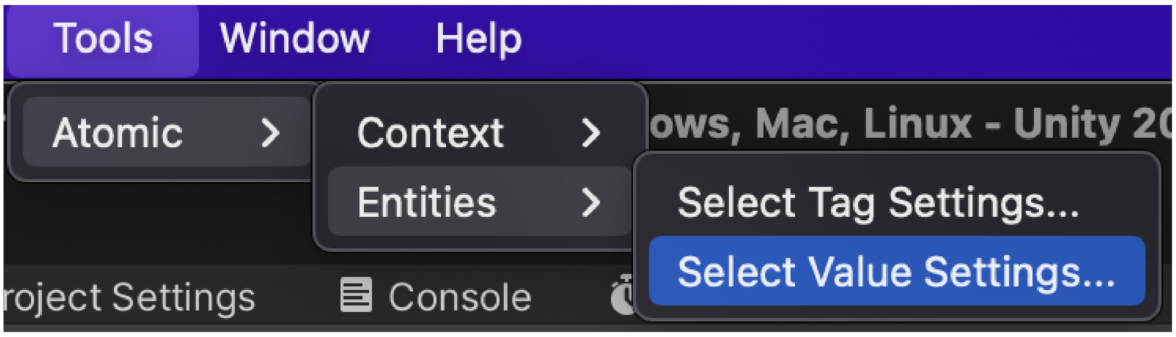


Next, a developer can create various data identifiers that can be accessed from both Unity and code. To create an identifier, you must specify the name and type.



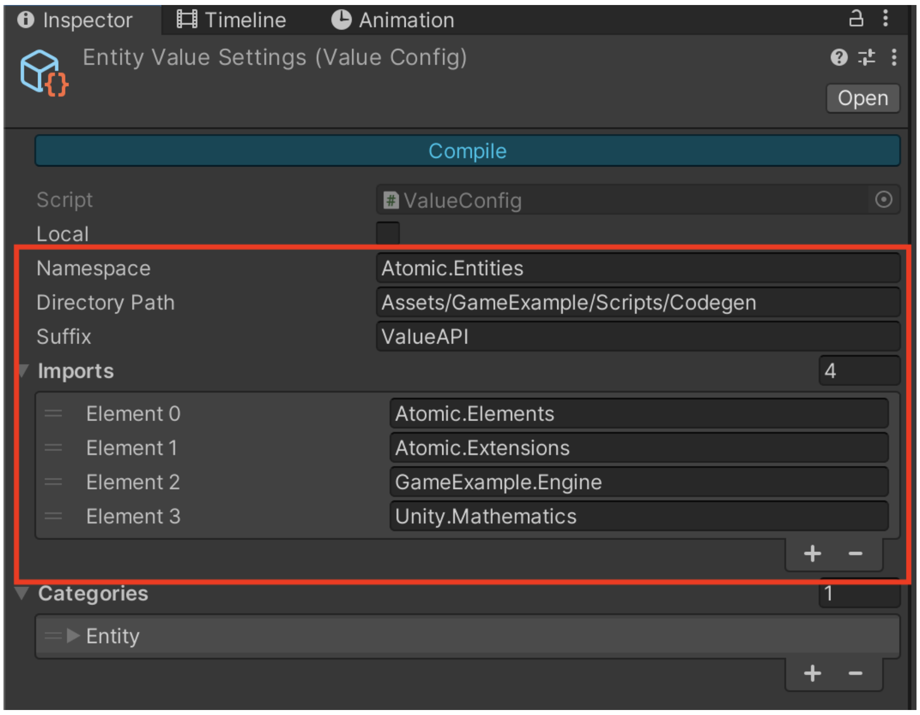
Index: 0	RESERVED	
Index: 1	Transform	Transform
Index: 2	GameObject	GameObject
Index: 3	Position	ReactiveVariable<Vector3>
Index: 4	Rotation	ReactiveVariable<Quaternion>
Index: 5	MoveSpeed	ReactiveVariable<float>
Index: 6	MoveDirection	ReactiveVariable<Vector3>
Index: 7	AngularSpeed	ReactiveVariable<float>

Then a developer can generate these identifiers by clicking the “Compile” button, but before these we recommend setting up code generation. To do this, click Tools→Entities→Select Value Settings…



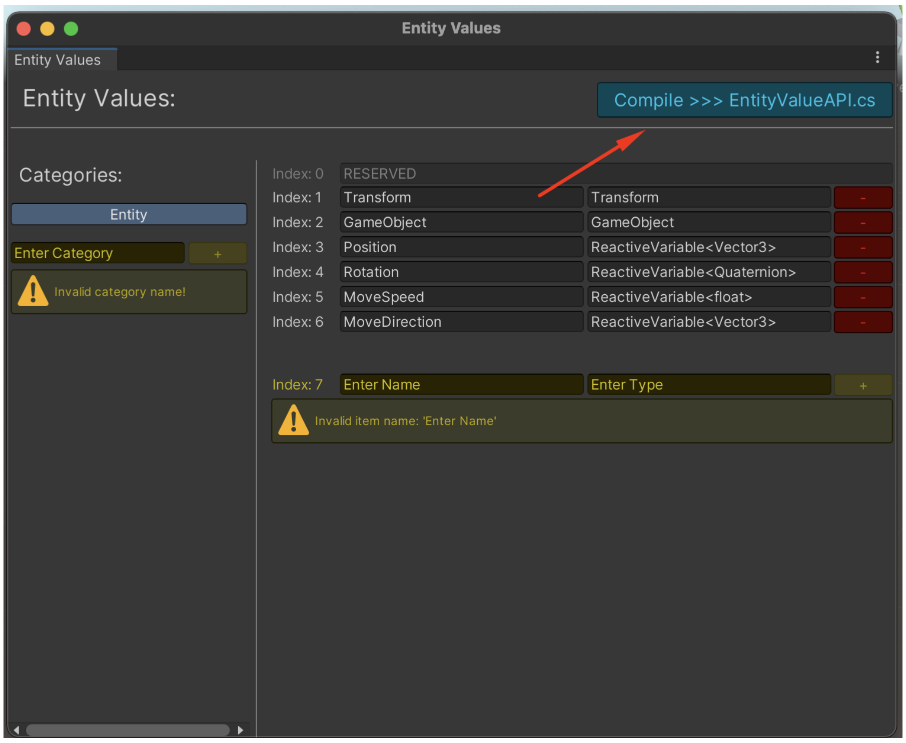
The inspector will be able to configure the following parameters of the generated class:

- namespace
- file generation path
- class suffix. The full name of the generated class is CategoryName + Suffix.
- imports



Script	ValueConfig
Local	
Namespace	Atomic.Entities
Directory Path	Assets/GameExample/Scripts/Codegen
Suffix	ValueAPI
Imports	4
Element 0	Atomic.Elements
Element 1	Atomic.Extensions
Element 2	GameExample.Engine
Element 3	Unity.Mathematics
Categories	1
Entity	

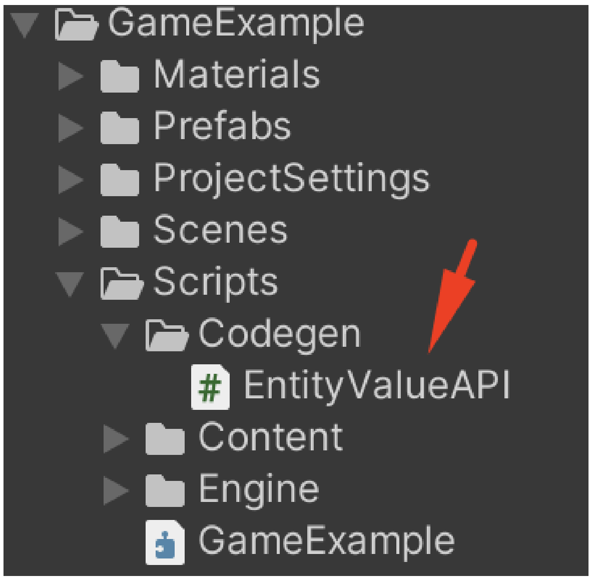
After setting up, go back to the "Entity Values" window and click the "Compile" button

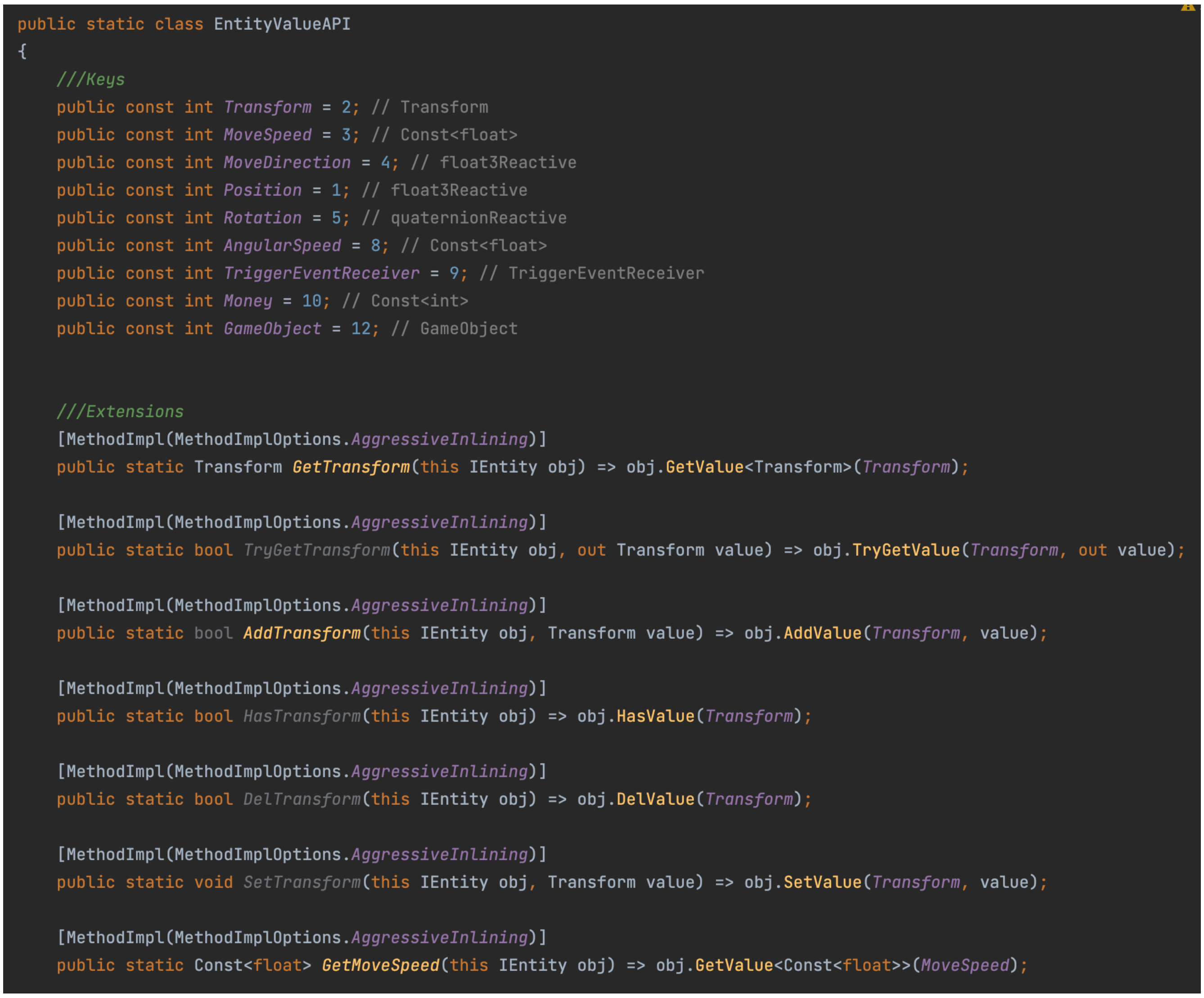


Index: 0	RESERVED	
Index: 1	Transform	Transform
Index: 2	GameObject	GameObject
Index: 3	Position	ReactiveVariable<Vector3>
Index: 4	Rotation	ReactiveVariable<Quaternion>
Index: 5	MoveSpeed	ReactiveVariable<float>
Index: 6	MoveDirection	ReactiveVariable<Vector3>
Index: 7	Enter Name	Enter Type

Invalid item name: 'Enter Name'

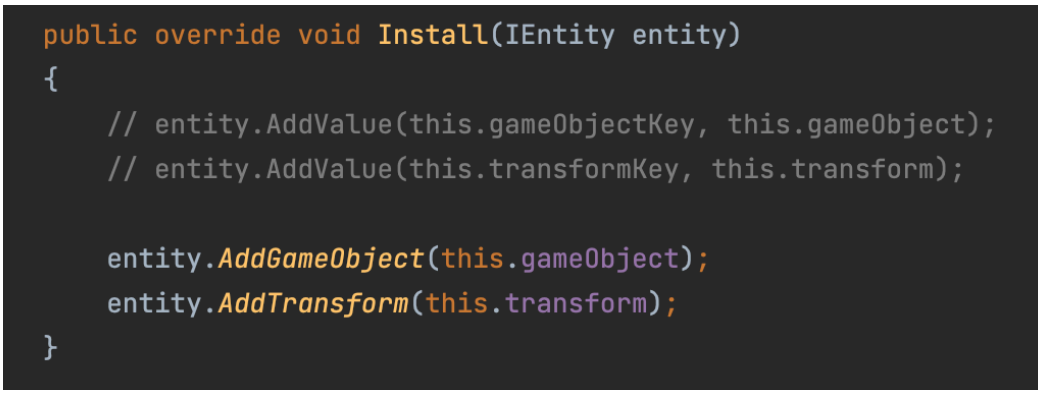
After the compilation is complete, you will see the generated CSharp Class.





```
public static class EntityValueAPI
{
    ///Keys
    public const int Transform = 2; // Transform
    public const int MoveSpeed = 3; // Const<float>
    public const int MoveDirection = 4; // float3Reactive
    public const int Position = 1; // float3Reactive
    public const int Rotation = 5; // quaternionReactive
    public const int AngularSpeed = 8; // Const<float>
    public const int TriggerEventReceiver = 9; // TriggerEventReceiver
    public const int Money = 10; // Const<int>
    public const int GameObject = 12; // GameObject


    ///Extensions
    [MethodImpl(MethodImplOptions.AggressiveInlining)]
    public static Transform GetTransform(this IEntity obj) => obj.GetValue<Transform>(Transform);

    [MethodImpl(MethodImplOptions.AggressiveInlining)]
    public static bool TryGetTransform(this IEntity obj, out Transform value) => obj.TryGetValue(Transform, out value);

    [MethodImpl(MethodImplOptions.AggressiveInlining)]
    public static bool AddTransform(this IEntity obj, Transform value) => obj.AddValue(Transform, value);

    [MethodImpl(MethodImplOptions.AggressiveInlining)]
    public static bool HasTransform(this IEntity obj) => obj.HasValue(Transform);

    [MethodImpl(MethodImplOptions.AggressiveInlining)]
    public static bool DelTransform(this IEntity obj) => obj.DelValue(Transform);

    [MethodImpl(MethodImplOptions.AggressiveInlining)]
    public static void SetTransform(this IEntity obj, Transform value) => obj.SetValue(Transform, value);

    [MethodImpl(MethodImplOptions.AggressiveInlining)]
    public static Const<float> GetMoveSpeed(this IEntity obj) => obj.GetValue<Const<float>>(MoveSpeed);
```

Using the generated extension methods will make development more comfortable and faster



```
public override void Install(IEntity entity)
{
    // entity.AddValue(this.gameObjectKey, this.gameObject);
    // entity.AddValue(this.transformKey, this.transform);

    entity.AddGameObject(this.gameObject);
    entity.AddTransform(this.transform);
}
```