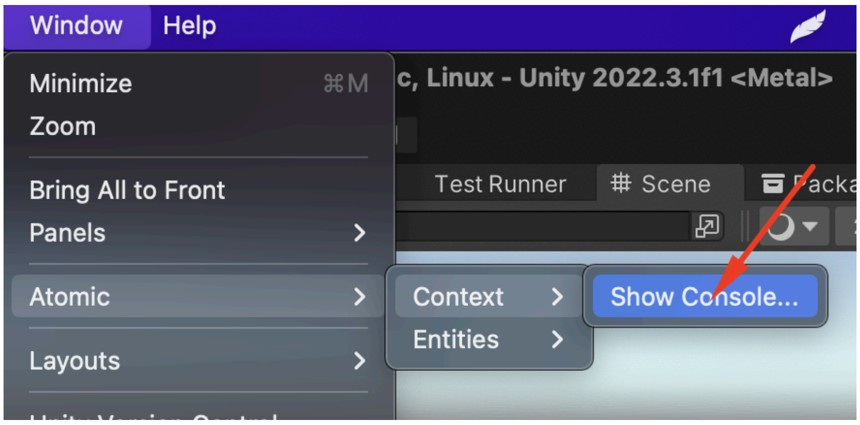
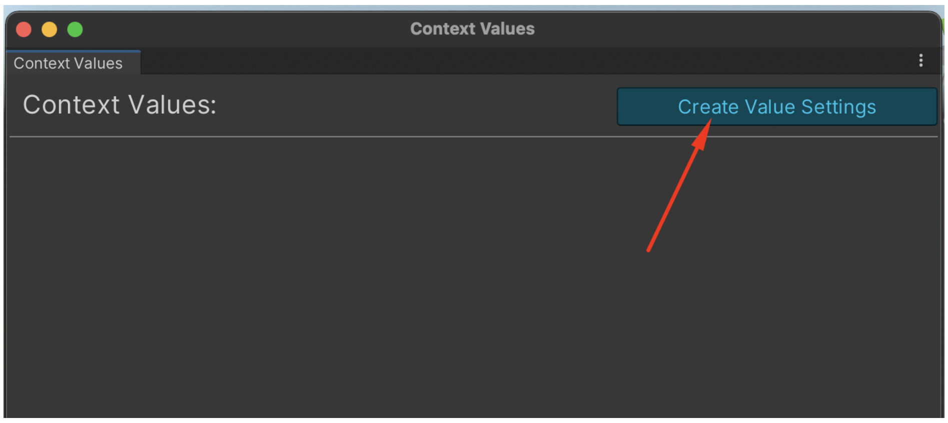
Before you start developing a game system, you need to get acquainted with consoles in the Atomic Framework.

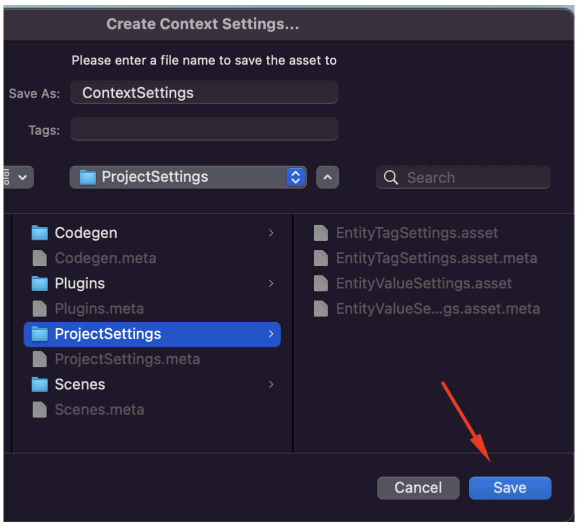
First thing, click on Window→Atomic→Context→Show Console…



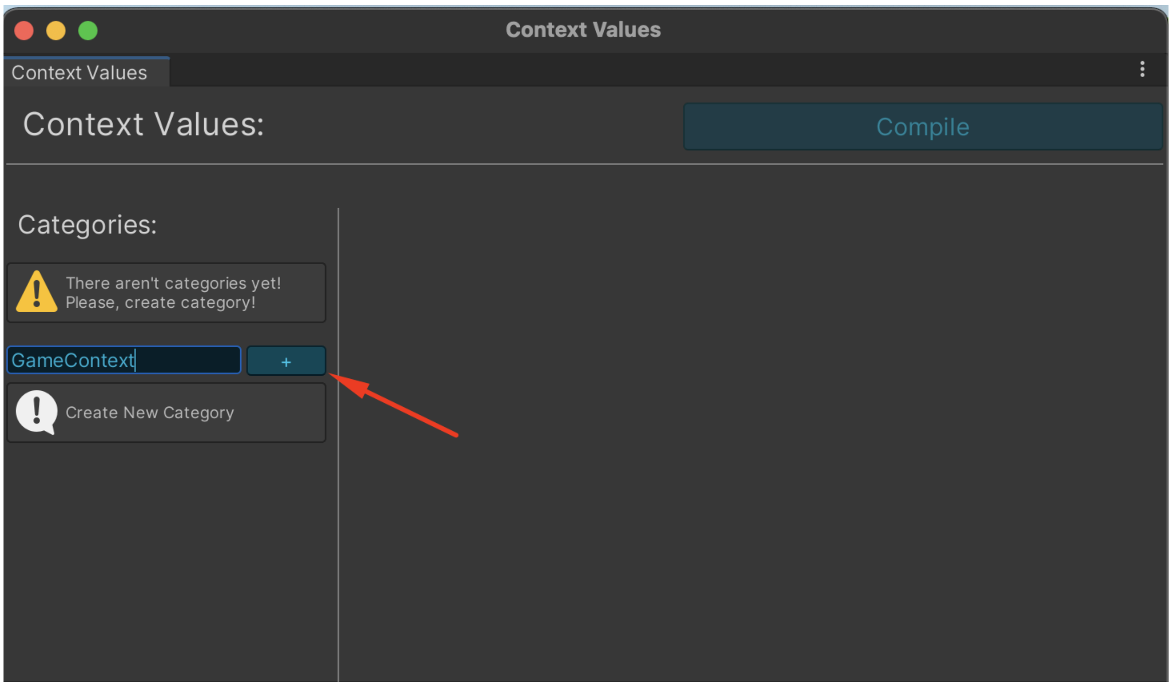
Next, click on “Create Value Settings”, to create an asset that stores various identifiers for context data



Save in “ProjectSettings” folder



After creating the asset in the “Context Values” 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 “GameContext” category in which data identifiers will be stored as part of the demo game.

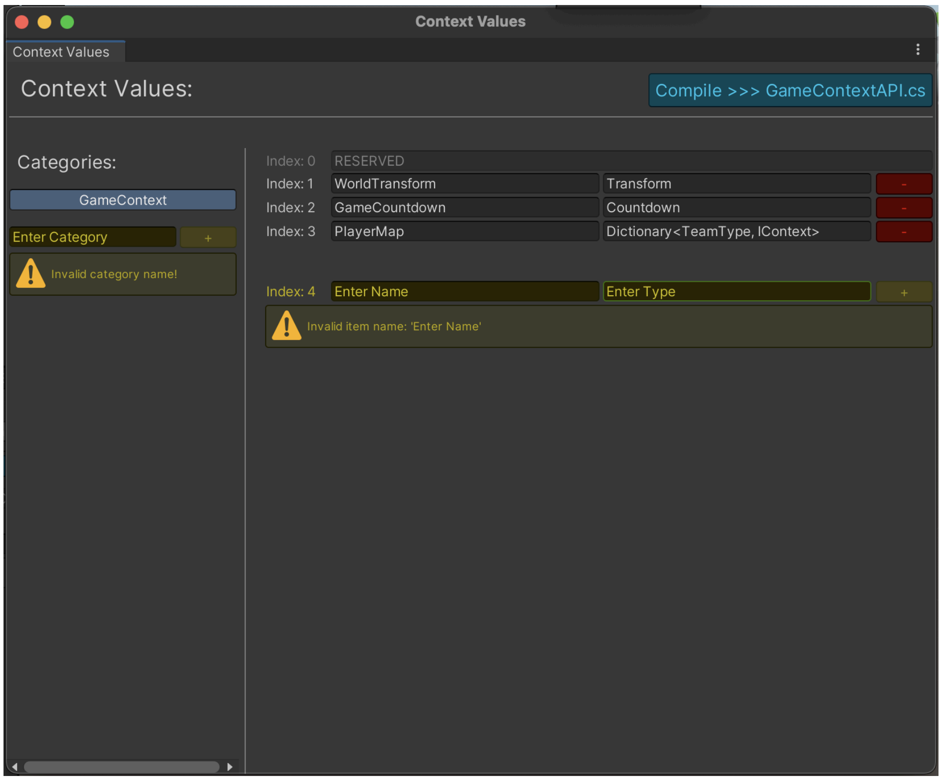


There aren't categories yet!
Please, create category!

GameContext

Create New Category

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.



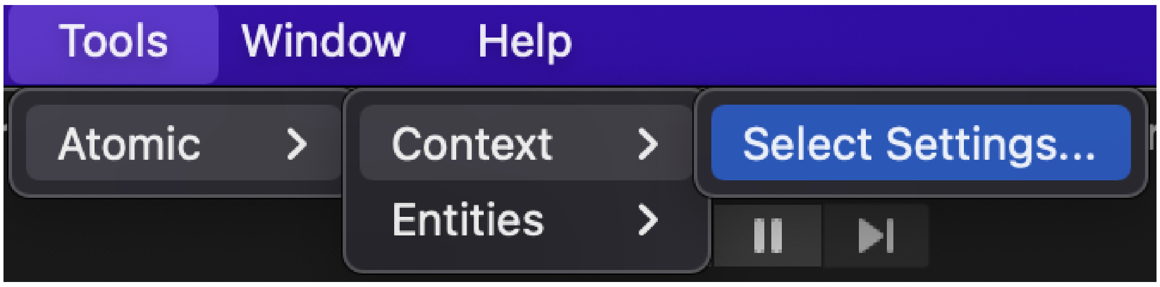
Enter Category

Invalid category name!

Index: 0	RESERVED	
Index: 1	WorldTransform	Transform
Index: 2	GameCountdown	Countdown
Index: 3	PlayerMap	Dictionary<TeamType, IContext>
Index: 4	Enter Name	Enter Type

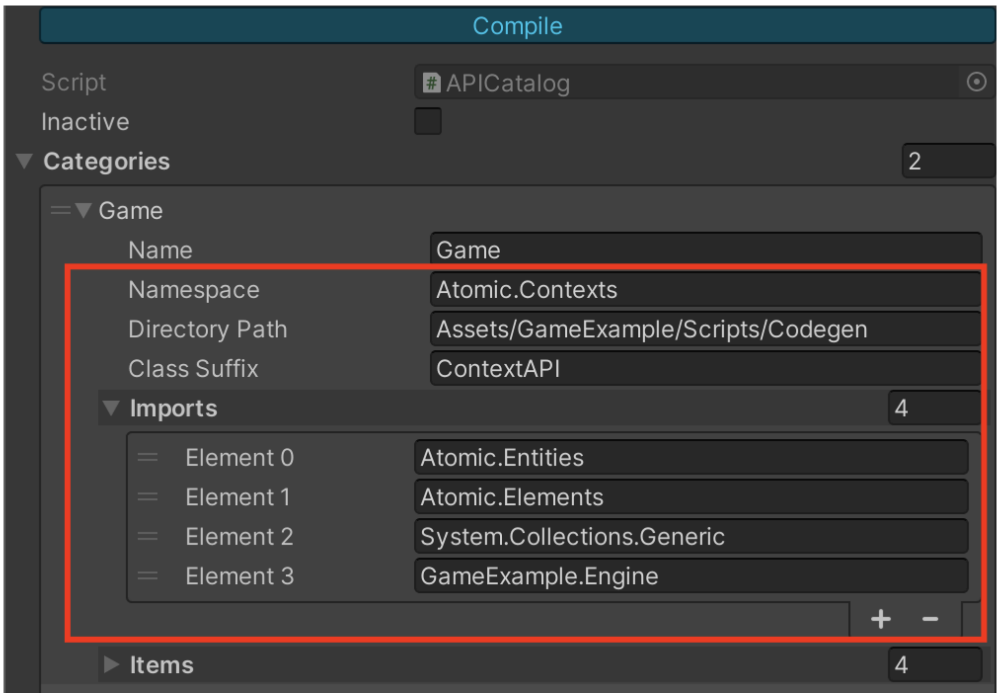
Invalid item name: 'Enter Name'

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→Context→Select Settings…



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

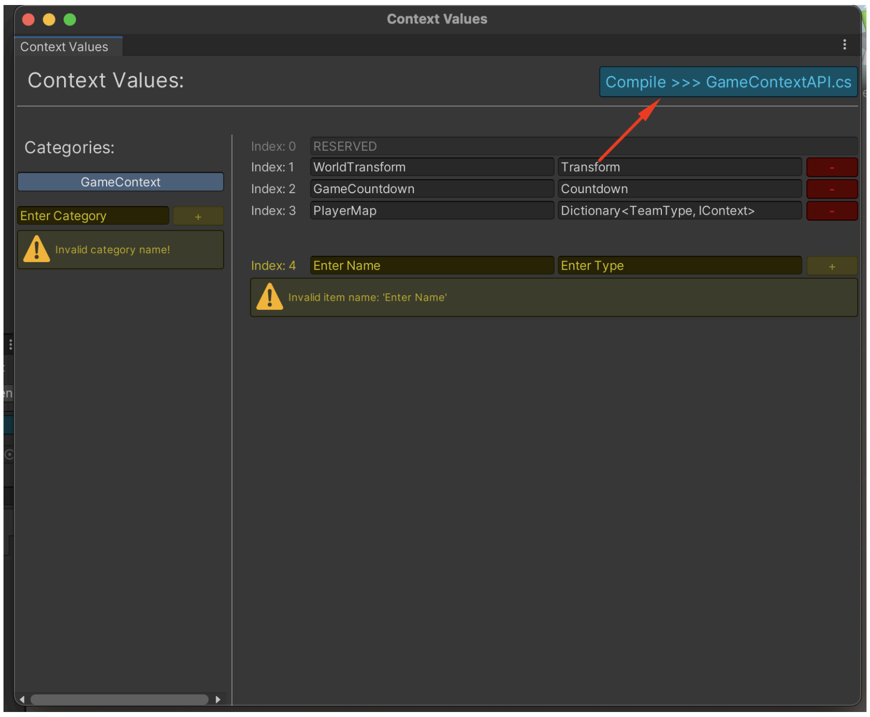
- namespace
- file generation path
- class suffix. The full name of the generated class is CategoryName + Suffix.
- imports



Compile

Script	APICatalog
Inactive	
Categories	2
Game	
Name	Game
Namespace	Atomic.Contexts
Directory Path	Assets/GameExample/Scripts/Codegen
Class Suffix	ContextAPI
Imports	4
Element 0	Atomic.Entities
Element 1	Atomic.Elements
Element 2	System.Collections.Generic
Element 3	GameExample.Engine
Items	4

After setting up, go back to the “Context Values” window and click the “Compile” button



Context Values

Context Values:

Compile >>> GameContextAPI.cs

Categories:

GameContext

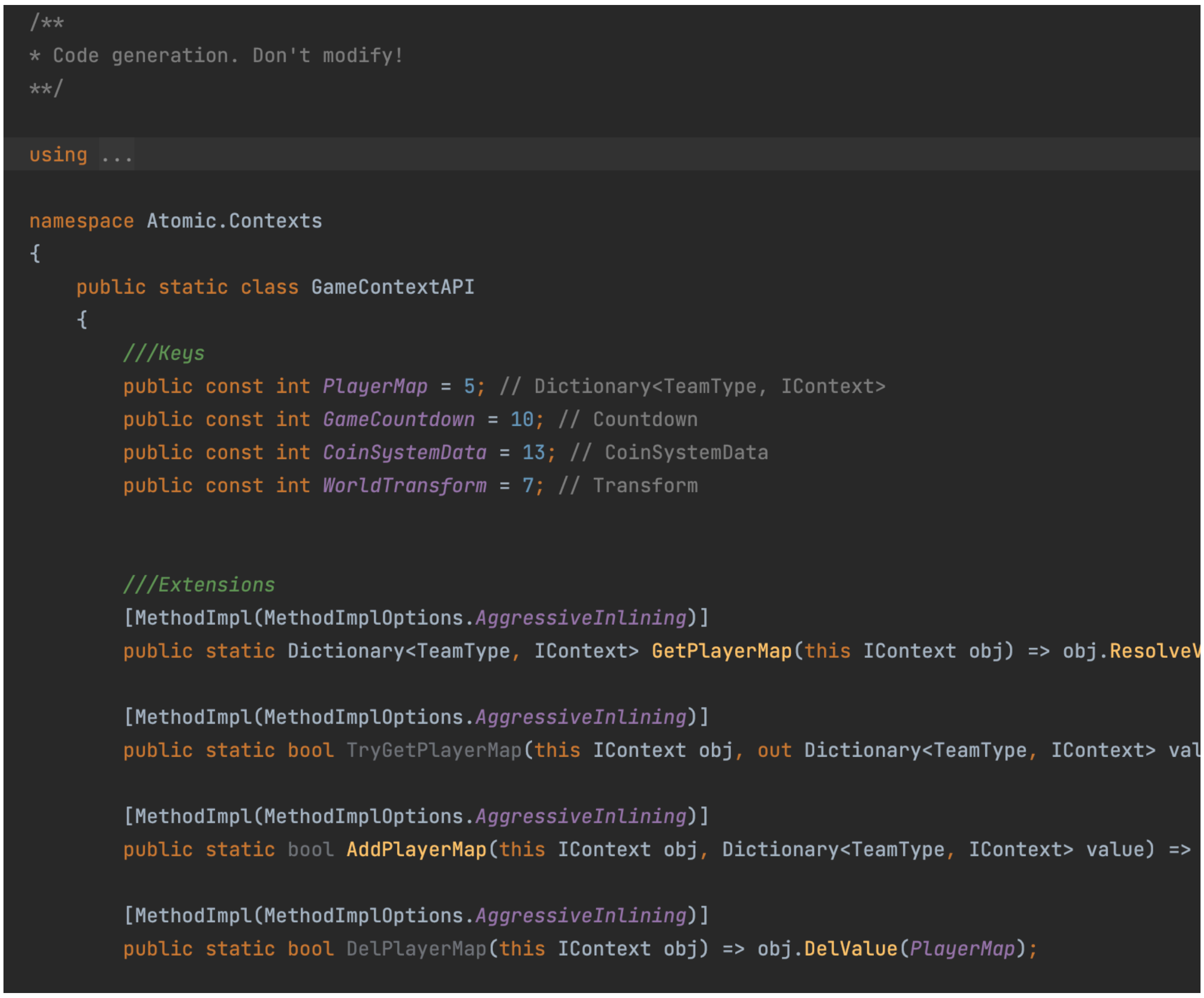
Enter Category

Invalid category name!

Index: 0	RESERVED	
Index: 1	WorldTransform	Transform
Index: 2	GameCountdown	Countdown
Index: 3	PlayerMap	Dictionary<TeamType, IContext>
Index: 4	Enter Name	Enter Type

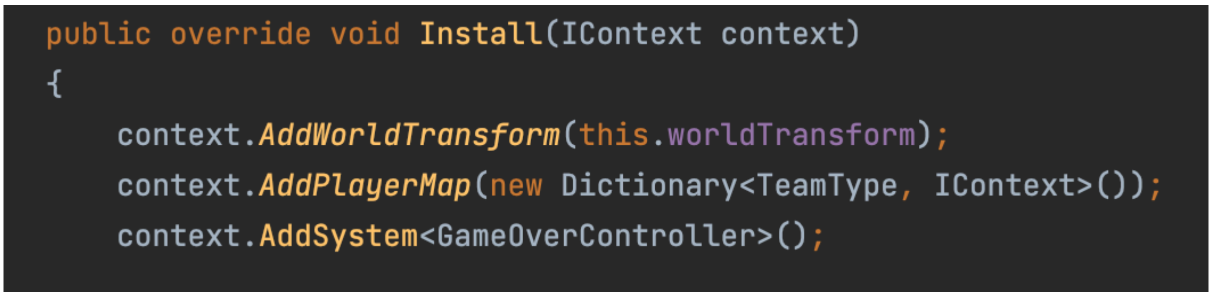
Invalid item name: 'Enter Name'

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



```
/**
* Code generation. Don't modify!
**/

using ...

namespace Atomic.Contexts
{
    public static class GameContextAPI
    {
        ///Keys
        public const int PlayerMap = 5; // Dictionary<TeamType, IContext>
        public const int GameCountdown = 10; // Countdown
        public const int CoinSystemData = 13; // CoinSystemData
        public const int WorldTransform = 7; // Transform


        ///Extensions
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static Dictionary<TeamType, IContext> GetPlayerMap(this IContext obj) => obj.ResolveV

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static bool TryGetPlayerMap(this IContext obj, out Dictionary<TeamType, IContext> val

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static bool AddPlayerMap(this IContext obj, Dictionary<TeamType, IContext> value) =>

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static bool DelPlayerMap(this IContext obj) => obj.DelValue(PlayerMap);
```

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



```
public override void Install(IContext context)
{
    context.AddWorldTransform(this.worldTransform);
    context.AddPlayerMap(new Dictionary<TeamType, IContext>());
    context.AddSystem<GameOverController>();
```